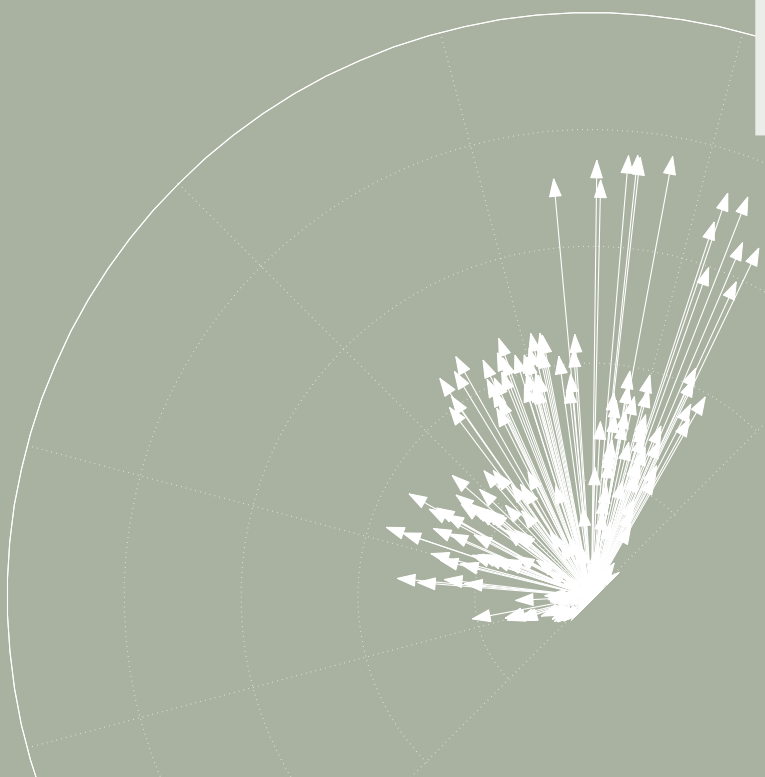
MANUAL DE COMPUTAÇÃO EVOLUTIVA E METAHEURÍSTICA

ANTÓNIO GASPAR-CUNHA
RICARDO TAKAHASHI
CARLOS HENGGELER ANTUNES
COORDENADORES



IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS

EDITORA ufmg

(Página deixada propositadamente em branco)

E N S I N O

IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS

CO-EDIÇÃO

Imprensa da Universidade de Coimbra
Email: imprensauc@ci.uc.pt
URL: http://www.uc.pt/imprensa_uc
Vendas online http://www.livrariadaimprensa.com

Editora da Universidade Federal de Minas Gerais
URL: http://www.editoraufmg.com.br/

CONCEPÇÃO GRÁFICA

António Barros

INFOGRAFIA DA CAPA

Carlos Costa

EXECUÇÃO GRÁFICA

Sersilito • Maia

ISBN

978-989-26-0150-2 (IUC)
978-85-7041-950-7 (EDITORAufmg)

ISBN Digital

978-989-26-0583-8

DOI

http://dx.doi.org/10.14195/978-989-26-0583-8

DEPÓSITO LEGAL

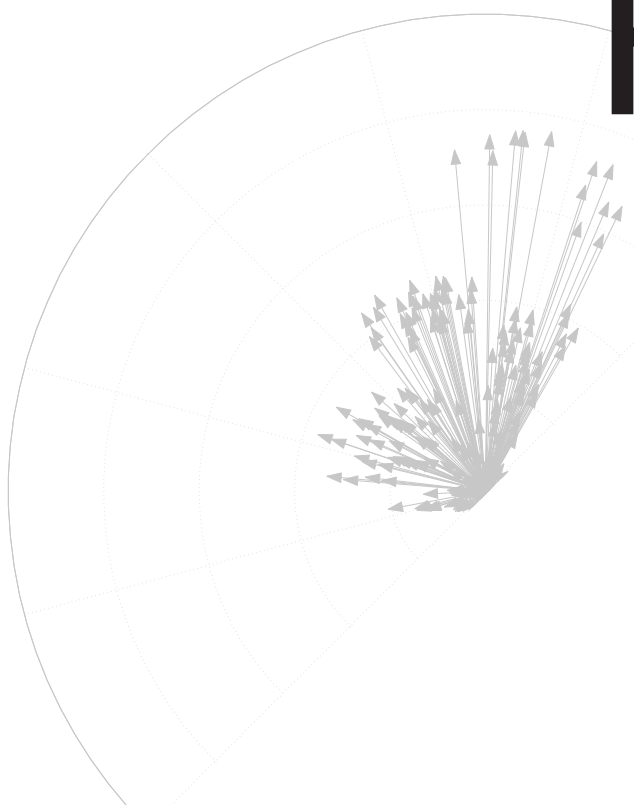
344533/12

A edição desta obra contou com o apoio da Universidade do Minho e do Instituto de Engenharia de Sistemas e Computadores de Coimbra - INESC Coimbra.

MANUAL DE

COMPUTAÇÃO EVOLUTIVA E META HEURÍSTICA

ANTÓNIO GASPAR-CUNHA
RICARDO TAKAHASHI
CARLOS HENGGELER ANTUNES
COORDENADORES



IMPRENSA DA
UNIVERSIDADE
DE COIMBRA
COIMBRA
UNIVERSITY
PRESS

EDITORA ufmg

(Página deixada propositadamente em branco)

Sumário

III Tópicos Avançados

Prefácio

As técnicas computacionais que são hoje denominadas por Computação Evolutiva e por Metaheurísticas se desenvolveram, de maneira relativamente independente, durante os últimos 40 anos do século XX, no seio de duas comunidades científicas que mantiveram relativamente pouco contato ao longo desse período. Durante esse tempo, ambos os conjuntos de técnicas se consolidaram, sendo hoje reconhecidos como parte integrante do repertório fundamental de ferramentas da Computação e da Engenharia que possibilitam a síntese de muitos dos sistemas tecnológicos hoje existentes. Apenas no decorrer da última década do século XX se formou, nas respectivas comunidades científicas, uma consciência das conexões existentes entre esses dois corpos de conhecimento, que partilham muitos dos seus princípios e fundamentos. Em 2008, um grupo de colegas Portugueses e Brasileiros que trabalham nas áreas de algoritmos evolutivos e metaheurísticas decidiu promover a organização de uma Escola Luso-Brasileira de Computação Evolutiva (ELBCE). O primeiro evento ocorreu em Outubro de 2009 na Universidade Federal de Minas Gerais, em Belo Horizonte, e a segunda escola em Junho de 2010 na Universidade do Minho, em Guimarães. O terceiro evento ocorreu na Universidade de São Paulo, no Campus I da cidade de São Carlos, em Abril de 2012. Os temas cobertos nesses eventos apresentaram um panorama abrangente dessas áreas do conhecimento. Além disso, ficou claro o efeito de uma apresentação articulada desses temas para potencializar a compreensão dos seus aspectos mais fundamentais.

O grande interesse manifestado pelos pesquisadores envolvidos na lecionação das aulas, bem como o grande número de estudantes que aderiu às três edições do evento, sugeriram a pertinência da escrita deste livro que inclui um vasto leque de temas relacionados com a computação evolutiva e metaheurísticas. O presente livro foi escrito com o objetivo de constituir uma obra de referência em Língua Portuguesa, abrangendo os níveis de graduação e pós-graduação do nosso ensino universitário e politécnico. Além disso, mesmo na literatura técnica em geral, não temos conhecimento de obra similar, escrita em qualquer língua, que proponha tal recorte temático. Gostaríamos de agradecer aos autores dos vários capítulos pela rapidez na sua resposta à solicitação feita bem como pelo rigor científico colocado na sua escrita. Gostaríamos de agradecer também aos revisores anónimos que permitiram melhorar a qualidade deste trabalho e à Imprensa da Universidade de Coimbra que se dispôs a investir neste projeto.

Abril de 2012

António Gaspar-Cunha
Ricardo H. C. Takahashi
Carlos Henggeler Antunes

CAPÍTULO 1

Introdução

Ricardo H. C. Takahashi [*] *António Gaspar-Cunha* [**]

[*] *Departamento de Matemática*
Universidade Federal de Minas Gerais

[**] *Instituto de Polímeros e Compósitos / I3N*
Universidade do Minho

Este livro trata dos temas da Computação Evolutiva e da Metaheurística, considerando principalmente o contexto da Otimização. Neste capítulo, uma breve discussão introdutória a esses três assuntos é apresentada, procurando primeiro dar uma noção sobre o que é cada um desses três campos científicos. A seguir, procura-se mostrar como esses assuntos se relacionam, e quais razões têm levado a uma tendência de crescente convergência entre eles. Ao longo da discussão, este capítulo procura apresentar um panorama dos problemas de fundo que serão abordados nos demais capítulos deste livro.

1. Otimização

A Otimização é o campo de conhecimentos cujas técnicas visam determinar os extremos (máximos ou mínimos) de funções, em domínios determinados[1]. De maneira concreta, podemos pensar que uma

[1] O leitor deve estar atento para o fato de que esta definição pretende dar uma noção geral do que significa Otimização, mas não pretende conter, de maneira exaustiva, toda a diversidade e complexidade desse campo do conhecimento.

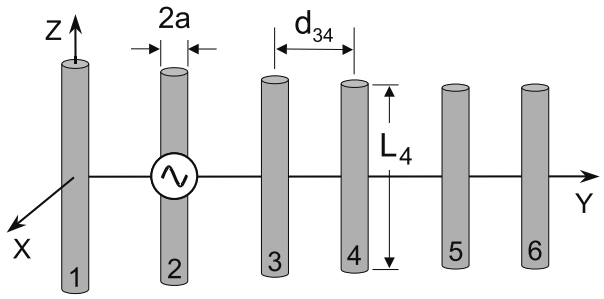


Figura 1.1: Representação esquemática de uma antena de seis elementos. Na prática, uma antena real corresponde a uma estrutura metálica com geometria igual à do diagrama.

função cujo extremo se quer descobrir representa algum fator de mérito relacionado com um sistema que se deseja construir ou analisar. Se o sistema for, por exemplo, um dispositivo tecnológico que se pretende construir, a função pode representar seu custo de construção, ou sua eficiência energética, ou sua taxa de falhas esperada, ou outros fatores de mérito ainda. Deve ficar claro qual é o motivo pelo qual se deseja determinar o extremo de uma função com tal significado: o extremo (máximo ou mínimo) representa a melhor maneira possível de construir o sistema que está sendo projetado – no sentido de ser o projeto de mínimo custo, ou de máxima eficiência energética, ou de mínima taxa de falhas, etc. O domínio no qual é executada a otimização[2], ou seja, a região do espaço de variáveis da função na qual se procura determinar seu extremo, corresponde ao conjunto das alternativas diferentes disponíveis para construir o sistema em questão. Desenvolveremos essas idéias a partir de dois exemplos.

Exemplo: otimização com variáveis contínuas

Como primeiro exemplo, considere-se o problema de projetar uma antena, como a da Figura 1.1. Projetar essa antena significa escolher os tamanhos dos seis elementos da antena e as cinco distâncias que separam esses elementos entre si. São portanto onze valores a serem representados pelo vetor $\mathbf{x}$, o qual seria dado, nesse caso, por:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{11} \end{bmatrix}, \tag{1.1}$$

sendo as variáveis x_1 a x_6 correspondentes aos comprimentos dos elementos, e as variáveis x_7 a x_{11} correspondentes às distâncias entre eles.

O dispositivo representado na Figura 1.1 pode ser uma antena para transmissão de sinais que deva enviar dados em uma direção específica. Um dos critérios de mérito que frequentemente são empregados para avaliar o desempenho de antenas de transmissão desse tipo é a sua *razão frente-costas*. Essa função representa a razão entre a energia que a antena transmite na direção desejada e a energia transmitida para outras direções do espaço. Essa função, portanto, deve assumir valores os maiores possíveis, para que a antena apresente bom desempenho. O domínio no qual será realizada a otimização corresponde à região do espaço das variáveis de decisão (as variáveis constituintes do vetor $\mathbf{x}$) na qual essas variáveis assumem valores considerados praticáveis. No caso de uma antena dessas, para trabalhar em frequências da ordem das centenas de MHz, as variáveis que representam os comprimentos dos elementos e as distâncias entre esses elementos usualmente poderão variar entre zero

[2] A palavra *Otimização*, com inicial maiúscula, diz respeito ao campo do conhecimento, enquanto a palavra *otimização*, com inicial minúscula, diz respeito ao procedimento de determinação do extremo de uma função.

e algumas dezenas de centímetros. Portanto, um processo de otimização que irá procurar a melhor antena dentre todas as possíveis poderá restringir sua busca, por exemplo, à região do espaço de variáveis na qual todas essas variáveis tenham valor menor que 0,5 metro – dessa forma estabelecendo-se o domínio no qual será realizada a otimização. Dessa forma aparecem as seguintes expressões que definem esse domínio:

$$\begin{aligned} 0 \leq x_1 \leq 0,5 \\ 0 \leq x_2 \leq 0,5 \\ \vdots \\ 0 \leq x_{11} \leq 0,5 \end{aligned} \tag{1.2}$$

Além da condição de que cada uma das variáveis que representam as dimensões da antena deva ter valor menor que 1 metro, é possível que haja outro tipo de condição sobre as variáveis decorrente da situação prática em cada caso. Por exemplo, pode ser necessário que a antena, uma vez montada, caiba dentro de uma determinada caixa na qual ela deverá ser transportada. Assim, além de se restringir o tamanho de cada elemento, é necessário também restringir a soma das distâncias entre os elementos, de forma que a antena montada não exceda o tamanho máximo permitido, por exemplo 1,5 metro. Nesse caso, aparece uma outra desigualdade que deve ser obedecida, do tipo:

$$x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 1,5 \tag{1.3}$$

Chamemos de $f(\mathbf{x})$ a função que atribui a uma antena cujas variáveis de decisão estejam especificados no vetor $\mathbf{x}$ o valor de sua razão frente-costas. Essa é a chamada *função-objetivo* do problema. Na prática, os valores dessa função para os diferentes valores que o vetor $\mathbf{x}$ pode assumir serão calculados por meio de um programa que irá simular a antena, dados os parâmetros $\mathbf{x}$, sendo esse programa baseado na teoria eletromagnética e em métodos numéricos diversos de interpolação, integração, etc. O problema de otimização é representado, de maneira sintética, por:

$$\begin{aligned} &\mathbf{x}^* = \arg\max f(\mathbf{x}) \\ &\text{sujeito a:} \begin{cases} x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 1,5 \\ 0 \leq x_1 \leq 0,5 \\ 0 \leq x_2 \leq 0,5 \\ \quad \vdots \\ 0 \leq x_{11} \leq 0,5 \end{cases} \end{aligned} \tag{1.4}$$

Pode-se ler essa expressão da seguinte forma: $\mathbf{x}^*$, que é o vetor de variáveis de decisão que soluciona o problema, é o argumento que maximiza a função $f(\mathbf{x})$ (ou seja, é o conjunto de valores das variáveis de decisão que faz a função assumir seu máximo valor possível), considerando os valores das variáveis de decisão que satisfaçam ao conjunto de desigualdades às quais o problema encontra-se sujeito. Essas desigualdades que especificam os valores permitidos para as variáveis de decisão são denominadas *restrições* do problema de otimização, e a região do espaço de variáveis de decisão cujos pontos satisfazem as restrições é chamada *região factível*, ou *conjunto factível* do problema.

Algumas características desse problema de otimização merecem ser destacadas:

- As variáveis deste problema são de natureza contínua, ou seja, podem assumir quaisquer valores dentro de intervalos. A função-objetivo $f(\cdot)$ é uma função não-linear das variáveis presentes no vetor de variáveis de decisão, $\mathbf{x}$. Para cada instância (ou seja, conjunto específico de valores nas coordenadas) do vetor $\mathbf{x}$, a função-objetivo assume um valor correspondente, provavelmente mantendo uma certa continuidade. Isso significa que valores muito próximos de $\mathbf{x}$ devem produzir valores parecidos de $f(\mathbf{x})$.

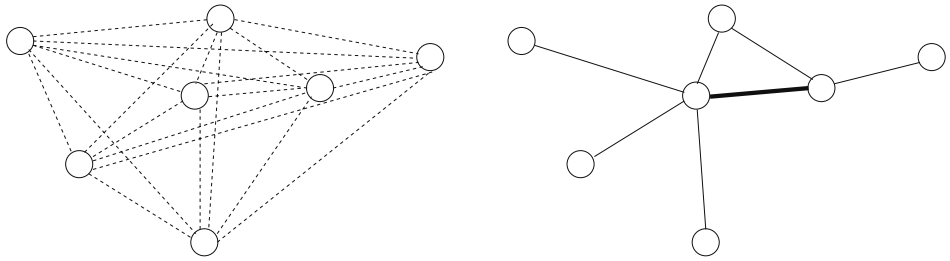


Figura 1.2: Problema de projeto de uma rede de comunicações. Na sub-figura da esquerda, são mostradas todas as possíveis conexões ligando pares de nós. A sub-figura da direita mostra uma rede contendo um conjunto de conexões que liga todos os nós entre si, contendo ainda um caminho alternativo para o caso de falha do arco por onde passa o maior fluxo de dados. Esse arco é representado com traço mais grosso que os demais.

- Não encontra-se disponível uma expressão explícita para a função $f(\mathbf{x})$. O cálculo dessa função é feito por uma simulação computacional das leis físicas envolvidas no problema. Dessa forma, qualquer que seja o algoritmo de otimização que vá tentar determinar um extremo dessa função, tal algoritmo não poderá pressupor a utilização de uma expressão da função. A única informação que se pode empregar é a avaliação da função (ou seja o valor que ela assume) em determinados pontos do espaço de variáveis, para os quais se executou o algoritmo de cálculo da função.

- Cada cálculo de função-objetivo, para cada instância do vetor de variáveis de decisão, envolve um processamento computacional relativamente complexo, que provavelmente irá consumir tempo de execução não desprezível. O processo de otimização, portanto, deve usar com parcimônia a possibilidade de avaliar essa função. Isso significa que está fora de questão, por exemplo, variar cada uma das variáveis de centímetro em centímetro, e avaliar a função-objetivo para cada uma das combinações resultantes, pois isso conduziria a executar o algoritmo de cálculo da função nada menos que 51^{11} vezes[3]. Mesmo após tal esforço computacional gigantesco, o resultado ainda seria muito pobre sob o ponto de vista do projeto da antena, pois a precisão necessária para que a qualidade da antena fosse garantida deveria ser da ordem dos décimos de milímetro, e não apenas dos centímetros.

Exemplo: otimização com variáveis discretas

Para ilustrar o conceito de Otimização, convém apresentar mais um exemplo, com características diferentes. Considere-se agora o problema de projeto de uma rede de comunicação de dados. O problema, representado na Figura 1.2, consiste em escolher, dentre todas as possíveis conexões que podem ligar um conjunto de nós, aquelas conexões a serem efetivamente construídas para constituir uma rede de comunicações. Pode-se imaginar um contexto em que tal sistema significasse, por exemplo, o conjunto de conexões de fibra ótica usado para interligar as redes de telefonia de cidades diversas (nesse caso, cada cidade seria um nó).

O problema de projeto que se deseja resolver é definido como: projetar uma rede de baixo custo de construção, que seja capaz de interligar todos os nós, garantindo ainda que exista uma rota alternativa, em caso de falha da conexão por onde passa o maior fluxo de dados.

[3] Para se ter uma idéia do que isto significa, caso cada avaliação de função objetivo durasse 1 segundo, a avaliação de todos esses valores demoraria aproximadamente 193 bilhões de anos — muito mais do que qualquer fabricante de antenas estaria disposto a esperar...

Como no exemplo anterior, o problema de projeto pode ser formulado como um procedimento de otimização. Agora, cada variável de decisão corresponde à inclusão ou não de cada um dos $N = n(n-1)/2$ arcos que podem ligar os n nós. Adota-se por convenção que uma variável $x_{i,j}$, que representa a conexão entre o nó i e o nó j, assume o valor 1 se tal conexão for colocada, e o valor 0 se não for colocada na rede, e o vetor de variáveis de decisão fica:

$$\mathbf{x} = \begin{bmatrix} x_{1,2} \\ \vdots \\ x_{1,n} \\ x_{2,3} \\ \vdots \\ x_{2,n} \\ \vdots \\ x_{(n-1),n} \end{bmatrix}. \tag{1.5}$$

O custo de instalação do arco que conecta o nó i ao nó j é definido como o valor $c_{i,j}$. Dessa forma, a função-objetivo que representa o custo de construção da rede é expresso por:

$$f(\mathbf{x}) = c_{1,2}x_{1,2} + c_{1,3}x_{1,3} + \ldots + c_{(n-1),n}x_{(n-1),n} = \sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} c_{ij}x_{ij}, \tag{1.6}$$

ou seja, é a soma dos custos dos arcos que forem instalados na rede (aqueles para os quais $x_{i,j} = 1$). As restrições do problema, que como no caso anterior vão dizer quais vetores de variáveis de decisão são admissíveis, desta vez não são representadas por expressões simples: são admissíveis os vetores que representarem grafos conexos (redes em que exista um caminho entre todos os nós) mesmo com a retirada do arco de maior fluxo. Essas restrições, portanto, não serão verificadas pela avaliação do valor de uma expressão matemática, mas sim pela execução de algoritmos de teste que irão de alguma forma "percorrer" a rede, examinando o atendimento de tais condicionantes. Defina-se a função $g(\mathbf{x})$ de maneira a expressar o resultado dessa verificação, da seguinte forma:

$$g(\mathbf{x}) = \begin{cases} 1\ , \text{ se a rede descrita pelo vetor } \mathbf{x} \text{ ficar desconexa} \\ \quad \text{após a retirada do arco de maior fluxo} \\ \\ -1\ , \text{ se a rede descrita pelo vetor } \mathbf{x} \text{ permanecer conexa} \\ \quad \text{após a retirada do arco de maior fluxo} \end{cases}. \tag{1.7}$$

Com essa representação para as restrições, o problema de otimização pode ser agora formulado da seguinte maneira:

$$\begin{aligned} &\mathbf{x}^* = \arg\min f(\mathbf{x}) \\ &\text{sujeito a: } g(\mathbf{x}) < 0 \end{aligned} \tag{1.8}$$

Novamente, $\mathbf{x}^*$ corresponde à instância do vetor de variáveis de decisão que produz o valor extremo da função-objetivo (agora, ao contrário do exemplo anterior, o extremo procurado é o mínimo, e não o máximo), ou seja, é o *argumento* que minimiza a função $f(\mathbf{x})$ garantindo o atendimento da restrição $g(\mathbf{x}) < 0$.

O problema de otimização deste exemplo possui as seguintes características:

- As variáveis de decisão agora são de natureza discreta, assumindo apenas os valores 0 ou 1. Devido a isso, não é possível utilizar nenhuma propriedade característica dos espaços contínuos (tais como a existência de vizinhanças arbitrariamente pequenas, a continuidade de funções, o conceito de "direção", etc) no processo de otimização.

- O cálculo do valor da função $f(\mathbf{x})$, no caso deste exemplo, é bastante simples (trata-se de uma soma de valores). Isso significa que, no curso de um processo de otimização, provavelmente o tempo computacional despendido com a execução de instruções do algoritmo de otimização será dominante, comparado com o tempo despendido com as avaliações da função-objetivo. Note-se que esta situação corresponde ao oposto do que ocorria no exemplo anterior. A avaliação da função de restrição agora também tende a ser muito mais dispendiosa que a avaliação da função objetivo.

- Embora exista um número finito de possíveis redes, normalmente também não existe a alternativa de avaliar todas as possibilidades, de maneira exaustiva, como procedimento para determinação do ótimo, pois o número de alternativas é muito grande. Deve-se notar que, para n nós, existem 2^N diferentes redes (nem todas elas factíveis). Assim, por exemplo, num problema com $n = 15$ cidades a serem interligadas, o número de arcos ligando cidades seria de $N = 105$, e haveria 2^{105} redes cujas funções objetivo e de restrição teriam de ser avaliadas[4].

Formulação geral dos problemas de otimização

Grande parte dos problemas de otimização podem ser formulados como:

$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$

$$\text{sujeito a: } \begin{cases} g_1(\mathbf{x}) \leq 0 \\ g_2(\mathbf{x}) \leq 0 \\ \quad \vdots \\ g_m(\mathbf{x}) \leq 0 \end{cases} \tag{1.9}$$

Neste livro, a maioria dos problemas a serem tratados terão essa estrutura. O problema é formulado, então, como a questão de determinar o vetor $\mathbf{x}^*$ que minimiza o valor de uma função-objetivo $f(\mathbf{x})$. Note-se que mesmo no caso de problemas em que a interpretação natural da função-objetivo seria a de algo que se deseja maximizar, sempre é possível fazer uma transformação para uma função que se deseja minimizar (por exemplo multiplicando a função original por -1). Assim, neste livro adotaremos a convenção de sempre procurar a minimização de funções. Também por convenção, as funções de restrição $g_1(\mathbf{x})$ a $g_m(\mathbf{x})$ serão definidas como funções que são satisfeitas quando são menores ou iguais a zero. Dessa forma, uma restrição do tipo $0 \leq x_1 \leq 0.5$, que apareceu no primeiro exemplo, seria desdobrada em $g_1(\mathbf{x}) = -x_1 \leq 0$ e $g_2(\mathbf{x}) = x_1 - 0.5 \leq 0$.

Como já sugerem os exemplos anteriormente mostrados, o vetor de variáveis de decisão pode ser constituído por variáveis contínuas ou por variáveis discretas, ou mesmo por uma combinação de variáveis dos dois tipos. A função-objetivo pode ser desde uma expressão aritmética simples envolvendo as variáveis de decisão, até uma relação de elevada complexidade cuja fórmula explícita não se conheça, e cujo cálculo seria feito a partir de algoritmos numéricos possivelmente custosos computacionalmente. O mesmo vale para as funções de restrição $g_1(\mathbf{x})$ a $g_m(\mathbf{x})$.

Por fim, cabe mencionar que grande parte dos estudos realizados no campo da otimização consideram a função-objetivo $f(\mathbf{x})$ como uma função de várias variáveis (as variáveis de decisão) que retorna um único valor real. No entanto, em situações práticas, é muito frequente que o projeto de um determinado sistema deva considerar mais de um objetivo. Por exemplo, é frequente que o projeto de um dispositivo considere a minimização do custo e a maximização de um índice de desempenho

[4] Supondo uma avaliação a cada microsegundo (ou um milhão de avaliações por segundo), o tempo total a ser gasto com o processo exaustivo seria de aproximadamente $1,28 \times 10^{24}$ anos ($1,28$ trilhões de trilhões de anos) – tempo que um consumidor de serviços telefônicos hoje em dia dificilmente estaria disposto a esperar para poder fazer ligações interurbanas.

como metas a serem alcançadas. A generalização do problema de otimização para tais situações é denominada *otimização multiobjetivo*. Este assunto é tratado no capítulo 17.

Otimização com convergência garantida

Várias técnicas clássicas de otimização foram desenvolvidas para casos especiais do problema (1.9), de forma a garantidamente obter a solução exata. No caso de variáveis contínuas, alguns desses casos especiais importantes são:

- A função objetivo e as restrições são funções afins (ou seja, são construídas como combinações lineares das variáveis de decisão). Técnicas tais como o método SIMPLEX e os métodos de pontos interiores para programação linear são capazes de resolver eficientemente problemas dessa classe, com determinação garantida do ótimo exato. Essa classe de problemas chega a definir um ramo da Otimização, que é chamado de Programação Linear.

- A função objetivo é diferenciável (possui gradiente em todos os pontos) e unimodal (as curvas de nível correspondentes a um dado valor $f(\mathbf{x}) = \alpha_1$ delimitam regiões conexas que contêm inteiramente as curvas de nível $f(\mathbf{x}) = \alpha_2$, sempre que $\alpha_1 > \alpha_2$). Se o problema não possui restrições (recaindo na chamada otimização irrestrita), os pontos de ótimo desses problemas podem ser determidados de maneira eficiente e com garantia de convergência por métodos tais como os quasi-Newton ou os de regiões de confiança. Caso haja restrições lineares, métodos de programação quadrática sequencial podem ser empregados também com garantia de convergência. Esses problemas e seus respectivos métodos de solução fazem parte de outro ramo da Otimização, que se denomina Programação Não-Linear.

- A função objetivo é convexa e todas as restrições são convexas. Nesse caso, métodos de pontos interiores para programação convexa são capazes de determinar de maneira exata o ótimo do problema, de maneira computacionalmente eficiente. Também esses problemas com seus respectivos métodos de solução definem uma área da Otimização: a Programação Convexa.

Muitas vezes é possível aplicar um desses métodos mesmo que a função objetivo e as restrições do problema não tenham as características que fariam com que a aplicação do método levasse garantidamente ao ótimo exato. Por exemplo, se um método quasi-Newton for empregado para otimizar uma função-objetivo que não seja unimodal, seu resultado poderá ser um ótimo apenas local, e não mais necessariamente o ótimo global do problema. Outras vezes, o método não pode ser aplicado a problemas fora da classe para a qual foi projetado; um exemplo disso é o método SIMPLEX, que só se aplica a problemas de Programação Linear.

Quando todas as variáveis de decisão são discretas, define-se uma sub-área da Otimização denominada Otimização Combinatória. No jargão peculiar dessa área, é chamado de *algoritmo* um procedimento computacional para resolver problemas dotado da garantia de que o ótimo exato da função-objetivo será atingido em um número finito de passos[5]. Já uma *aproximação* é um procedimento dotado da garantia de que, após um número finito de passos, será obtida uma solução cujo valor de função-objetivo será "próximo" do valor do ótimo exato, não divergindo deste mais que uma quantidade teoricamente assegurada. Por exemplo, se o valor ótimo da função-objetivo a ser minimizada for igual a J^*, um procedimento de aproximação poderá gerar uma solução cuja função objetivo seja garantidamente menor que $(1 + \epsilon)J^*$. Por fim, uma *heurística* é um procedimento de busca de "boas soluções" para o problema que não é dotado de garantias quanto à qualidade relativa da solução encontrada em comparação com o ótimo exato.

[5] No restante deste livro, entretanto, o termo *algoritmo* será utilizado em um sentido mais fraco, designando um procedimento computacional construído para desempenhar determinada função.

Já deve estar claro que o exame de todas as possíveis combinações de valores das variáveis de decisão, no caso da otimização combinatória, sempre é um *algoritmo*, no sentido de que tal procedimento, se executado, conduz necessariamente à determinação do ótimo exato do problema. No entanto, devido ao rápido crescimento do número de alternativas a serem examinadas à medida em que o número de variáveis cresce, tal tipo de procedimento normalmente não é aplicável exceto em problemas muito pequenos, com poucas variáveis. Em grande parte dos problemas combinatórios, o número de alternativas cresce exponencialmente com o número de variáveis de decisão. No exemplo de projeto de uma rede de comunicações visto anteriormente, o número de alternativas era igual a 2^N, ou seja, um crescimento exponencial com o número N de possíveis arcos que poderiam compor a rede. Dessa forma, a questão relevante em Otimização Combinatória normalmente não diz respeito a se existe algum algoritmo que teoricamente sempre resolva o problema (pois pelo menos o algoritmo "exaustivo" sempre existe), mas se existe algum procedimento computacional que resolva o problema em "tempo razoável" (alguns minutos, ou algumas horas, ou até alguns dias, normalmente é aceitável).

Uma propriedade que frequentemente está associada à noção de "tempo razoável" de um procedimento é sua classe de complexidade. Os procedimentos cuja execução demanda um número de operações proporcional a uma expressão polinomial envolvendo o número de variáveis do problema pertencem à chamada classe $\mathcal{P}$. Formalmente, a classe $\mathcal{P}$ contém todos os problemas de decisão que podem ser resolvidos por uma máquina de Turing determinística usando uma quantidade polinomial de tempo de computação. Note-se que um algoritmo exaustivo, como mencionado acima, não pertence à classe $\mathcal{P}$, pois necessariamente envolve um número de operações que cresce exponencialmente (portanto mais rapidamente que qualquer polinômio) com o número de variáveis do problema.

Há diversos tipos de problemas combinatórios para os quais foram desenvolvidos *algoritmos* de classe de complexidade $\mathcal{P}$. Um desses é o problema de construção da árvore geradora mínima: dado um conjunto de nós, deseja-se determinar o conjunto de arcos que produz um grafo conexo (ou seja, um grafo no qual exista algum caminho que possa ser percorrido saindo de qualquer nó e chegando a qualquer outro nó), com a mínima soma de custos dos arcos. Para esse problema, são hoje corriqueiramente empregados algoritmos[6] cuja ordem de complexidade é $\mathcal{O}(N\log(N))$, ou seja, o esforço computacional para a determinação da solução ótima é proporcional a $N\log(N)$, sendo N o número de variáveis (arcos que podem ser instalados) do problema[7].

O problema de construção da rede de comunicações descrito no exemplo anteriormente mostrado, em particular, pode ser resolvido por meio de uma adaptação simples de algoritmos para árvore geradora mínima. Usando tais algoritmos, a solução ótima para o problema de interligar 15 cidades seria obtida em alguns milisegundos, não sendo portanto necessária a enumeração exaustiva de alternativas que levaria trilhões de anos.

2. Heurística

Nesta seção, é apresentada a motivação para o desenvolvimento de *heurísticas*. Conforme foi visto na seção anterior, heurísticas são procedimentos que tratam problemas de otimização sem dispor de garantias teóricas nem de que a solução ótima exata seja obtida, e nem tampouco de garantias de que a solução obtida tenha algum tipo de proximidade em relação à solução ótima exata. A lógica de uma heurística é, ao contrário, a de garantir que a solução obtida seja melhor que a vasta maioria das alternativas disponíveis, sendo nesse sentido uma "boa solução". Com tal sentido, pode-se falar de heurísticas tanto no contexto da otimização com variáveis discretas quanto no contexto da otimização com variáveis contínuas. A questão pertinente para motivar o desenvolvimento de heurísticas é: por

[6] Os algoritmos mais comumente empregados para o problema da árvore geradora mínima são o algoritmo de Kruskal e o algoritmo de Prim.

[7] A expressão $N\log(N)$ não é um polinômio, mas cresce mais devagar que o polinômio N^2, o que faz com que algoritmos dessa ordem de complexidade sejam $\mathcal{P}$.

quê não empregar métodos exatos de otimização (ou, no caso de variáveis discretas, algoritmos)?

Conforme foi visto na seção anterior, diversos ramos da Otimização vieram se dedicando a tratar diferentes classes de problemas, ao longo do século XX, de forma que hoje encontra-se disponível um repertório relativamente vasto de métodos de otimização capazes de determinar, forma eficiente e garantida, a solução ótima exata de problemas dessas classes. No entanto, a variedade de problemas de interesse prático, para os quais seria desejável que se conhecessem métodos exatos e eficientes de resolução é muito mais vasta do que a dos problemas para os quais tais métodos encontram-se já desenvolvidos.

Para explicar esse ponto, considerem-se os seguintes problemas:

- Dado um conjunto de nós, determinar um conjunto de arcos que ligam todos os nós, formando um grafo conexo com a mínima soma dos custos dos arcos.

- Dado um conjunto de nós, determinar um conjunto de arcos que ligam todos os nós, formando um grafo conexo com a mínima soma dos custos dos arcos, com a restrição adicional de que nenhum nó possa ter um número de arcos incidindo nele (grau) maior que uma constante R.

O primeiro desses problemas é o chamado problema da *árvore geradora mínima*, que constitui um desses problemas "básicos", que já mereceram estudos extensos, e para os quais já se conhecem boas soluções (algoritmos de classe $\mathcal{P}$). O segundo problema, por outro lado, é enunciado como uma pequena variação em torno do primeiro problema, e com tal enunciado ele é conhecido como o problema da *árvore geradora mínima com restrição de grau*. No que diz respeito à complexidade computacional do problema, a modificação causa um impacto profundo: o problema passa a integrar a classe dos problemas $\mathcal{NP}$-difíceis, para os quais não se conhecem hoje algoritmos capazes de determinar seu ótimo garantidamente em tempo polinomial.

Não é difícil intuir que problemas práticos, correspondentes a situações do mundo real, normalmente devem atender a certos requisitos que são impostos pela situação prática. A restrição de grau em uma rede de computadores, por exemplo, pode ser originária da característica dos equipamentos da rede, que possivelmente teriam um número de saídas máximo igual a R. O raciocínio que desenvolvemos aqui é o seguinte: (i) Mesmo que problemas práticos tenham uma estrutura próxima daquela dos problemas especiais, para os quais são conhecidos procedimentos exatos e eficientes, normalmente será necessária alguma adaptação para que cada situação particular seja representada adequadamente. (ii) Mesmo que a adaptação seja pequena, é grande a chance de que os métodos de solução conhecidos para o problema especial não sejam mais aplicáveis ao problema modificado. Tende a haver, portanto, uma profusão de problemas que não se sabe resolver de maneira exata.

Na verdade, a questão de por quê desenvolver heurísticas vai bem além da colocação de fundo pragmático que afirmasse que elas seriam necessárias até que se desenvolvam algoritmos exatos de tempo polinomial para substituí-las. Uma das questões fundamentais em aberto na teoria da computação, hoje em dia, diz respeito exatamente à possibilidade ou impossibilidade de se construírem tais algoritmos, para uma vasta classe de problemas conhecidos como $\mathcal{NP}$ (a sigla significa "não-deterministicamente polinomial"). Formalmente, esses são os problemas em que se procura responder "sim" ou "não" a uma pergunta que diga respeito, por exemplo, à existência de um tipo de elemento em um conjunto, sendo que para a verificação de cada elemento do conjunto encontra-se disponível um algoritmo de tempo polinomial.

Exemplifiquemos isto no contexto de um problema de otimização com variáveis discretas. Suponha-se que se deseja saber se existe algum vetor de variáveis de decisão $\mathbf{x}$ cujo valor de função objetivo seja menor que um certo valor ϵ. Normalmente, fixando-se um certo vetor de variáveis $\mathbf{x}_a$, é possível em tempo polinomial responder à pergunta quanto a se a expressão $f(\mathbf{x}_a) < \epsilon$ é verdadeira ou não. Se um procedimento for executado de forma a que sejam testadas diversas alternativas aleatoriamente combinadas de valores das variáveis de decisão, existe a possibilidade de que seja encontrado em tempo

polinomial algum vetor que atenda a $f(\mathbf{x}) < \epsilon$. Um problema desse tipo é definido como pertencente à classe $\mathcal{NP}$.

Os quadros a seguir realçam as definições de problema $\mathcal{P}$ e de problema $\mathcal{NP}$.

Problema $\mathcal{P}$

- Dados de entrada: $f(\mathbf{x})$ e ϵ.
- Pergunta: Existe $\mathbf{x}$ tal que $f(\mathbf{x}) < \epsilon$?
- O problema será da classe $\mathcal{P}$ se existir um algoritmo determinístico que responda, em tempo polinomial, à pergunta.

Problema $\mathcal{NP}$

- Dados de entrada: $f(\mathbf{x})$ e ϵ.
- Pergunta 1: se for dado um certo $\mathbf{x}$, esse vetor satisfaz $f(\mathbf{x}) < \epsilon$?
- Pergunta 2: existe algum vetor $\mathbf{x}$ que satisfaz $f(\mathbf{x}) < \epsilon$?
- O problema de responder à Pergunta 2 será da classe $\mathcal{NP}$ se existir um algoritmo determinístico que responda, em tempo polinomial, à Pergunta 1.

A motivação que leva os problemas $\mathcal{NP}$ a serem designados por este nome pode ser explicada a partir do seguinte procedimento para tratar a Pergunta 2, cuja construção é feita a partir de uma sequência de respostas à Pergunta 1:

Algoritmo 1 Pergunta 1 - procedimento não-determinístico

1: **Entradas:** A função $f(\cdot)$ e o escalar ϵ.
2: **Saída:** A resposta R à pergunta: existe x tal que $f(x) < \epsilon$?
3: $R \leftarrow$ `FALSO`
4: **enquanto** $R =$`FALSO` **faça**
5: Gerar aleatoriamente uma instância do vetor x
6: **se** $f(x) < \epsilon$ **então**
7: $R \leftarrow$ `VERDADEIRO`
8: **fim se**
9: **fim enquanto**

Para entender a denominação $\mathcal{NP}$, suponha-se que existam m instâncias do vetor $\mathbf{x}$ que satisfazem $f(x) < \epsilon$, de um total de M possíveis $\mathbf{x}$ diferentes. O passo 5 do procedimento que gera uma instância aleatória de $\mathbf{x}$ tem uma probabilidade $\frac{m}{M}$ de gerar um $\mathbf{x}$ tal que $f(x) < \epsilon$. Note-se que para cada valor de ϵ diferente, ocorrerá um número m de instâncias do vetor $\mathbf{x}$ que irão satisfazer $f(x) < \epsilon$. Assim, dado um número ϵ, em média serão necessárias $\frac{M}{m}$ tentativas aleatórias até que se gere um $\mathbf{x}$ tal que $f(x) < \epsilon$. O número de tentativas, nesse sentido, não depende do número n de variáveis do problema. Assim, o problema de determinação de algum $\mathbf{x}$ que satisfaça $f(x) < \epsilon$ será resolvido, em média, em tempo polinomial, caso exista algum $\mathbf{x}$ que satisfaça a essa relação. Caso não exista tal $\mathbf{x}$, o procedimento nunca irá terminar.

A pergunta considerada fundamental é: será que existe, para todo problema $\mathcal{NP}$, um algoritmo determinístico de tempo polinomial que o resolva? Para alguns problemas dessa classe, que constituem o subconjunto conhecido como $\mathcal{P}$, sabemos que isso é possível, e são conhecidos tais algoritmos. Um exemplo desse tipo de problema pode ser definido a partir da função que mede o comprimento total de

uma árvore geradora que conecta um conjunto de nós. Seja $\mathbf{x}$ o vetor que codifica os arcos que podem ligar esse conjunto de nós, como na expressão (1.5), e seja $f(\mathbf{x})$ a função que representa o comprimento total da árvore formada por esses arcos, como na expressão (1.6). Considere-se a pergunta: existe algum $\mathbf{x}$ tal que $f(\mathbf{x}) < \epsilon$? Dado um vetor $\mathbf{x}$, é possível saber, em tempo polinomial, se $f(\mathbf{x}) < \epsilon$; basta calcular o valor da expressão (1.6) para esse $\mathbf{x}$. Portanto, essa pergunta é um problema $\mathcal{NP}$. No entanto, como já foi discutido, existem procedimentos – os algoritmos de árvore geradora mínima – que são capazes de determinar, em tempo polinomial, de maneira determinística, o vetor $\mathbf{x}^*$ que minimiza $f(\mathbf{x})$. Portanto, tais algoritmos podem ser empregados para responder à pergunta quanto a se existe algum $\mathbf{x}$ que irá satisfazer $f(x) < \epsilon$. Basta determinar $f(\mathbf{x}^*)$: se $f(\mathbf{x}^*) < \epsilon$, então a resposta é **sim**, pois existe pelo menos um vetor que satisfaz a expressão, e esse vetor é $\mathbf{x}^*$. Do contrário, a resposta será **não**, pois não pode existir nenhum vetor que produza uma árvore com comprimento ainda menor que o mínimo. Assim, alguns problemas $\mathcal{NP}$ são também $\mathcal{P}$.

No entanto, há uma sub-classe dos problemas $\mathcal{NP}$, que são denominados problemas $\mathcal{NP}$-completos, para os quais não apenas não se conhecem algoritmos de tempo polinomial: há hoje a suspeita de que não possam existir algoritmos de tempo polinomial para tais problemas. Se isso for verdade, haverá uma impossibilidade teórica impedindo o desenvolvimento de algoritmos de tempo polinomial que garantidamente conduzam à solução ótima do problema.

Há ainda os chamados problemas $\mathcal{NP}$-difíceis, que são definidos como problemas pelo menos tão difíceis quanto problemas $\mathcal{NP}$-completos relacionados com eles. Problemas de otimização frequentemente são $\mathcal{NP}$-difíceis. Suponha-se que o problema de determinar se existe $\mathbf{x}$ tal que $f(\mathbf{x}) < \epsilon$ seja pertencente à classe $\mathcal{NP}$-completa. Nesse caso, o problema de determinar o vetor $\mathbf{x}^*$ que minimiza a função $f(\mathbf{x})$ será $\mathcal{NP}$-difícil, pois determinar esse vetor ótimo pode envolver um procedimento de executar várias vezes um algoritmo que determina um vetor $\mathbf{x}$ que satisfaça $f(\mathbf{x}) < \epsilon$, para valores cada vez menores de ϵ. Deve-se notar que se colocam dois problemas não triviais: (i) se não existir $\mathbf{x}$ que satisfaça $f(\mathbf{x}) < \epsilon$, o procedimento que procura tal vetor pode nunca parar; e (ii) mesmo que exista $\mathbf{x}$ tal que $f(\mathbf{x}) < \epsilon$, fica cada vez mais difícil encontrar esse vetor, à medida em que ϵ diminui, pois diminui a proporção $\frac{m}{M}$ dos vetores que satisfazem a desigualdade, em relação ao conjunto de todos os possíveis vetores. Dessa forma, é possível que o problema de minimizar $f(\mathbf{x})$ seja de tempo exponencial (e portanto não seja $\mathcal{NP}$) mesmo que o problema de determinar um $\mathbf{x}$ que satisfaça $f(\mathbf{x}) < \epsilon$ seja $\mathcal{NP}$.

Retornamos então à justificativa para o desenvolvimento de heurísticas: elas são necessárias não apenas porque não são conhecidos hoje algoritmos eficientes para resolver diversos problemas de importância prática, mas principalmente porque é possível que tais algoritmos simplesmente não existam. Dada a diferença de escala de tempo entre a duração da execução de um algoritmo exato com complexidade exponencial (que, no caso do exemplo discutido anteriormente seria da ordem de trilhões de trilhões de anos) e a de uma heurística capaz de encontrar uma "boa solução" para o mesmo problema em alguns minutos, fica clara a necessidade de se desenvolverem heurísticas.

Embora a discussão nesta seção, até este ponto, tenha se concentrado em problemas de variáveis discretas, as mesmas idéias gerais se aplicam a problemas com variáveis contínuas. Os métodos de otimização para tais problemas que têm garantia de convergência para a solução exata sempre se baseiam em premissas fortes sobre a estrutura da função a ser otimizada. Quando a função atende a premissas tais como a de convexidade, ou de unimodalidade e diferenciabilidade, métodos especialmente desenvolvidos para a classe de funções em questão permitem determinar o ótimo do problema de otimização com eficiência computacional e garantia de convergência. A Figura 1.3 mostra o gráfico de uma função unimodal diferenciável, cujo mínimo pode ser determinado facilmente por métodos tais como os quasi-Newton.

Problemas reais, no entanto, não se restringem a ocorrer apenas nessas formas especiais: pelo contrário, funções com estrutura bastante arbitrária podem ocorrer em problemas práticos. A Figura 1.4 mostra o gráfico de uma função multimodal que possui um grande número de bacias de atração,

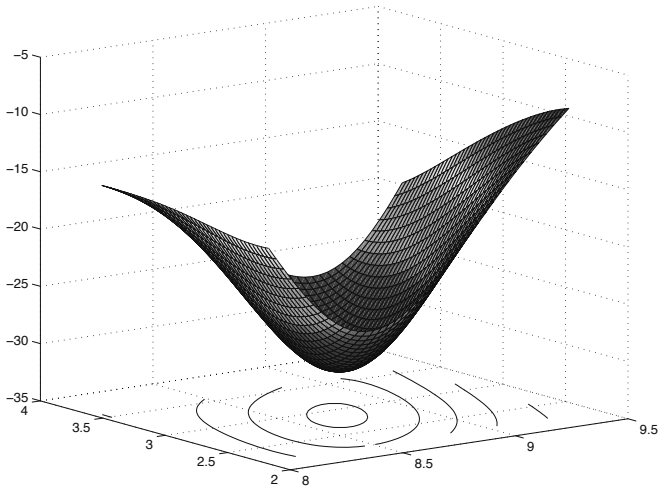


Figura 1.3: Gráfico de uma função diferenciável e unimodal.

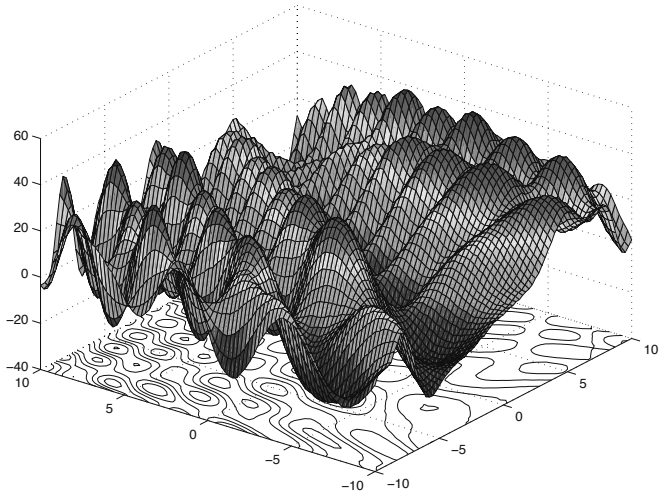


Figura 1.4: Gráfico de uma função multimodal.

cada uma com um mínimo local. Para tais funções, os métodos clássicos de otimização não iriam apenas demorar a convergir para o ótimo: em geral, a convergência simplesmente não ocorreria. Se se aplicar um método "de descida" como tentativa de determinação do ponto de mínimo da função representada na Figura 1.4, o método irá convergir para o mínimo local em cuja bacia de atração o ponto de partida do método estiver localizado. Esse tipo de método simplesmente não possui qualquer mecanismo capaz de guiá-lo, em meio a múltiplos mínimos locais, até o mínimo global. Na verdade, esse tipo de método não tem sequer um mecanismo capaz de conduzí-lo a outro mínimo local melhor, caso ele convirja para um mínimo local qualquer.

A necessidade de otimizar funções desse tipo tem sido invocada como a principal motivação para o desenvolvimento de métodos de otimização baseados em heurística, no caso de problemas de otimização com variáveis contínuas. A heurística teria o papel de guiar uma busca no domínio desejado, de

forma a examinar a "paisagem" geral da função, assim realizando uma busca em várias regiões-candidatas promissoras, com a progressiva concentração da busca nas melhores regiões encontradas, na tentativa de se determinar o ótimo global do problema. Heurísticas, por definição, não serão dotadas de garantias de convergência para o ótimo global exato. No entanto, no caso de problemas com variáveis contínuas, elas constituem uma forma efetiva (frequentemente a única forma) de se determinarem "boas soluções", no sentido de serem soluções significativamente melhores que aquelas que seriam normalmente encontradas por métodos de busca apenas local[8].

3. Computação Evolutiva

Nos últimos 40 anos do século XX, foi progressivamente se firmando a idéia de se construírem heurísticas inspiradas em mecanismos de adaptação dos seres vivos, conforme observados na natureza. A justificativa geral para esse programa de trabalho é que tais mecanismos de adaptação, na natureza, devem produzir respostas adequadas para problemas de grande complexidade. Esse efeito ainda deve ocorrer como consequência da execução de ações simples por múltiplos agentes individuais que interagem entre si e com o ambiente, produzindo coletivamente a solução do problema de adaptação. Transposta para o contexto da computação, tal idéia torna-se bastante atraente, pois implica que agentes que executam tarefas computacionalmente muito simples podem resolver problemas de grande complexidade quando agrupados em coletividades, com a possibilidade adicional de se explorar o paralelismo computacional que a estrutura de múltiplos agentes sugere.

A adaptação das espécies ao ambiente natural, por meio do processo evolutivo, ou a busca de alimentos efetuada de maneira coordenada por colônias de formigas ou bandos de pássaros, ou ainda a resposta do sistema imunológico de mamíferos à invasão de agentes patógenos, envolvem essencialmente a realização de buscas em espaços de alternativas de elevada dimensão, sobre funções que não apenas são de elevada complexidade, mas que também mudam com o passar do tempo, exigindo uma contínua execução desses mecanismos de adaptação. As soluções para tais problemas de adaptação exibidas pelos atuais seres biológicos existentes no planeta na verdade tratam-se de mecanismos bastante bem sucedidos, pois permitiram a essas espécies evoluir e estar hoje presentes, após bilhões de anos de evolução da vida e de luta pela sobrevivência. Por esses motivos, acredita-se que a estrutura de tais mecanismos naturais de adaptação[9] possa ser transposta com sucesso para a construção de mecanismos computacionais de adaptação destinados a tratar de problemas de otimização de alta complexidade.

A primeira grande metáfora a ser sugerida, nesse contexto, foi a da evolução natural dos seres vivos. O mecanismo heurístico de adaptação iria tratar simultaneamente um conjunto de diferentes vetores de variáveis de decisão (esse conjunto seria o análogo computacional de uma "população" de seres vivos). Cada um desses vetores de variáveis de decisão, levando adiante a analogia, seria um "indivíduo". Os indivíduos de uma população deveriam sofrer variações aleatórias que, por um lado, representassem novos pontos a serem analisados, mas que por outro lado tivessem algum tipo de semelhança (ou proximidade) com os pontos anteriores, seguindo a lógica de que mais pontos fossem gerados perto dos melhores pontos anteriores, e menos pontos fossem gerados perto dos piores, assim se criando uma "memória" dentro do mecanismo de otimização. Por fim, fazendo analogia com o mecanismo de seleção natural darwiniana, os indivíduos pertencentes a uma população são selecionados para passar para a geração seguinte de acordo com um processo que pode ser aleatório mas que pelo menos garante maiores probabilidades de serem selecionados aos indivíduos com melhores valores de função-objetivo – assim gerando uma pressão que conduz o processo à convergência para valores sempre melhores de função.

[8] Deve-se notar que os métodos clássicos de otimização, quando aplicados a problemas com funções arbitrárias, podem simplesmente ficar inaplicáveis, ou ainda podem passar a ter comportamento imprevisível (convergindo para pontos que não são sequer mínimos locais), ou por fim podem executar uma busca local, convergindo para o mínimo local da bacia de atração na qual estiver localizado o ponto inicial da busca.

[9] Um caso particular de problema de adaptação é o problema de otimização, do qual trata este livro.

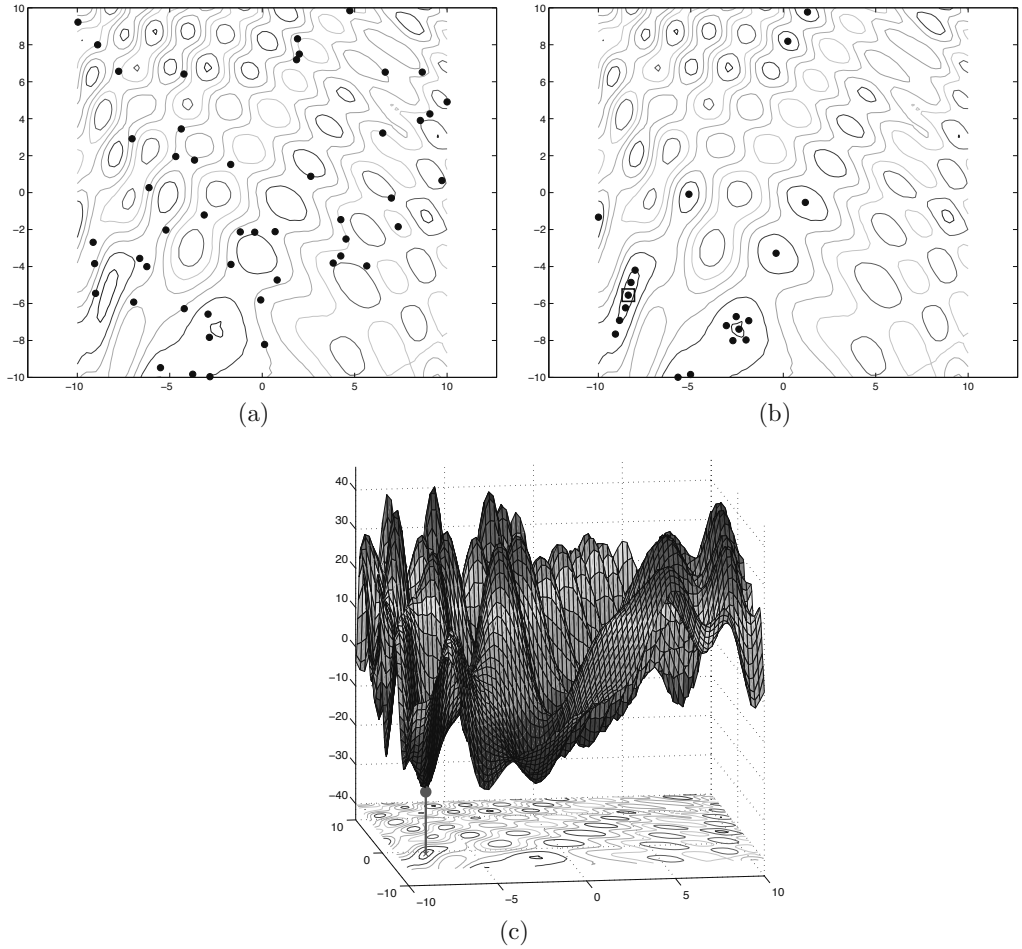


Figura 1.5: Resultados da execução de um Algoritmo Genético para a determinação do mínimo da função representada na Figura 1.4: (a) População inicial gerada aleatoriamente. (b) Vários pontos candidatos a mínimos locais determinados pelo AG, sendo o ponto de mínimo global destacado por um quadrado. (c) O ponto de ótimo, tanto no mapa das curvas de nível quanto no gráfico da função.

Essa metáfora inspirou o desenvolvimento das *Estratégias Evolutivas*, dos *Algoritmos Genéticos*, e da *Programação Genética*, cada qual interpretando de forma diferente os diversos elementos desse mecanismo geral de adaptação. Esses algoritmos são apresentados respectivamente nos capítulos 3, 2 e 4 deste livro.

A execução de um algoritmo genético sobre a função multimodal representada na Figura 1.4, leva aos resultados ilustrados na Figura 1.5. Na Figura 1.5(a), observa-se a distribuição dos pontos que compõem a população inicial, os quais foram gerados aleatoriamente com distribuição uniforme sobre o domínio de interesse. Esses pontos encontram-se sobrepostos ao mapa de curvas de nível da função. Na Figura 1.5(b), são mostrados pontos pertencentes à população final, após 150 gerações, os quais se localizam nas proximidades de diferentes mínimos locais da função. O ponto de ótimo que foi encontrado (aquele cujo valor de função foi o menor dentre todos) se encontra representado em destaque. A Figura 1.5(c), por fim, mostra o ponto de ótimo encontrado, que de fato coincide com

o ponto no qual ocorre o mínimo valor da função dentro do domínio, referenciado tanto ao mapa de curvas de nível quanto ao gráfico da função. Deve-se notar no gráfico que o ponto de ótimo global é localizado, de fato, um pouco abaixo dos demais mínimos locais.

Com o avanço das pesquisas, verificou-se que heurísticas dotadas de mecanismos distintos de funcionamento podem ser mais adequadas para problemas com determinada estrutura, e outras heurísticas podem funcionar melhor em outras classes de problemas. Isso motivou a que outros processos da natureza diferentes do da evolução das espécies servissem de inspiração para o desenvolvimento de novas heurísticas, dotadas de propriedades que as tornam peculiarmente adequadas para tratar problemas com características específicas. Alguns exemplos disso são:

- Um tipo de heurística, o *sistema de colônia de formigas*, cuja motivação biológica é a forma como as colônias de formigas encontram caminhos ótimos que as levem até suas fontes de alimento, veio sendo desenvolvida nos últimos 15 anos principalmente para tratar de problemas de determinação de caminhos ótimos. Essa classe de métodos é tratada no Capítulo 5 deste livro.

- Outro tipo de heurística, o *sistema imunológico artificial*, motivada em mecanismos dos sistemas imunológicos dos mamíferos, acaba herdando de seu mecanismo natural inspirador a característica de privilegiar a preservação da diversidade de soluções – isso é necessário, no caso dos seres vivos, para que estes estejam sempre preparados para combater diferentes organismos invasores, e no caso de problemas de otimização, essa característica pode ser também valiosa em diversos contextos práticos. Esses algoritmos são abordados no Capítulo 6 deste livro.

- Algumas funções-objetivo são particularmente irregulares, com uma dificuldade extra de que essa irregularidade se manifesta em diferentes escalas. Isso pode ser imaginado como se as bacias de atração da função contivessem outras bacias menores, e essas ainda outras, e assim por diante. Para tratar desse tipo de função, uma heurística que se inspirou no fenômeno físico do recozimento de metais[10], o *recozimento simulado* veio se mostrando bastante adequada. O Capítulo 8 deste livro apresenta esses algoritmos.

O crescente entendimento dos mecanismos que fundamentam os algoritmos da computação evolutiva tem permitido, de certa forma, um afastamento dos novos algoritmos que vêm sendo desenvolvidos em relação à estrita inspiração biológica. De meados da década dos 1990 para cá, diversos algoritmos foram desenvolvidos sem que houvesse a preocupação de que de alguma forma fosse imitado algum mecanismo de adaptação observado na natureza. Hoje podem ser citados algoritmos tais como o de *evolução diferencial*, ou o de *estimação da distribuição* que, pelos critérios: (1) de terem sido propostos de certa forma em continuidade com a tradição dos algoritmos bio-inspirados; e (2) de serem baseados em busca com aleatoriedade e envolvendo uma população de pontos – pode-se dizer que pertençam ao campo da Computação Evolutiva. Esses algoritmos são descritos respectivamente no Capítulo 7 e no Capítulo 11 deste livro. Por outro lado, ao contrário dos primeiros algoritmos desse campo, estes dois se fundamentam em operações que não reivindicam qualquer semelhança com nenhum processo natural, e que fazem parte do algoritmo simplesmente porque parece computacionalmente razoável que tais operações contribuam para a determinação do ótimo. Os novos algoritmos que se encontram em desenvolvimento, na linha de frente da pesquisa em Computação Evolutiva, tendem a aprofundar essa tendência de incorporação de operações e mecanismos que não sejam bio-inspirados, mas sim inspirados em argumentos matemáticos ou computacionais.

Metaheurísticas e Computação Evolutiva

O fio temático deste livro são as heurísticas originalmente derivadas do campo da Computação Evolutiva. Acabamos de discutir que, embora a inspiração biológica ainda seja uma fonte de idéias para

[10] Na metalurgia, o recozimento serve para que a estrutura cristalina de metais atinja níveis de menor energia, melhorando as propriedades mecânicas do material.

as pesquisas nesse campo, ele já não pode ser definido a partir dessa inspiração, pois já se tornou frequente o uso de mecanismos cuja justificativa é essencialmente matemática e computacional. Além dessa inflexão conceitual dentro da própria Computação Evolutiva, há um outro processo que também tem causado um realinhamento das fronteiras da área: tem havido, nos últimos anos, uma convergência entre a Computação Evolutiva e linhas de pesquisa provenientes do campo da Pesquisa Operacional, que há muito têm se dedicado ao estudo de técnicas heurísticas estocásticas de propósito geral para problemas de otimização combinatória – as *metaheurísticas*.

Diferente das heurísticas mais clássicas, que eram desenvolvidas para classes de problemas bastante particulares, as metaheurísticas são heurísticas que foram desenvolvidas com o objetivo de constituírem técnicas baseadas em operações suficientemente gerais para serem aplicáveis à maior parte dos problemas combinatórios. Dessa forma, as metaheurísticas em geral se fundamentaram em conceitos relativamente genéricos, tais como o de vizinhanças de pontos no espaço de soluções discretas, ou de trajetórias conectando dois pontos, ou ainda na organização da informação adquirida a respeito de regiões desse espaço de soluções, permitindo a construção de algoritmos eficientes com características tanto de busca local quanto de busca global.

Dentre as técnicas originariamente classificadas como metaheurísticas que têm concorrido para essa fusão com os algoritmos da Computação Evolutiva, podem ser mencionadas como exemplos a *busca tabu*, as técnicas de *busca gulosa randomizada adaptativa*, a *busca local iterada*, a *busca por vizinhança variável*, e outras. Essas técnicas constituem os temas dos capítulos 9, 10 e 12 deste livro.

4. Premissa: Localidade Fraca

Conforme já foi discutido, as técnicas clássicas de otimização dependem de que sejam satisfeitas premissas relativamente fortes sobre as funções a serem otimizadas, para funcionarem corretamente. Assim, no caso de variáveis contínuas, há métodos que dependem de premissas tais como a convexidade, ou a diferenciabilidade, ou a unimodalidade da função-objetivo. No caso de variáveis discretas, os problemas para os quais são conhecidos algoritmos exatos de tempo polinomial usualmente são aqueles para os quais a realização de uma busca local gulosa[11] é suficiente para garantir que se atinja o ótimo – esse tipo de propriedade da função objetivo se parece muito com a propriedade de unimodalidade das funções de variáveis contínuas, que também garante que uma busca local conduza ao mínimo global da função-objetivo. Problemas de otimização combinatória que não são susceptíveis de serem resolvidos de maneira exata por nenhuma estratégia gulosa costumam ser classificados na classe $\mathcal{NP}$-difícil, não sendo hoje conhecidos para esses problemas algoritmos exatos de tempo polinomial.

Pois bem, para atingirem o ótimo, os métodos da Computação Evolutiva não dependem de premissas dessa ordem. A questão a ser aqui discutida é: esses métodos seriam então capazes de produzir boas soluções aproximadas para problemas de qualquer natureza, independente da suposição de qualquer estrutura para a função-objetivo? É previsível que a resposta a essa pergunta seja negativa. Mas qual seria então a premissa da qual depende o funcionamento dos algoritmos evolutivos? Que tipo de estrutura deve possuir uma função objetivo para que seja possível aplicar um algoritmo desse tipo?

Essa premissa tem sido identificada como sendo a *localidade fraca* da função. Para explicar esse conceito, recorremos à Figura 1.6. Essa figura procura ilustrar a ideia de que mesmo uma função com grande grau de aleatoriedade (portanto insuficientemente dotada de estrutura, pelo menos no sentido que normalmente seria exigido pelos métodos clássicos de otimização) ainda pode ser otimizada, pelo

[11] Uma busca "gulosa" é aquela feita procurando modificar soluções previamente conhecidas de forma a melhorá-las imediatamente (com uma única mudança em uma variável, ou com um pequeno número predefinido de mudanças em veriáveis). No caso mais geral, esse tipo de busca costuma levar a ótimos locais dos problemas combinatórios, ou seja, ótimos que não podem ser melhorados considerando apenas a pequena modificação permitida pelo método, mas que ainda não atingem o ótimo global, que só seria atingido por meio de uma modificação envolvendo um maior número de variáveis.

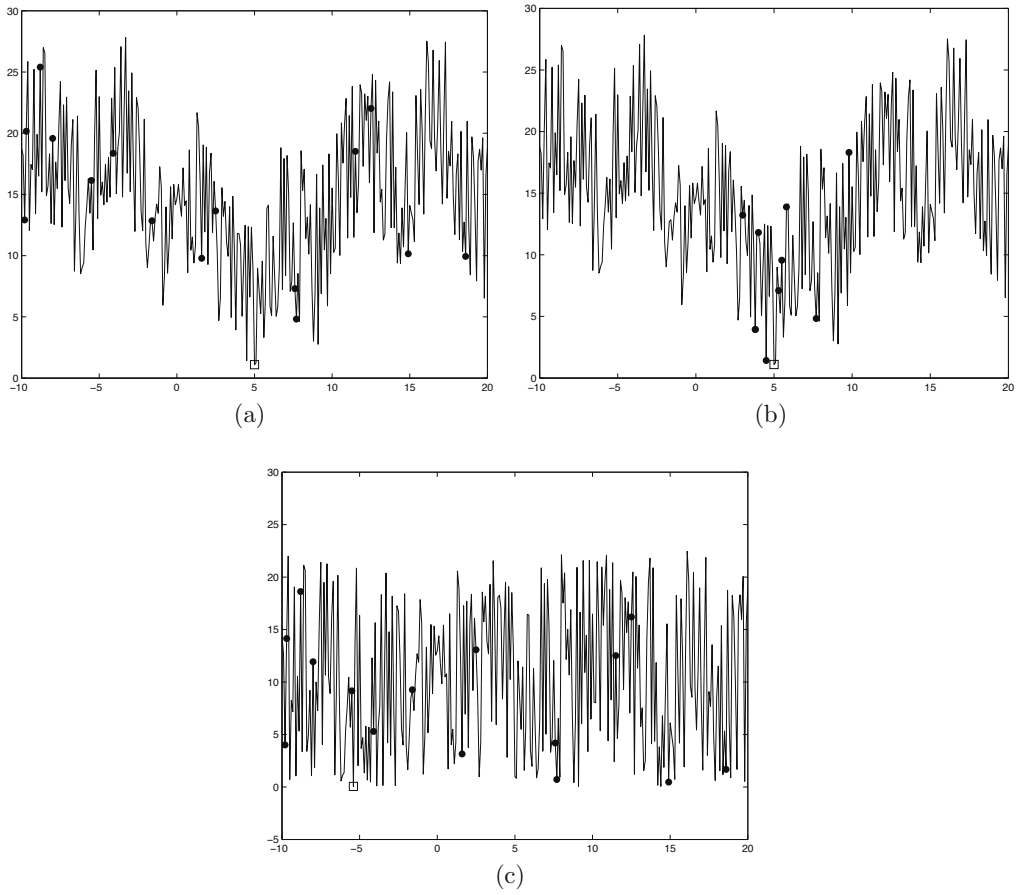


Figura 1.6: As subfiguras mostram funções localmente aleatórias. As subfiguras (a) e (b) representam uma função na qual ocorre a localidade fraca, ou seja, que apresentam algum tipo de tendência global. Na subfigura (a) uma população de amostras gerada aleatoriamente é representada por pontos sobre a curva, e o ponto de mínimo exato é representado por um quadrado. A subfigura (b) mostra uma nova população, gerada a partir de perturbações dos melhores pontos da população da subfigura (a). A subfigura (c) mostra uma função de característica globalmente aleatória, sem nenhum tipo de tendência global. Uma população inicial gerada aleatoriamente, representada por pontos sobre o gráfico, não contém informação capaz de direcionar a busca para as imediações do ponto de mínimo, representado por um quadrado.

menos em um sentido aproximado, por meio de métodos característicos da Computação Evolutiva. O que se exige, para que isso seja possível, é que a função possua pelo menos a característica de *localidade fraca*, que pode ser entendida como possuir uma espécie de tendência global que se superponha à aleatoriedade local, de forma a permitir que informações coletadas em diferentes regiões do domínio da função possam ser comparadas de forma a discriminar as melhores regiões. Isso é ilutrado pelas subfiguras 1.6(a) e 1.6(b), que mostram uma função com aleatoriedade local, mas que exibe uma tendência global. Uma população inicial, mostrada na subfigura 1.6(a), é gerada de forma a amostrar aleatoriamente alguns valores da função, e a comparação entre os valores de função obtidos permite que alguns pontos sejam identificados como sendo os melhores. Uma nova amostragem realizada nas proximidades desses pontos, mostrada na subfigura 1.6(b), já permite a obtenção de boas soluções aproximadas para o ótimo da função.

Contrastando com a situação em que existe a localidade fraca, a subfigura 1.6(c) exibe uma função que não dispõe dessa propriedade. Seu gráfico, em toda a extensão do domínimo de interesse, se assemelha a uma linha passando por uma sequência de pontos gerados aleatoriamente. Em uma situação assim, o conhecimento do valor da função em alguns pontos do domínio não traz informação a respeito de como a função deve se comportar em regiões próximas desses pontos[12]. Assim, uma função desprovida de localidade fraca tende a não ser otimizável, no sentido de que nenhum procedimento sistemático será em média melhor que uma busca aleatória (feita com o simples sorteio sucessivo de pontos para serem examinados), e esta também não será em média melhor que nenhum procedimento organizado da forma que for. Independente do que se faça, deparar com o ótimo dependerá exclusivamente do acaso.

A propriedade de localidade fraca pode então ser definida, de maneira mais precisa, como a presença de *correlação* entre os valores assumidos pela função-objetivo, sendo tal correlação tanto maior quanto menor for a *distância* entre os pontos do espaço de variáveis de decisão nos quais se avalia a função-objetivo. A ideia é que, quanto mais próximos estiverem dois pontos, mais parecidos deverão ser seus valores de função-objetivo, pelo menos em um sentido probabilístico. Funções que tenham tal propriedade serão otimizáveis por meio de técnicas da Computação Evolutiva.

Essa formulação evidencia um componente essencial para que faça sentido o conceito de localidade fraca: é necessário que esteja bem definida uma noção de *distância* no espaço de variáveis de decisão. Somente o uso de tal noção é que permite que um algoritmo faça buscas em regiões *perto* de um ponto, ou ainda que o algoritmo evite regiões *próximas* de outros pontos. Sem alguma medida de distância, seria impossível construir algoritmos evolutivos. No caso de funções de variáveis contínuas, essa noção decorre diretamente das propriedades do espaço vetorial normado $\mathbb{R}^n$, que já incluem automaticamente uma definição de distância (usualmente, mas não necessariamente, a distância entre dois pontos é a norma euclidiana do vetor diferença entre esses pontos).

No caso de problemas combinatórios, a situação é bastante diferente. O conceito que seria mais óbvio para definir a distância entre soluções, nesse caso, seria a contagem do número de variáveis cujos valores não coincidem, de uma solução para a outra. Tal conceito, entretanto, frequentemente se revela inadequado para medir distâncias de forma útil. A Figura 1.7 ilustra essa inadequação. Para conduzir a discussão aqui, imagine-se que o grafo da Figura 1.7(a) represente uma rede de distribuição de água, que sai de um reservatório representado por um quadrado, e que percorre a rede até chegar nos consumidores, representados pelos nós circulares. As figuras 1.7(b) e 1.7(c) representam modificações na primeira rede. Deve-se notar que a rede da Figura 1.7(b) é formada pela mínima perturbação possível sobre a rede da Figura 1.7(a): apenas um arco é retirado, sendo incluído outro arco para

[12] O leitor mais atento talvez fique incomodado com um detalhe que está sendo apresentado sem muito rigor: uma função contínua, como a da subfigura 1.6(c), necessariamente tem de possuir alguma localidade, pois pelo menos a uma distância muito pequena, os valores que a função assume devem ser parecidos, pela própria definição de continuidade. Se o leitor admitir que as amostras devem estar espaçadas de uma distância mínima δ entre si (essa distância pode representar, por exemplo, a resolução da representação numérica em um sistema computacional), a argumentação aqui desenvolvida torna-se precisa.

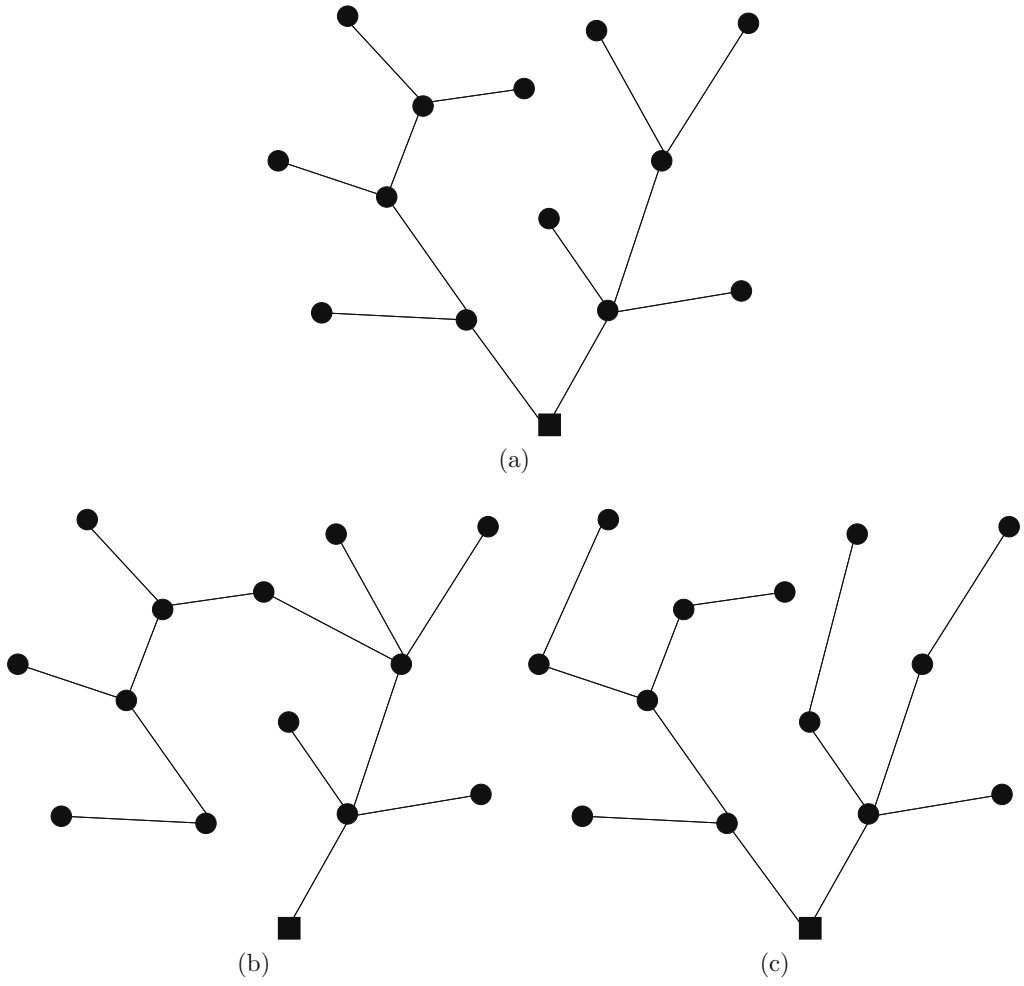


Figura 1.7: (a) Uma rede de distribuição de água, ligando o reservatório, representado por um quadrado, aos nós consumidores, representados por círculos. (b) Outra rede de distribuição de água, obtida retirando um arco da primeira rede, e acrescentando outro arco nessa rede. (c) Ainda outra rede de distribuição de água, obtida agora retirando dois arcos da primeira rede, e acrescentando outros dois arcos.

substituí-lo. Já a rede da Figura 1.7(c) é formada por uma perturbação maior: dois arcos são retirados, sendo incluídos dois outros arcos não existentes na configuração da rede da Figura 1.7(a).

O ponto relevante é: a rede da situação (b) é muito mais diferente da rede da situação (a) que a rede da situação (c). Se o leitor analisar as três redes, irá notar que o fluxo de água do nó-raiz até os nós consumidores muda completamente da situação (a) para a situação (b). Em muitos arcos da situação (b), o fluxo aumenta muito, em outros a direção do fluxo se inverte, em comparação com a rede da situação (a). Já na rede da situação (c), a mudança não é tão drástica: a maior parte dos fluxos de água, após a mudança de configuração, permanecem parecidos com os fluxos da situação (a). Os fluxos em alguns poucos ramos se alteram, mesmo assim não tão drasticamente quanto no caso anterior. A maioria dos fluxos permanece constante. Ou seja, ao contrário do que seria previsto pela métrica do número de perturbações, a rede (a) está mais distante da rede (b) que da rede (c).

Parece, portanto, que a modificação de algumas variáveis em um problema combinatório deveria significar uma alteração na distância maior que a modificação em outras variáveis, talvez à semelhança daquilo que ocorre quando perturbamos os algarismos de um número: a perturbação nos algarismos das unidades representa mudança menor que a perturbação nos algarismos das centenas ou dos milhares. Uma discussão a esse respeito é apresentada no Capítulo 13 deste livro.

O problema geral que se coloca pode ser enunciado dessa forma: a propriedade de localidade fraca é necessária para que seja possível a construção de algoritmos de otimização via Computação Evolutiva. No entanto, para que essa propriedade se manifeste em um problema, é preciso que a noção de distância esteja adequadamente construída, de forma que os algoritmos possam explorá-la. Em síntese, um algoritmo precisa "saber" quando uma solução é próxima de outra, para poder tomar uma decisão quanto a se uma solução que ainda não foi examinada deva ser ou não avaliada. Essa questão de definir o que é distância entre duas soluções, e consequentemente definir o que é a vizinhança de uma solução, ou ainda o que é uma trajetória que liga duas soluções, ou outros conceitos relacionados que servem de base para definir as operações realizadas pelos algoritmos, de uma forma ou de outra perpassa toda a discussão a respeito de algoritmos evolutivos e metaheurísticas para problemas combinatórios. Dessa maneira, os capítulos 10 e 12 deste livro também têm conexão com essa discussão.

5. Conclusões

Este livro irá tratar do conjunto de técnicas que vêm sendo desenvolvidas para tratar dos problemas de otimização dotados de pouca estrutura, ou seja, daqueles problemas sobre os quais se supõe haver uma localidade do tipo fraca em seu espaço de soluções. O livro está organizado em três partes. A Parte I, que compreende os capítulos 2 ao 6, agrega os métodos chamados de bio-inspirados, que foram desenvolvidos por analogia com algum processo biológico de adaptação. A Parte II, que inclui os capítulos 7 a 12, apresenta os métodos que foram desenvolvidos sem estarem ancorados nesse tipo de analogia.

Por fim a Parte III, intitulada de Tópicos Avançados, apresenta um apanhado de questões específicas que são transversais a todos os métodos. Nos capítulos 13, 14 e 15 e são abordadas questões que são comuns a algoritmos diversos, e que surgem em contextos bastante diversificados: a questão da codificação das soluções, o tratamento de restrições e o tratamento de incertezas. Os capítulos 16 e 17 abordam uma classe de problemas que hoje é central na Teoria de Otimização: os Problemas Multicritério. O capítulo 16 discute a questão da tomada de decisão nesse contexto, no qual faz-se necessário escolher uma dentre as múltiplas soluções eficientes disponíveis, sendo a escolha feita por meio da interação com um tomador de decisão, enquanto o capítulo 17 discute os algoritmos evolutivos empregados para se fazer a determinação do chamado conjunto de soluções eficientes nessa classe de problemas.

A Computação Evolutiva é usualmente classificada, hoje em dia, como um corpo de técnicas per-

tencentes a um campo do conhecimento um pouco maior, denominado *Inteligência Computacional*. Outras técnicas incluídas na Inteligência Computacional seriam, por exemplo, as *Redes Neurais Artificiais* e os *Sistemas Fuzzy*. O capítulo 18 aborda a aplicação de algoritmos evolutivos a problemas de classificação. Essa aplicação além de ser de grande relevância *per si*, cumpre neste livro o papel de ilustrar a conexão da temática aqui tratada com outros ramos da Inteligência Computacional – uma vez que esse tipo de problema é mais comumente abordado por meio das chamadas técnicas de *Aprendizado de Máquina*, nas quais se incluem, por exemplo, as Redes Neurais Artificiais.

(Página deixada propositadamente em branco)

PARTE I

Métodos Bio-Inspirados

(Página deixada propositadamente em branco)

CAPÍTULO 2

Algoritmos Genéticos

Francisco J. B. Pereira

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra

Os Algoritmos Genéticos (AGs) são métodos de pesquisa probabilísticos inspirados nos princípios da selecção natural e da genética. Foram desenvolvidos por John Holland e pelos seus alunos da Universidade de Michigan durante os anos 1960 (Holland, 1975). De acordo com Goldberg (1989), a investigação tinha um duplo objectivo: efectuar um estudo rigoroso dos mecanismos de adaptação existentes na natureza e desenvolver modelos computacionais que retivessem os princípios básicos identificados nos sistemas naturais.

1. A Inspiração Biológica

A evolução biológica efectua uma pesquisa num espaço de grande dimensão e complexidade, constituído por todas as possíveis combinações genéticas que podem ser geradas. Alguns dos elementos pertencentes a este espaço de procura poderão originar organismos viáveis. O processo evolutivo contribui para a obtenção de um conjunto de indivíduos bem adaptados ao meio ambiente em que se encontram.

De acordo com os pressupostos apresentados por Darwin, o processo de evolução natural recorre a dois mecanismos básicos: a **selecção** e a **reprodução com variação** (Darwin, 1859). A selecção

garante que os indivíduos mais aptos (os que melhor se adaptam ao meio ambiente) têm maior probabilidade de sobreviver. Graças a esta situação, têm a possibilidade de gerar um maior número de descendentes, propagando assim as suas características ao longo das gerações. A existência de um mecanismo de variação associado à reprodução garante que os descendentes gerados não são uma cópia fiel dos progenitores. A combinação destas duas forças permite que, ao longo de sucessivas gerações, a população de indivíduos evolua de forma gradual.

A formulação, no início do século XX, da teoria genética da hereditariedade permitiu integrar o conceito de gene no processo evolutivo, tornando mais claro de que forma é que as alterações são introduzidas na população. A variabilidade das espécies passa a ser explicada através da aplicação de um conjunto de operadores genéticos, como o operador de recombinação[1] ou de mutação. A moderna teoria da evolução natural, resultante da fusão das ideias originais de Darwin com a genética Mendeliana é, normalmente, designada por Neodarwinismo.

Os princípios básicos da evolução biológica serviram de inspiração para o desenvolvimento de algoritmos de pesquisa e adaptação em ambientes computacionais, tendo em vista aplicações, tanto no campo da engenharia, como no da biologia. A questão central associada ao surgimento dos AGs é colocada de forma simples por John Holland: "Como transformar as ideias de Darwin em algoritmos?"(Holland, 2000).

2. Estrutura de um Algoritmo Genético

Adoptando a inspiração natural, os AGs processam conjuntos de elementos do espaço de procura (potenciais soluções para o problema). Estes conjuntos, usualmente denominados populações, são transformados (evoluídos) ao longo de sucessivas iterações (gerações). O objectivo é encontrar uma solução de qualidade elevada (idealmente a solução óptima) para o problema que está a ser resolvido. No início são escolhidas aleatoriamente várias soluções. A partir deste conjunto inicial, e de acordo com os dois mecanismos básicos utilizados pela evolução natural referidos anteriormente, as transformações na população de soluções são efectuadas da seguinte forma:

- Os elementos mais aptos de uma determinada geração são seleccionados para servirem de progenitores das soluções que irão aparecer na geração seguinte. Na maior parte dos casos, o processo de selecção é probabilístico;
- Operadores de transformação, designados operadores genéticos, actuam sobre os elementos seleccionadas originando as novas soluções.

Existem duas decisões cruciais que é necessário tomar quando se pretende aplicar um AG na resolução de um problema:

- Escolha da representação para as soluções que fazem parte do espaço de procura: a proposta original de Holland defendia a utilização de um código binário para efectuar a representação das possíveis soluções de um problema. Desta forma, os indivíduos processados por um AG clássico são simplesmente sequências de 0's e 1's codificando a informação necessária para representar um ponto do espaço de procura;
- Definição da função de aptidão que associe a cada solução uma medida de qualidade, representando a sua capacidade para resolver o problema em causa.

Para além da inspiração biológica, os AGs adoptaram um conjunto de designações para qualificar situações e entidades relacionadas com o seu modo de funcionamento. Assim, o conjunto das

[1] O operador de *recombinação* também é chamado, por vezes, de operador de *cruzamento*.

soluções que vão sendo processadas é usualmente designado por **população**. Cada uma das iterações do processo de optimização é considerada uma **geração** e a sequência das várias populações reflecte a **evolução** ao longo do tempo. Os elementos que constituem uma determinada população são designados por **indivíduos** ou **cromossomas**. Os **genes** são os constituintes básicos dos cromossomas, podendo ser encarados como cada uma das características elementares de uma solução. Estão localizados numa determinada posição ou **locus** e cada um deles pode tomar um valor de entre um conjunto de possibilidades, denominadas **alelos**.

A estrutura genérica de um AG é ilustrada no Algoritmo 1. Nele pode ser consultado o processo iterativo que vai gerando sucessivas populações (passos 4 a 9).

Algoritmo 1 Estrutura Genérica de um Algoritmo Genético

1: $t \leftarrow 0$
2: Gerar a população inicial $P(t)$
3: Avaliar os indivíduos de $P(t)$
4: **repita**
5: Seleccionar progenitores $P'(t)$ a partir de $P(t)$
6: Aplicar operadores genéticos a $P'(t)$ obtendo a nova população $P(t+1)$
7: Avaliar P(t+1)
8: $t \leftarrow t+1$
9: **até** (critério de terminação atingido)
10: Devolver resultado final da optimização

Devido ao mecanismo de selecção, a qualidade média dos elementos que constituem a população tem tendência para aumentar ao longo do tempo. Os operadores genéticos são os responsáveis pela obtenção de novas soluções, ao mesmo tempo que tentam garantir que o processo mantenha um nível adequado de diversidade. Um AG possui dois tipos de operadores genéticos fundamentais: recombinação e mutação. Os detalhes de implementação de cada um destes operadores dependem da representação adoptada. Tal como as duas questões previamente identificadas (representação e aptidão), este é outro ponto fulcral e deve ser abordado com grande cuidado, de modo a maximizar a eficácia do processo de optimização. Existem, no entanto, princípios genéricos associados ao funcionamento destes operadores que são descritos a seguir:

- **Recombinação**: Operador estocástico que efectua a troca de material genético entre dois progenitores, conduzindo à criação de dois novos indivíduos (os descendentes). Estes serão formados por sequências genéticas parciais de cada um dos elementos originais. O operador de recombinação é considerado o principal responsável pela pesquisa efectuada por um AG. O objectivo central da sua aplicação é simples de compreender: ele actua sobre indivíduos que foram seleccionados na população actual, ou seja, são soluções com algum potencial. A combinação de características complementares de indivíduos promissores poderá original novas soluções que integrem as vantagens de ambos e que possuam uma aptidão acrescida para o problema que está a ser resolvido.

- **Mutação**: Operador unário que actua sobre as soluções resultantes da recombinação e que altera ligeiramente algumas das suas características. Num AG, a mutação é um processo completamente aleatório e tem como objectivo manter um nível de diversidade adequado na população, garantindo que alelos que eventualmente desapareçam tenham a possibilidade de reaparecer. Existem variantes de algoritmos evolutivos descritos em outros capítulos deste manual, nos quais a mutação desempenha um papel diferente.

Na proposta original de Holland (1975), é descrito um terceiro operador genético denominado **inversão**. Este operador unário tem a capacidade de proceder a um reordenamento parcial dos genes

existentes no cromossoma com o objectivo de tornar a aplicação do operador de recombinação mais eficaz. Embora faça parte da descrição original de um AG, a inversão é muito menos utilizada do que os dois outros operadores referenciados[2]. O próprio Holland, num trabalho recente (Holland, 2000), refere-se à recombinação e à mutação como os dois principais operadores genéticos de um AG. Por esse motivo, apenas estes dois operadores serão apresentados neste texto.

À medida que o número de gerações aumenta, um AG converge gradualmente para regiões do espaço de procura onde se encontram soluções promissoras. A optimização termina quando um determinado critério de terminação é atingido. Os critérios mais comuns são:

- Número limite de gerações atingido;
- Descoberta de uma solução com qualidade pretendida;
- Inexistência de melhoria durante um determinado período de tempo.

Ao atingir um critério de terminação, o AG devolve o resultado final da optimização. Existem duas alternativas para apresentação de resultados: devolver a melhor solução encontrada ao longo da optimização ou devolver um conjunto de indivíduos de qualidade elevada (por exemplo, devolver todos os indivíduos que fazem parte da última geração).

Bibliografia Básica

John Holland apresentou a sua ideia original sobre AGs no livro *Adaptation in Natural and Artificial Systems*, originalmente publicado em 1975 (Holland, 1975). É uma obra rigorosa, onde os principais conceitos relacionados com o funcionamento destes algoritmos são apresentados e analisados. É de leitura essencial para quem pretenda compreender os fundamentos teóricos associados ao funcionamento de um AG.

Ao longo dos anos têm sido publicados outros textos introdutórios que são uma excelente porta de entrada para quem pretende perceber os principais conceitos relacionados com o funcionamento de um AG. Todos estes livros descrevem exemplos práticos de aplicação, o que facilita a compreensão dos conceitos apresentados. O livro de David Goldberg *Genetic Algorithms in Search, Optimization, and Machine Learning*, apesar de ter sido publicado em 1989, mantém a sua actualidade (Goldberg, 1989). Alternativas mais recentes e igualmente relevantes são os livros *Genetic Algorithms + Data Structures = Evolution Programs* de Michalewicz (1992), *An Introduction to Genetic Algorithms* de Mitchell (1996) ou *An Introduction to Evolutionary Computing* de Eiben e Smith (2003).

A obra *Handbook of Evolutionary Computation* editada por Bäck et al. (1997) agrupa um conjunto muito abrangente de contribuições relativas a toda a área da computação evolutiva. É de consulta obrigatória sempre que se pretenda aprofundar ou procurar referências adicionais sobre um assunto específico. Por sua vez, o livro *Inteligência Artificial: Fundamentos e Aplicações* de Costa e Simoes (2008) merece uma referência. Embora seja um livro genérico de introdução à área da inteligência artificial possui uma secção importante dedicada a algoritmos evolutivos. Para alguns leitores poderá ter a vantagem adicional de ser escrito em português. Finalmente, ao longo do texto apresentamos diversas referências bibliográficas que podem servir de introdução aos diferentes tópicos abordados.

3. Exemplo: Aplicação de um AG ao Problema da Mochila

Em termos práticos, para aplicar um AG a um determinado problema é necessário definir quatro componentes essenciais: representação para as soluções, função de aptidão, método de selecção

[2] A razão para este facto está, provavelmente, relacionada com o surgimento de representações e de operadores de recombinação alternativos que tornam a sua aplicação menos importante.

Tabela 2.1: Propriedades dos objectos da instância do KSP a optimizar

	Obj_1	Obj_2	Obj_3	Obj_4	Obj_5	Obj_6	Obj_7	Obj_8
Peso	10	18	12	14	13	11	8	6
Valor	5	8	7	6	9	5	4	3

e operadores genéticos. É ainda necessário escolher valores para alguns parâmetros que controlam o funcionamento do AG, tais como o tamanho da população, o número máximo de gerações a processar (assumindo que este é o critério de terminação adoptado) ou a probabilidade de aplicação dos operadores genéticos. Para além destes, podem existir outros parâmetros específicos das componentes que constituem o AG.

O Problema da Mochila

O problema da mochila[3] (KSP) é um exemplo clássico de uma situação de optimização combinatória e pode ser enunciado da seguinte forma: dado um conjunto de objectos, cada um deles com um peso e um valor específico, determinar o subconjunto mais valioso cujo peso total não ultrapasse a capacidade de uma mochila.

O problema pode ser descrito mais formalmente: existe um conjunto S com N objectos, sendo cada objecto Obj_i caracterizado por um peso P_i e por um valor V_i, $i = 1, ..., N$ (todos os pesos e todos os valores são positivos). Existe, além disso, uma mochila com capacidade $C > 0$. O objectivo é encontrar um subconjunto de objectos que satisfaça as seguintes condições:

$$\max \sum_{i=1}^{N} x_i \times V_i \tag{2.1}$$

$$\text{sujeito a } \sum_{i=1}^{N} x_i \times P_i \leq C \tag{2.2}$$

Nas equações 2.1 e 2.2, $x_i \in \{0, 1\}$ ($x_i = 1$ se o objecto Obj_i se encontra na mochila, $x_i = 0$ caso contrário).

Implementação do AG

As características da instância do KSP que vai ser utilizada para ilustrar o funcionamento de um AG são as seguintes[4]:

- Número de objectos: 8
- Capacidade da mochila: 35
- O peso e o valor de cada um dos objectos podem ser consultados na tabela 2.1.

[3] Do inglês, *Knapsack Problem*. Na literatura, o problema descrito nesta secção é conhecido como 0-1 Knapsack Problem. Existem diversas variantes do KSP como, por exemplo, o Multidimensional Knapsack Problem.

[4] A instância escolhida é extremamente simples e serve apenas para exemplificar o funcionamento de um AG.

Representação

Uma representação comum para este problema é uma sequência binária com N posições (em que N é o número de objectos existentes). Cada posição está associada a um objecto: se tiver o valor 1 significa que foi escolhido para ser incluído na mochila. Em relação ao exemplo que estamos a considerar, a sequência {10100000} é uma solução possível que especifica que os objectos Obj_1 e Obj_3 se encontram na mochila.

A representação proposta é a mais natural para o problema do KSP. Para além disso, é uma representação binária, o que a torna especialmente apropriada para aplicação de um AG simples. Possui, no entanto, uma limitação importante, uma vez que permite a existência de muitas soluções inválidas no espaço de procura. Uma solução inválida no KSP ocorre quando o peso total do conjunto de objectos escolhidos excede a capacidade total da mochila. Uma vez que a representação não inclui nenhum mecanismo que limite a quantidade de objectos a escolher (por exemplo, a solução {11111111} faz parte do espaço de procura definido por esta representação), o processamento de soluções inválidas por parte do AG é inevitável. Existem várias alternativas para lidar com esta situação. A lista seguinte enuncia as três mais relevantes:

- **Penalizar**: A opção mais simples é tratar das soluções inválidas durante a avaliação. De acordo com este princípio, as soluções ilegais são penalizadas. A amplitude da penalização é proporcional ao grau de violação das restrições.

- **Reparar**: A reparação de uma solução inválida é uma possibilidade interessante, uma vez que permite manter apenas indivíduos legais na população. Esta abordagem obriga a desenvolver um algoritmo de reparação simples e rápido, específico para o problema a optimizar. O principal ponto fraco desta estratégia é a eventual dificuldade em encontrar um algoritmo de reparação. Em alguns casos, a correcção de uma solução ilegal pode ser uma tarefa extremamente díficil de resolver.

- **Alterar Representação**: Existem sempre várias representações possíveis para o mesmo problema. A decisão de qual a representação mais apropriada para uma determinada situação deve levar em consideração as propriedades do espaço de procura associado a cada uma delas (por exemplo, se admitem ou não soluções ilegais). No entanto, é importante referir que este não deve ser o único critério a considerar quando se procede à escolha da representação. É essencial considerar as vantagens e desvantagens associadas a cada hipótese e optar pela que oferece mais garantias em termos de eficácia.

Neste ponto optamos pela abordagem baseada em penalização. Na secção 5 serão descritas e analisadas abordagens alternativas.

Aptidão

A função de aptidão atribui a qualidade a uma solução, especificando a sua adequação para o problema em causa. A qualidade é estabelecida de forma quantitativa, o que facilita a comparação entre o mérito relativo de diferentes soluções.

No caso do KSP, o critério essencial para determinar a qualidade de uma solução é o valor total proporcionado pelos objectos escolhidos. É formalizado como um problema de maximização, pelo que a função de aptidão deve atribuir valores mais elevados aos melhores indivíduos. No entanto, e dada a representação adoptada, é necessário incluir na função de aptidão um mecanismo que penalize as soluções inválidas. Se esta componente não fosse considerada, é trivial verificar que a melhor solução para o problema seria a que escolhe todos os objectos para serem incluídos na mochila. Claramente esta é uma solução inválida, uma vez que o peso total dos objectos excede a capacidade da mochila.

A avaliação da aptidão de uma solução S é feita recorrendo à equação 2.3:

$$Qualidade(S) = \sum_{i=1}^{N} x_i \times V_i - Pen(S) \tag{2.3}$$

em que $x_i \in \{0, 1\}$ ($x_i = 1$ se o objecto Obj_i se encontra na mochila, $x_i = 0$ caso contrário) e $Pen(S)$ é a penalização associada à solução S. O valor da penalização pode ser calculado de diversas formas. Deve, no entanto, obedecer a três princípios básicos: i) a penalização é proporcional ao grau de violação da restrição de capacidade; ii) nunca deve compensar exceder a capacidade da mochila; iii) a penalização de um índivíduo válido é 0. Neste texto adoptamos uma penalização linear, calculada de acordo com a expressão apresentada na equação 2.4. Para uma descrição de outros tipos de penalização, consultar (Costa e Simoes, 2008; Michalewicz, 1992), e o capítulo 14 deste livro.

$$Pen(S) = \begin{cases} 0, & \text{se S legal;} \\ \rho \times (\sum_{i=1}^{N} x_i \times P_i - C), & \text{caso contrário} \end{cases} \tag{2.4}$$

O valor de ρ é obtido através da seguinte expressão:

$$\rho = max_{i=1,\ldots,N} \frac{V_i}{P_i} \tag{2.5}$$

AG: Simulação de uma Iteração

Estabelecidas a representação e a aptidão, o AG pode iniciar a optimização. Será ainda necessário especificar alguns valores para parâmetros, mas eles serão apresentados à medida que forem necessários.

Nesta secção apresentamos detalhadamente todas as etapas que constituem um ciclo iterativo completo do AG. De acordo com o algoritmo 1, a criação da população inicial é a primeira tarefa a ser realizada. Existe um parâmetro, o **tamanho da população**, que indica quantos indivíduos fazem parte de cada geração. Num AG simples, o seu valor mantém-se constante ao longo da optimização. A dimensão da população a escolher para um caso concreto depende de vários factores, mas os valores padrão utilizados na maioria das situações pertencem ao intervalo [100, 500]. No exemplo descrito nesta secção adoptamos uma população de apenas 10 indivíduos para manter a explicação o mais simples possível.

A população inicial é gerada aleatoriamente. Considerando a representação adoptada e a instância do KSP que está a ser optimizada, é necessário apenas gerar 10 sequências binárias de tamanho 8. Após serem gerados os 10 indivíduos da população inicial, aplica-se a equação 2.3 para calcular a qualidade de cada um deles. A tabela 2.2 apresenta um resumo dos individuos gerados e avaliados. Como se pode verificar, na população inicial existem 3 indivíduos inválidos. A melhor solução (I_5) tem qualidade 17.0, correspondendo à escolha dos objectos Obj_2 e Obj_5.

O processo que conduz à criação da população da nova geração inicia-se com a selecção dos progenitores. Neste passo, os indivíduos de melhor qualidade da geração actual têm maior probabilidade de serem seleccionados para dar origem às novas soluções. Vários métodos de selecção têm sido apresentados por diversos autores. No trabalho original de Holland é proposta a selecção por roleta (ou selecção proporcional) (Holland, 1975). Com este método, a probabilidade de um indivíduo ser seleccionado baseia-se na razão entre a sua qualidade e a soma das qualidades de todos os indivíduos da população. Diversas desvantagens têm sido associadas à selecção por roleta (Eiben e Smith, 2003). A principal é a dificuldade em controlar a pressão selectiva, i.e., em estabecer uma relação apropriada entre o mérito relativo de um indivíduo e a probabilidade de ser seleccionado como progenitor. Situações em que a pressão selectiva é demasiado elevada, i.e., quando apenas um pequeno grupo dos melhores indivíduos conseguem ser escolhidos como progenitores, podem levar o algoritmo a convergir prematuramente.

Tabela 2.2: População Inicial

Indivíduos	$\sum V_i$	$\sum P_i$	Penalização	Qualidade
$I_1 = \{10100000\}$	12	22	0	12.0
$I_2 = \{10100010\}$	16	30	0	16.0
$I_3 = \{00010011\}$	13	38	2.1	10.9
$I_4 = \{00010000\}$	6	14	0	6.0
$I_5 = \{01001000\}$	17	31	0	17.0
$I_6 = \{00000010\}$	4	8	0	4.0
$I_7 = \{01110110\}$	30	63	19.4	10.6
$I_8 = \{01000100\}$	13	29	0	13.0
$I_9 = \{10101101\}$	29	62	18.7	10.3
$I_{10} = \{00011000\}$	15	27	0	15.0

Por outro lado, se a pressão selectiva for demasiado baixa, i.e., se todos os indivíduos tiverem aproximadamente a mesma probabilidade de serem escolhidos como progenitores, o algoritmo deixa de ter a capacidade de identificar e convergir para áreas promissoras no espaço de procura.

Neste texto optamos por aplicar a **selecção por torneio** (Eiben e Smith, 2003; Mitchell, 1996). Este método apresenta duas características importantes que o ajudam a controlar a pressão selectiva: por um lado, não está excessivamente dependente dos valores concretos da qualidade dos indivíduos. A única informação de que necessita para fazer a selecção é uma hierarquização das soluções (em termos de mérito). Em segundo lugar, possui um parâmetro designado **tamanho do torneio** que ajuda a controlar explicitamente a pressão selectiva. O algoritmo 2 ilustra como a selecção por torneio escolhe um progenitor.

Num AG simples, o tamanho da população mantém-se constante ao longo das gerações. Se a população for constituída por P indivíduos, é necessário seleccionar P progenitores para dar origem aos P descendentes que formarão a nova geração. Deste modo, o processo descrito no algoritmo 2 é repetido P vezes. Num processo de selecção alguns elementos (os mais aptos) serão seleccionados várias vezes, enquanto que outros poderão nunca ser escolhidos. É esta pressão evolutiva que possibilita a convergência gradual do algoritmo para áreas promissoras do espaço de procura. Tal como foi referido anteriormente, o tamanho de torneio ajuda a controlar a pressão selectiva. Em regra, este parâmetro assume um valor entre 2 e 5, sendo raro encontrar situações em que seja superior. Numa situação de torneio binário (tamanho de torneio igual a 2), a pressão selectiva é mínima[5]. Pelo contrário, se o tamanho de torneio for 5, a pressão selectiva já é considerável[6].

No exemplo do KSP que está a ser analisado, é adoptado um torneio binário. A tabela 2.3 ilustra o processo de selecção e indica quais os indivíduos escolhidos como progenitores. Cada linha mostra os dois indivíduos envolvidos num torneio e apresenta o vencedor (o que tem melhor qualidade)

Após a selecção, os progenitores são agrupados em pares, aos quais é aplicado o operador de recombinação. Neste exemplo recorremos ao operador original proposto por Holland (1975), designado **recombinação com um ponto de corte**. De acordo com este operador, os dois indivíduos são alinhados, um ponto de corte é escolhido aleatoriamente ao longo dos cromossomas e os descendentes

[5] O número esperado de cópias do melhor elemento da população actual no conjunto de progenitores é 2 e apenas a pior solução está garantidamente afastada do processo reprodutivo.

[6] Como é evidente, o tamanho da população também influencia a pressão selectiva.

Algoritmo 2 Selecção por torneio com tamanho de torneio k

1: Escolher da população actual k indivíduos aleatoriamente
2: Seleccionar o melhor dos k indivíduos como progenitor

Tabela 2.3: Selecção dos progenitores

Torneio	Participantes	Vencedor
1	$\{I_8(13.0),\ I_3(10.9)\}$	I_8
2	$\{I_1(12.0),\ I_5(17.0)\}$	I_5
3	$\{I_4(6.0),\ I_9(10.3)\}$	I_9
4	$\{I_4(6.0),\ I_{10}(15.0)\}$	I_{10}
5	$\{I_8(13.0),\ I_3(10.9)\}$	I_8
6	$\{I_9(10.3),\ I_{10}(15.0)\}$	I_{10}
7	$\{I_7(10.6),\ I_3(10.9)\}$	I_3
8	$\{I_5(17.0),\ I_2(16.0)\}$	I_5
9	$\{I_7(10.6),\ I_5(17.0)\}$	I_5
10	$\{I_7(10.6),\ I_2(16.0)\}$	I_2

são criados a partir de sequências complementares dos dois progenitores. A figura 2.1 resume o modo de actuação do operador de recombinação com um ponto de corte. Existem outras propostas, como a recombinação com vários pontos de corte ou a recombinação uniforme, que generalizam o modo de actuação deste operador para representações binárias. A descrição de seu funcionamento pode ser consultada em (Eiben e Smith, 2003; Booker et al., 1997). A probabilidade de aplicação da recombinação é um parâmetro do AG e define a probabilidade deste operador ser efectivamente aplicado a cada um dos pares de progenitores. O seu valor é usualmente elevado (valores típicos entre 0.7 e 1.0), uma vez que é o principal responsável pela construção de boas soluções. Nos casos em que o operador de recombinação não é aplicado, os progenitores envolvidos passam para a fase seguinte sem alterações.

Na tabela 2.4 exemplifica-se o efeito da aplicação da recombinação com um ponto de corte aos elementos escolhidos anteriormente. Considera-se que os pares de progenitores são formados de acordo com a ordem pela qual foram seleccionados[7]. A barra vertical identifica o ponto de corte em cada

[7] Deve evitar-se que o operador de recombinação seja aplicado a duas cópias do mesmo indivíduo.

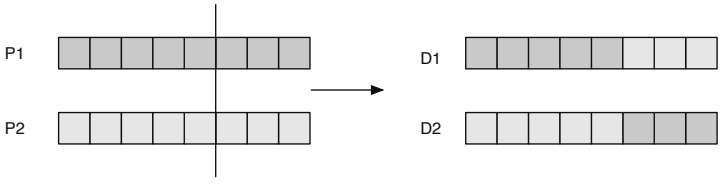


Figura 2.1: Recombinação com um ponto de corte

Tabela 2.4: Aplicação do operador de recombinação com um ponto de corte aos indivíduos seleccionados. A barra vertical identifica o local de corte em cada par de progenitores.

Progenitores	Descendentes
{0 1 0 | 0 0 1 0 0}	{0 1 0 0 1 0 0 0}
{0 1 0 | 0 1 0 0 0}	{0 1 0 0 0 1 0 0}
{1 0 1 0 | 1 1 0 1}	{1 0 1 0 1 0 0 0}
{0 0 0 1 | 1 0 0 0}	{0 0 0 1 1 1 0 1}
{0 1 0 0 0 1 0 0}	{0 1 0 0 0 1 0 0}
{0 0 0 1 1 0 0 0}	{0 0 0 1 1 0 0 0}
{0 1 0 0 0 1 0 | 0}	{0 1 0 0 0 1 0 1}
{0 0 0 1 0 0 1 | 1}	{0 0 0 1 0 0 1 0}
{0 1 0 0 1 0 | 0 0}	{0 1 0 0 1 0 1 0}
{1 0 1 0 0 0 | 1 0}	{1 0 1 0 0 0 0 0}

um dos pares. Neste exemplo assume-se que o operador de recombinação não é aplicado ao terceiro par, pelo que estes progenitores passam sem alteração para a fase seguinte. Pode ainda observar-se um fenómeno curioso na aplicação da recombinação ao primeiro par de soluções: embora os dois cromossomas sejam diferentes, a localização do ponto de corte faz com que os descendentes gerados sejam iguais aos progenitores.

Os indivíduos que resultam da recombinação são sujeitos a mutação. No exemplo que estamos a relatar adoptamos a **mutação binária**, o operador clássico para representações binárias (Holland, 1975; Goldberg, 1989). O seu modo de actuação é extremamente simples: quando aplicada a um gene do cromossoma (i.e., a um constituinte básico da solução que neste caso é um bit), troca o seu valor. A probabilidade de aplicação do operador de mutação é mais um parâmetro do AG, sendo definida em relação a genes individuais, ou seja, define qual a probabilidade de um gene sofrer uma mutação. O seu valor é bastante baixo, sendo vulgar encontrar valores entre 0.001 e 0.01. Tal como foi referido anteriormente, num AG este operador tem como função principal agitar a pesquisa, reintroduzindo diversidade e evitando que esta fique presa em óptimos locais de fraca qualidade.

Na tabela 2.5 exemplifica-se o efeito da aplicação da mutação binária no exemplo que está a ser descrito. Na coluna da esquerda surgem os indivíduos que resultam da recombinação. Os genes sublinhados são sujeitos a mutação, dando origem ao novo conjunto de indivíduos que surge na coluna da direita. Depois da aplicação do operador de mutação, o processo de criação da população da nova geração está concluído. As novas soluções são avaliadas (tabela 2.6) e, caso não tenha sido atingido nenhum dos critérios de terminação que estejam a ser considerados, pode iniciar-se imediatamente uma nova iteração do algoritmo.

Em resumo, a aplicação de um AG a uma instância do KSP obrigou a definir as seguintes componentes:

- Representação para as soluções;
- Função de aptidão;

Tabela 2.5: Aplicação do operador de mutação binária aos indivíduos que resultam da recombinação. Os genes sublinhados foram escolhidos para ser sujeitos a mutação.

Antes da mutação	Depois da mutação
{01$\underline{0}$01000}	{01101000}
{01000100}	{01000100}
{1010$\underline{1}$000}	{10100000}
{00011101}	{00011101}
{0$\underline{1}$0001$\underline{0}$0}	{00000110}
{00011000}	{00011000}
{01000101}	{01000101}
{00010$\underline{0}$10}	{00010110}
{0$\underline{1}\underline{0}$01010}	{00101010}
{1010$\underline{0}$000}	{10101000}

Tabela 2.6: Avaliação dos descendentes

Indivíduos	$\sum V_i$	$\sum P_i$	Penalização	Qualidade
$D_1 = \{01101000\}$	24	43	5.5	18.5
$D_2 = \{01000100\}$	13	29	0	13.0
$D_3 = \{10100000\}$	12	22	0	12.0
$D_4 = \{00011101\}$	23	54	13.2	9.8
$D_5 = \{00000110\}$	9	19	0	9.0
$D_6 = \{00011000\}$	15	27	0	15.0
$D_7 = \{01000101\}$	16	45	6.9	9.1
$D_8 = \{00010110\}$	15	33	0	15.0
$D_9 = \{00101010\}$	20	33	0	20.0
$D_{10} = \{10101000\}$	21	35	0	21.0

- Mecanismo de selecção;
- Operadores genéticos de recombinação e mutação;
- Critério de terminação.

Para além destas componentes, foi ainda necessário especificar valores para os seguintes parâmetros: tamanho da população, tamanho de torneio, probabilidade de aplicação da recombinação, probabilidade de aplicação da mutação e número máximo de gerações (assumindo que é este o critério de terminação adoptado).

Medidas de Desempenho

Quando se aplica um AG a um determinado problema é importante conseguir perceber se ele apresenta um bom desempenho. A avaliação pode ser feita em comparação com outro método de optimização ou pode servir para determinar qual a melhor configuração para o AG que está a ser analisado. Os AGs são métodos de pesquisa estocásticos pelo que as comparações devem ser efectuadas utilizando testes estatísticos[8]. Para que estes testes possam ser utilizados, a aplicação de um AG a um determinado problema deve ser repetida várias vezes. Só assim poderão ser obtidos dados suficientes para tornar a análise estatística significativa. O número de repetições de um AG é um parâmetro adicional que deve ser especificado[9].

As medidas mais utilizadas para aferir o desempenho de um AG são a eficiência e a eficácia. A primeira está directamente relacionada com a sua rapidez, enquando que a segunda mede a qualidade das soluções. Há três critérios comuns para estimar a qualidade:

- **Taxa de sucesso**: Se a qualidade da solução óptima for conhecida, pode contabilizar-se quantas repetições do AG a conseguem descobrir. A razão entre este valor e o total de repetições corresponde à taxa de sucesso do algoritmo.
- **Média das melhores soluções**[10]: Para cada uma das repetições realizadas determina-se a qualidade da melhor solução. Esta medida é obtida efectuando a média destes valores.
- **Melhor solução encontrada**[11]: Qualidade da melhor solução encontrada pelo AG.

Os três critérios descritos estão evidentemente relacionados. Além disso, dependendo do problema concreto e do objectivo da optimização, pode fazer mais sentido utilizar uma ou outra medida. Em (Eiben e Smith, 2003, cap. 14) pode ser consultada uma descrição detalhada das vantagens e limitações de cada um destes critérios.

4. Propriedades dos Algoritmos Genéticos

Apesar da sua simplicidade, os AGs possuem um conjunto de propriedades que os tornam especialmente robustos para efectuar pesquisas em espaços de grande dimensão e complexidade. Três das principais características que contribuem para esta situação são:

- Processamento simultâneo de uma população de potenciais soluções para o problema. Esta particularidade permite que diferentes áreas do espaço de procura sejam analisadas em paralelo, reduzindo a probabilidade de convergência prematura para um óptimo local.

[8] A apresentação dos testes existentes e das condições em que podem ser aplicados está fora do âmbito deste capítulo. Consultar (Zar, 2009) para uma apresentação detalhada deste tópico.

[9] Embora existam excepções, 30 repetições é um valor padrão normalmente aceite pela comunidade da computação evolutiva.

[10] Do inglês *mean best fitness*.

[11] Do inglês, *best-ever fitness*.

- Possuem componentes probabilísticas e não recorrem a informação específica sobre o problema que estão a resolver (a única informação de que necessitam é o resultado da aplicação da função de aptidão). Estes dois factores, combinados com a existência de uma população de soluções, tornam estes métodos particularmente versáteis e robustos.

- Equilíbrio entre exploração de novas regiões do espaço e conservação da informação já adquirida: a selecção actua como principal força de conservação, permitindo manter e consolidar a informação descoberta anteriormente. Por seu lado, através de transformações efectuadas nos elementos seleccionados, é promovida a exploração de novas regiões do espaço. Assumindo que este possui algum tipo de regularidade, as referidas transformações poderão conduzir à descoberta de novas regiões, onde se encontrem soluções de qualidade superior.

Fundamentação Teórica

Na sua proposta, Holland analisou teoricamente o desempenho dos AGs como estratégia de pesquisa. A descrição rigorosa da análise efectuada está fora do âmbito deste texto, aconselhando-se a consulta das obras (Holland, 1975; Goldberg, 1989) para um estudo mais detalhado. Neste ponto limitamo-nos a descrever de forma simplificada algumas das ideias principais. Toda a análise teórica definida por Holland é sustentada por dois conceitos básicos: a noção de **esquema** e a noção de **blocos construtores**. Um esquema é um modelo representativo de um determinado subconjunto de cromossomas que partilham os mesmos valores em algumas posições. Mais concretamente, e limitando a análise a alfabetos binários, um esquema é uma sequência constituída pelos símbolos $\{0, 1, *\}$. Posições contendo 0 ou 1 são consideradas definidas, enquanto que as que contêm o símbolo $*$ são consideradas irrelevantes para a análise[12]. A utilização de esquemas permite representar um subconjunto do espaço de procura de forma muito eficiente. Um cromossoma *ch* pertence a um determinado esquema E se e só se as sequências *ch* e E forem iguais em todas as posições em que E não contém um $*$. Considere-se o seguinte exemplo: o esquema $E = \{1 * * * 1\}$ representa todos os cromossomas binários com 5 genes que possuam um bit com valor 1 na primeira e na última posição. As sequências que se encaixam no esquema (por exemplo, 10001 ou 10101) designam-se instâncias de E. Dois atributos dos esquemas são a sua **ordem** (número posições definidas) e o seu **tamanho definido** (distância entre a primeira e última posições definidas). Analisando o esquema E, verifica-se que é de ordem 2 e que o seu tamanho definido é 4.

Cada indivíduo da população representa um número elevado de esquemas (exactamente 2^n, sendo n o número de genes do cromossoma). Como consequência, numa determinada geração, apesar de o AG apenas avaliar explicitamente os indivíduos que pertencem à população, está implicitamente a estimar a qualidade de um número muito elevado de esquemas. A qualidade de um esquema é definida como a qualidade média de todas as suas instâncias que pertencem à população. Segundo Holland (1975), é a existência deste paralelismo implícito que permite a um AG efectuar de forma eficiente a pesquisa de um espaço de procura de grande dimensão. O teorema fundamental dos AGs, também chamado **teorema dos esquemas**[13], enuncia o modo como a pesquisa é efectuada: "O número de instâncias de esquemas de ordem baixa, com tamanho definido reduzido e com qualidade acima da média aumenta de forma exponencial ao longo das sucessivas gerações de um AG". Por sua vez, a **hipótese dos blocos construtores** é uma consequência lógica do teorema anterior: "Um AG efectua a pesquisa de soluções de boa qualidade através da combinação de esquemas de pequena dimensão (i.e., ordem baixa e tamanho definido reduzido) e de qualidade acima da média. Estes esquemas são denominados os blocos construtores da solução".

A generalização da utilização de AGs conduziu a novas tentativas no sentido de entender o seu modo de funcionamento. Uma das questões centrais associadas à análise do desempenho de um AG

[12] Do inglês, *don't care symbol*.
[13] Do inglês: *schemata theorem*.

é a seguinte: "Em quais problemas é suposto um AG ser eficaz?". A consulta dos volumes da série *Foundations of Genetic Algorithms* é uma boa introdução para se obter uma ideia dos desenvolvimentos teóricos recentes desta área.

5. Extensões ao Algoritmo Genético Simples

O AG descrito na secção 3 possui as componentes básicas originalmente propostas por Holland (1975), sendo habitualmente identificado como AG simples ou AG canónico. A única excepção é o mecanismo de selecção adoptado. Uma consequência lógica da disseminação e popularização desta técnica é a introdução de alterações à proposta original. Ao longo dos anos, diversas modificações foram introduzidas na sua estrutura, tanto ao nível da representação e operadores utilizados, como ao nível da própria arquitectura do AG. Actualmente, as opções de implementação estão, na maioria das situações, dependentes do problema que se pretende resolver. Alguns exemplos de representações alternativas incluem a utilização de vectores de números reais, permutações, tabelas multidimensionais, sistemas de regras ou grafos. Uma das consequências desta situação é a necessidade de desenvolver operadores genéticos que se adaptem às novas representações (Eiben e Smith, 2003; Michalewicz, 1992). Nesta secção apresentamos alguns exemplos de alternativas ao algoritmo canónico.

Representação e Operadores Genéticos

A selecção da representação a adoptar para as soluções é crucial. Esta é provavelmente a decisão mais importante a tomar quando se efectua o desenvolvimento de um AG para aplicar num problema de optimização, até porque várias outras componentes (como, por exemplo, os operadores genéticos ou a função de aptidão) dependem desta decisão.

Alguns factores que é necessário considerar são:

- O alfabeto deve ser binário ou pode conter mais símbolos?
- A representação deve ser discreta ou contínua?
- O ordenamento dos genes no cromossoma é relevante?
- O cromossoma pode conter informação redundante?
- A representação permite a existência de soluções inválidas? Em caso afirmativo, como lidar com elas?

As decisões a tomar em relação aos tópicos enunciados estão directamente relacionadas com o problema a optimizar. Encontrar a representação apropriada para um problema, ou conjunto de problemas, é uma tarefa difícil e é um tópico importante de investigação. Quando confrontados com um caso concreto, a experiência acumulada na utilização de AGs e uma análise cuidada do problema a resolver ajudam a tomar uma decisão sobre qual poderá ser a melhor opção. Além disso, a realização de um conjunto preliminar de testes com várias alternativas pode ajudar a fundamentar melhor a escolha efectuada.

Para além da representação binária, a representação real e a baseada em permutações são as 2 alternativas que são adoptadas mais vezes por AGs. Nas próximas secções descrevemos brevemente as suas propriedades e apresentamos propostas de operadores genéticos para lidar com cada uma delas.

Representação real

Os problemas nos quais as variáveis de decisão tomam valores reais são muito comuns. Nestes casos, uma solução é representada através de uma sequência de números reais. Embora seja possível recorrer a uma sequência binária para a codificação[14], é mais natural adoptar uma representação real em que cada gene corresponde directamente uma variável de decisão. Cada um destes genes tem associado o seu próprio domínio. Quando se lida com problemas em domínios contínuos, esta representação tem a vantagem de ser mais natural, ou seja, está mais próxima das soluções que representa. Esta é uma característica importante que deve ser levada em consideração quando se está a decidir qual a representação mais apropriada para um dado problema.

Considere que se pretende minimizar a função apresentada na equação 2.6.

$$f(x_1, x_2) = 100 \times (x_1^2 - x_2)^2 + (1 - x_1)^2 \qquad -2.048 \leq x_1, x_2 \leq 2.048 \tag{2.6}$$

Para este problema, a representação real de uma solução é feita recorrendo a 2 valores reais, respectivamente para as variáveis x_1 e x_2. Neste caso, uma solução possível será: $\{-1.235, 0.023\}$. O número de casas decimais a adoptar depende da precisão que se pretende, embora esteja obviamente condicionada pelas limitações físicas associadas à implementação computacional.

Os operadores de recombinação existentes para representações reais dividem-se em dois grupos: recombinação discreta vs. recombinação aritmética. Os do primeiro grupo actuam de forma análoga aos operadores de recombinação binários: determinam um ou mais pontos de corte entre genes e criam descendentes juntando secções complementares dos progenitores. A principal desvantagem dos operadores discretos é a incapacidade que têm para introduzir novos valores na população, garantindo apenas novas combinações para alelos já existentes.

Os operadores de recombinação aritméticos permitem ultrapassar esta limitação. Com estes operadores, o valor de cada um dos genes dos descendentes é encontrado dentro do intervalo definido pelos respectivos alelos que surgem nos progenitores[15]. Na equação 2.7 apresentamos um exemplo de aplicação de um operador de recombinação aritmético. Dados dois progenitores $P1$ e $P2$, os descendentes $D1$ e $D2$ são criados de acordo com as seguintes expressões:

$$\begin{cases} D_1 = \lambda \times P1 + (1 - \lambda) \times P2 \\ D_2 = \lambda \times P2 + (1 - \lambda) \times P1 \end{cases} \tag{2.7}$$

Na equação $\lambda \in [0, 1]$, o que significa que este operador efectua uma média pesada dos valores dos progenitores para gerar os descendentes.

Em traços gerais existem dois tipos de operadores de mutação para representações reais. Na mutação uniforme, o novo valor é escolhido aleatoriamente do domínio do gene em questão. Pelo contrário, na mutação não-uniforme o valor actual do gene é tido em consideração e é alterado de uma forma mais limitada. A mutação gaussiana é o exemplo mais comum, sendo que neste caso o novo valor para um gene G_i é obtido através da seguinte expressão $G_i \leftarrow G_i + N(0, 1)$.

Representação em permutação

Para apresentar as propriedades das representações baseadas em permutações regressamos ao exemplo do KSP. No que diz respeito a este problema, para além da representação binária, várias outras propostas têm sido apresentadas e analisadas. Em (Hinterding, 1999; Raidl e Gottlieb, 2005) pode

[14] Em (Michalewicz, 1992) pode ser consultado um exemplo detalhado de todos os passos envolvidos neste processo. Ver ainda (Herrera et al., 1998) para um discussão das vantagens/desvantagens da utilização de representações binárias em problemas contínuos.

[15] Vários operadores de recombinação relaxam esta restrição e possibilitam que o novo valor saia ligeiramente do intervalo (Herrera et al., 1998). Esta modificação pode ajudar a combater a convergência prematura do algoritmo.

Tabela 2.7: Descodificação da solução $\{1, 8, 7, 4, 2, 6, 3, 5\}$ utilizando a heurística *first-fit*. O símbolo $\surd$ indica que o objecto foi seleccionado para a mochila.

	1	8	7	4	2	6	3	5
Mochila	$\surd$	$\surd$	$\surd$	X	X	$\surd$	X	X
Peso Acumulado	10	16	24	24	24	35	35	35

ser consultada uma lista abrangente das várias representações. Uma das mais estudadas é a representação baseada numa permutação de valores, originalmente proposta por Hinterding (1999). As permutações são representações habitualmente utilizadas em problemas em que é necessário encontrar uma sequência ideal para um conjunto de objectos. Entre os exemplos mais comuns, incluem-se o problema do caixeiro viajante, o problema do encaminhamento de veículos ou problemas de escalonamento.

As representações em permutação codificam uma sequência ordenada de símbolos, na qual não podem existir repetições. A ordem pela qual os símbolos surgem na representação e/ou as relações de adjacência são as características mais relevantes para definir a qualidade de uma solução. No caso do KSP, a representação consiste numa permutação de todos objectos que fazem parte do problema que está a ser optimizado. Considerando o exemplo com 8 objectos apresentado na secção 3, duas soluções possíveis são $\{1, 4, 7, 8, 2, 6, 3, 5\}$ ou $\{5, 2, 4, 1, 8, 7, 3, 6\}$[16].

A representação em permutação proposta para o KSP designa-se **indirecta**, uma vez que a consulta de uma solução não especifica imediatamente quais os objectos que se encontram na mochila. É necessário aplicar um algoritmo de descodificação para interpretar a informação armazenada no cromossoma e indicar qual a solução final. A ordem pela qual os objectos surgem no cromossoma representa a sua prioridade no que diz respeito à entrada na mochila. Uma heurística simples do tipo *first-fit* considera os objectos pela ordem em que se encontram na solução e adiciona-os, se eles não provocarem a violação da restrição de capacidade. Considerando a permutação $\{1, 8, 7, 4, 2, 6, 3, 5\}$ e as propriedades indicadas na tabela 2.1, na tabela 2.7 exemplifica-se o processo de descodificação. A solução final obtida indica que os objectos $\{1, 8, 7$ e $6\}$ foram seleccionados para entrar na mochila.

Existem vantagens e desvantagens associadas à adoptação de uma representação baseada em permutações para resolver o problema do KSP. Aliás, a um nível mais geral, existem sempre pontos fortes e pontos fracos associados a cada representação que pode ser considerada para um determinado problema. Estes devem ser identificados e analisados, uma vez que isto ajudará a fazer uma escolha mais fundamentada pela representação que se acredita ser mais adequada. Regressando à proposta baseada em permutações para o KSP, podem ser identificadas as seguintes vantagens/desvantagens[17]:

- **Vantagens**:
  - A representação em permutação garante que as soluções processadas pelo AG são sempre válidas. A existência de um algoritmo de descodificação assegura que a restrição de capacidade da mochila nunca é violada.
  - Uma vez que o AG apenas processa soluções válidas, é mais simples implementar a função de aptidão. Não é necessário utilizar nenhum mecanismo de penalização para punir indivíduos

[16] Para simplificar a notação, cada objecto é representado apenas pelo seu identificador numérico.

[17] Embora a análise seja feita para um problema particular, praticamente todos os tópicos são válidos para situações em que se opta por uma representação indirecta

Tabela 2.8: Aplicação do operador de recombinação com um ponto de corte a permutações. A barra vertical identifica o local de corte em cada par de progenitores.

Progenitores	Descendentes
{3 5 4 2 | 1 7 9 8 6}	{3 5 4 2 4 3 8 6 9}
{5 1 2 7 | 4 3 8 6 9}	{5 1 2 7 1 7 9 8 6}

ilegais. A comparação entre soluções também se torna mais clara, uma vez que o único critério a considerar é o valor total proporcionado pelos objectos seleccionados.

- **Desvantagens**:
  - A principal desvantagem associada à utilização de uma representação indirecta está directamente relacionada com a necessidade de desenvolver e aplicar um algoritmo de descodificação. Em muitos casos não é trivial encontrar um algoritmo que permita descodificar o cromossoma numa solução final legal. Esta situação não se verifica no problema do KSP que está a ser utilizado como exemplo, mas é comum em muitos problemas de optimização.
  - O algoritmo de descodificação é específico do problema e da representação, o que diminui a generalidade do AG. Além disso, e embora se pretenda que o algoritmo de descodificação seja o mais simples possível, existe sempre um tempo adicional de processamento que pode condicionar a eficiência do processo de optimização.

O modo de actuar dos operadores genéticos depende da representação adoptada. É fácil verificar que a aplicação da recombinação com um ponto de corte e da mutação binária a soluções codificadas como permutações gera indivíduos ilegais (ilegais, no sentido de não pertencerem ao espaço de procura)[18]. Na tabela 2.8 ilustra-se como a aplicação da recombinação com um ponto de corte pode originar soluções ilegais.

Na literatura existem várias propostas de operadores genéticos para permutações. Tal como referimos anteriormente, esta é uma representação muito utilizada em alguns dos mais relevantes problemas de optimização que ocorrem no mundo real, o que naturalmente conduziu ao surgimento de diversas alternativas. Um operador de recombinação deve ter a capacidade de, dadas duas permutações, gerar dois descendentes tais que: i) são permutações legais (i.e., contêm todos os símbolos uma e uma só vez); ii) são constituídos por informação relevante (i.e., blocos construtores) presente em cada um dos progenitores. No caso de uma permutação, a informação relevante inclui a posição absoluta ocupada por cada um dos símbolos e as relações de precedência e de adjacência que se verificam. Dependendo do tipo de problema que está a ser resolvido, pode ser mais importante considerar a posição absoluta dos símbolos como, por exemplo, em problemas de escalonamento ou as relações de adjacência, como no caso do caixeiro viajante. Alguns dos exemplos mais conhecidos de operadores são a recombinação por ordem (*Order crossover*), recombinação por ciclo (*Cycle crossover*), recombinação por aresta (*Edge crossover*) ou o PMX (*Partially mapped crossover*). Em (Eiben e Smith, 2003; Michalewicz, 1992) pode ser consultado o funcionamento detalhado de cada um destes operadores e uma lista de problemas para os quais são mais apropriados.

Neste capítulo apresentamos o operador de recombinação por ordem, originalmente proposto por Davis (1991). Este operador é especialmente apropriado para lidar com situações em que a informação

[18] Em sentido estrito, a mutação binária não pode ser aplicada a uma representação contendo símbolos inteiros. No entanto pode ser facilmente extendida para um operador que troque o valor de um gene para outro pertencente ao domínio.

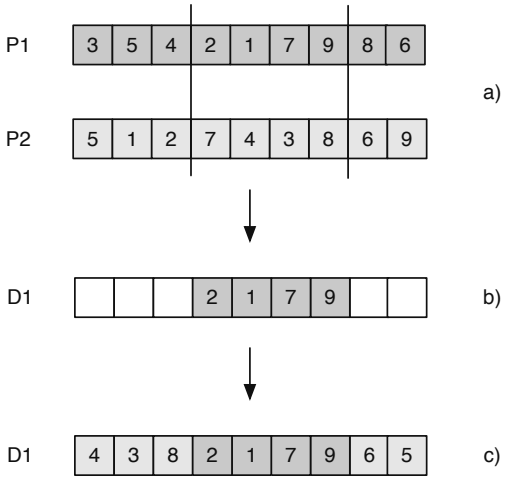


Figura 2.2: Recombinação por ordem

mais importante a transmitir aos descendentes é a ordem pela qual os símbolos aparecem nos progenitores. No algoritmo 3 podem ser consultados os passos executados por este operador para criar dois descendentes D1 e D2 a partir dos progenitores P1 e P2. A figura 2.2 apresenta um exemplo de aplicação deste operador. O painel a) da figura apresenta os dois progenitores e os locais dos pontos de corte. No painel b) pode ver-se a secção de P1 que é herdada por D1 e, finalmente, no painel c) surge o descendente já com os símbolos herdados de P2. A geração do descendente D2 fica como exercício para o leitor.

Algoritmo 3 Operador de recombinação por ordem. Gera dois descendentes D1 e D2 a partir dos progenitores P1 e P2.

1: Alinhar as sequências P1 e P2
2: Escolher aleatoriamente dois pontos de corte nas sequências P1 e P2. Os pontos de corte situam-se entre dois símbolos
3: Copiar a secção entre os pontos de corte de P1 para D1
4: Com início na posição a seguir ao segundo ponto de corte, copiar os restantes símbolos de P2 para D1. Os dois cromossomas P2 e D1 devem ser considerados circulares
5: Criar D2 repetindo os dois passos anteriores, mas trocando o papel desempenhado pelos progenitores P1 e P2.

Devido às propriedades das permutações, a aplicação da mutação não pode considerar genes individualmente. A alternativa é escolher subconjuntos de genes e alterar as suas posições. Existem vários operadores de mutação padronizados que são habitualmente aplicados a permutações: mutação por troca (*swap mutation*), mutação por inserção (*insert mutation*), mutação por deslocamento (*displacement mutation*) ou mutação por inversão (*inversion mutation*). Para uma descrição completa e detalhada do modo de funcionamento destes operadores consultar (Eiben e Smith, 2003; Michalewicz, 1992). Neste capítulo apresentamos dois exemplos simples que ilustram o modo de funcionamento de dois destes operadores. No painel a) da figura 2.3 apresentamos um exemplo de aplicação da mutação por troca: dois genes são escolhidos aleatoriamente no cromossoma e os seus valores são trocados; no painel b) da mesma figura, a mutação por inversão escolhe dois genes aleatoriamente e inverte a

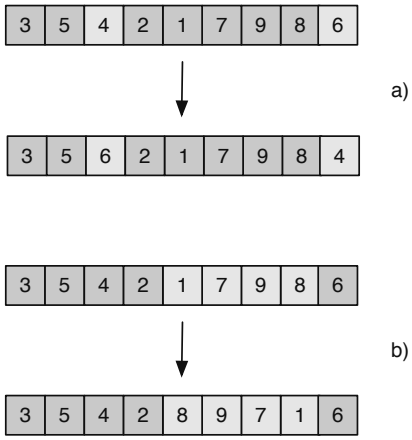


Figura 2.3: Exemplos de operadores de mutação para permutações: a) mutação por troca; b) mutação por inversão

ordem dos valores que se encontram entre essas duas posições.

Algoritmos de Reparação

Antes de propor novas formas de organização para o modo como os AGs processam soluções, é importante regressar por um momento à representação binária proposta para o KSP. Foram referidas na secção 3, três alternativas para lidar com soluções inválidas. Duas delas, penalização e desenvolvimento de representações alternativas, foram já discutidas. É altura de abordar brevemente a reparação de soluções. Esta abordagem possui vários pontos de contacto com a utilização de representações indirectas descrita anteriormente. Tal como neste caso existe uma heurística simples, neste caso designada algoritmo de correcção, que actua sobre as soluções inválidas da população, corrigindo-as[19]. O objectivo é garantir que todos os indivíduos avaliados são soluções legais. A definição de um algoritmo de correcção pode ser uma tarefa complicada. Em vários casos, poderá ser (quase) tão difícil corrigir uma solução inválida, como resolver o problema que se está a optimizar. Um caso paradigmático é o problema da construção dos horários de uma escola (Lewis, 2008). Devido a todas as restrições envolvidas, corrigir uma solução ilegal pode ser extremamente difícil.

No caso do KSP é trivial corrigir um indivíduo. Uma solução é ilegal porque a capacidade da mochila foi ultrapassada, logo, a correcção deve ir retirando objectos até que o problema esteja ultrapassado. O algoritmo 4 ilustra os passos a executar para corrigir a solução S. Embora este algoritmo não especifique como devem ser escolhidos os objectos a retirar da mochila, existem duas possibilidades: retirar os objectos de acordo com uma heurística (por exemplo, o objecto mais pesado ou o objecto com pior *ratio* entre valor e peso) ou retirá-los aleatoriamente. A primeira opção parece ser mais apropriada, mas a segunda hipótese não deve ser imediatamente descartada. Proporciona maior robustez e evita introduzir componentes sôfregas na pesquisa que poderão comprometer a eficácia global do algoritmo.

[19] Em traços gerais, as vantagens e desvantagens associadas à utilização de uma representação indirecta mantêm-se válidas.

Algoritmo 4 Reparação de uma representação binária para o KSP

1: **enquanto** $(\sum_{i=1}^{N} x_i \times P_i > C)$ **faça**
2: Escolher uma posição x_i com valor 1
3: $x_i \leftarrow 0$
4: **fim enquanto**

Arquitecturas alternativas

Num AG simples, as populações sucedem-se iteração após iteração sem que exista nenhum tipo de competição entre progenitores e descendentes. De acordo com esta organização, designada **geracional**, assim que termina a criação da nova população, a anterior desaparece. Ao longo dos anos, diversas alternativas de processamento das populações foram apresentadas com o objectivo de aumentar a eficácia dos AGs. Nas próximas secções descrevemos brevemente duas dessas propostas.

Elitismo

A qualidade média dos indivíduos que fazem parte das populações processadas por um AG aumenta ao longo das gerações. Este aumento é mais pronunciado no início da optimização e tem tendência para diminuir ou estagnar à medida que a população começa a convergir.

Pelo contrário, é relativamente comum que a qualidade da melhor solução diminua de uma geração para outra. O desaparecimento do indivíduo de melhor qualidade ocorre devido ao caracter probabilístico da selecção ou, mais provavelmente, devido à acção dos operadores genéticos. O **elitismo** evita que isto aconteça. O funcionamento desta estratégia é muito simples de descrever: quando termina a avaliação da população de descendentes verifica-se qual é a qualidade do melhor indivíduo. Se for inferior à qualidade da melhor solução da geração anterior, esta é recuperada e inserida na nova população. Como a população tem tamanho fixo, a adição de uma solução obriga a eliminar um dos descendentes que acabou de ser criado. A escolha do elemento sacrificado pode ser feita de forma aleatória ou de acordo com um critério pré-estabelecido (por exemplo, eliminar a solução de pior qualidade).

Algoritmos *steady-state*

Nos AGs do tipo *regime permanente*[20] deixa de existir a noção de gerações bem definidas em que os indivíduos são completamente substituídos em cada iteração (Whitley, 1989). Pelo contrário, os progenitores e descendentes passam a coexistir e a competir por um lugar na população. O primeiro autor a estudar esta possibilidade em AGs foi De Jong em 1975 (De Jong, 1975). No seu estudo definiu um parâmetro G (designado *generation gap*) que estabelece a fracção da população que é substituída em cada geração. O seu valor pode variar entre $G = 1$, correspondendo a uma organização geracional, até $G = 1/P$ (em que P é o tamanho da população), no qual apenas um indivíduo é substituído em cada iteração.

O algoritmo 5 ilustra o modo de funcionamento de um AG *regime permanente* para $G = 1/P$. Vários detalhes de implementação devem ser clarificados em relação ao funcionamento deste algoritmo (consultar (Lozano et al., 2004; Smith, 2007) para obter detalhes sobre as várias alternativas) :

- Selecção: o processo de selecção dos progenitores pode ser efectuado com um qualquer mecanismo de selecção (por exemplo, selecção por torneio), embora seja relativamente comum escolher as soluções aleatoriamente.

[20] Do inglês *steady-state*.

- Operadores genéticos: embora a recombinação crie habitualmente duas soluções, o algoritmo apresentado lida apenas com um descendente. Neste passo pode assumir-se que é escolhido o descendente de melhor qualidade (ou mesmo que a escolha é feita de forma aleatória).
- Domínio de substituição: existem duas possibilidades para seleccionar a solução Z que poderá ser substituída pelo descendente. A primeira restringe a escolha aos progenitores. A segunda alarga a escolha a toda a população ou a um subconjunto da população.
- Estratégia de substituição: Independentemente do domínio considerado, a escolha da solução Z a substituir pode ser feita de acordo com vários critérios. De entre as possibilidades, destacam-se a substituição por qualidade, por idade (i.e., a solução que está há mais tempo na população), por semelhança[21] ou mesmo aleatoriamente. É possível ainda considerar uma combinação de critérios para encontrar a solução Z a substituir.
- Condição de substituição: No último passo do algoritmo, a decisão sobre a entrada do descendente D na população está normalmente relacionada com a sua qualidade, ou seja, a solução é aceite se tiver melhor qualidade do que o indivíduo Z escolhido para ser substituído. Uma alternativa a esta abordagem é efectuar a substituição incondicional.

As regras genéricas enunciadas em cima permitem adoptar uma grande variedade de estratégias. Uma das mais conhecidas é a *replace worst*, proposta por Whitley no seu sistema GENITOR (Whitley, 1989) que advoga que a solução Z é a de pior qualidade na população e a que a condição de substituição é baseada no mérito, ou seja, o descendente entra população desde que tenha melhor qualidade do que o pior indivíduo que lá se encontra. Embora seja simples e intuitiva, esta estratégia provoca uma rápida diminuição de diversidade, levando o algoritmo a convergir prematuramente. Por este motivo é normalmente preferível adoptar um mecanismo que, para além da qualidade, considere medidas de semelhança entre indivíduos, de modo a manter níveis de diversidade adequados dentro da população.

Algoritmo 5 Algoritmo Genético em Regime Permanente (Steady-State)

1: Seleccionar dois progenitores P1 e P2 da população actual
2: Criar um descendente D através da aplicação de operadores genéticos
3: Avaliar o desdendente D
4: Seleccionar uma solução Z na população para substituição
5: Decidir se o descendente D substitui a solução Z

6. Aplicações Práticas

Os AGs são actualmente um método de resolução de problemas muito utilizado, tendo demonstrado a sua eficácia nos mais diferentes domínios, tanto científicos como em situações do mundo real. A seguir apresentamos alguns exemplos de áreas nas quais os AGs têm sido aplicados.

- **Optimização numérica**: Esta categoria inclui muitos problemas de engenharia que podem ser modelados através de um conjunto de parâmetros reais para os quais é necessário encontrar a melhor configuração. O que caracteriza esta classe é o facto de os domínios associados a cada parâmetro serem contínuos. Dois exemplos de áreas nas quais podem surgir este tipo de problemas são o controlo (Fleming e Purshouse, 2002) ou a química teórica e computacional (Johnston, 2004).

[21] Existem diversas medidas que procuram aferir a semelhança entre duas soluções para um dado problema. São normalmente métricas específicas de uma dada representação e/ou problema e procuram determinar até que ponto duas soluções partilham características idênticas.

- **Optimização combinatória**: Os problemas desta classe são, provavelmente, as situações de optimização às quais os AGs são aplicados com maior frequência. O que os distingue dos problemas do tópico anterior é que, neste caso, as variáveis de decisão são discretas. Existem inúmeras situações do mundo real que podem ser aproximadas por problemas de optimização combinatória, o que justifica a atenção da comunidade científica. Alguns dos exemplos mais relevantes são o problema do caixeiro viajante, o problema do encaminhamento de veículos ou problemas de escalonamento (Jensen, 2004; Lewis, 2008; Pereira e Tavares, 2009; Tsai et al., 2004).

- **Robótica evolutiva**: Nesta área recorre-se a algoritmos de inspiração biológica para evoluir a morfologia e comportamentos de robots (Floreano e Nolfi, 2004).

- **Simulação**: A capacidade de proceder à evolução/adaptação de um conjunto de entidades permite que os AGs sejam usados no desenvolvimento e estudo de modelos de simulação pertencentes a diversos domínios. Alguns exemplos relevantes são a economia (estudo da emergência de mercados económicos), o sistema imunitário (estudo das mutações somáticas), a ecologia (estudo de fenómenos do tipo hospedeiro/parasita) ou os sistemas sociais (estudo da evolução do comportamento social em colónias de insectos) (Mitchell, 1996).

A lista apresentada não pretende, de modo nenhum, ser exaustiva. A consulta de actas das conferências mais relevantes nesta área, a *Genetic and Evolutionary Computation Conference (GECCO)* [22] ou o *IEEE Congress on Evolutionary Computation (CEC)*[23] entre outras, pode ser útil para se obter uma ideia geral dos desenvolvimentos recentes e para ajudar a perceber quais as áreas de aplicação preferenciais para os AGs.

Programação Genética

Uma das aplicações mais interessantes de algoritmos de inspiração biológica é a possibilidade de evoluir programas de computador que resolvam um determinado problema. A **programação genética** pode ser considerada uma descendente directa dos AGs. Embora tenham existido algumas tentativas anteriores para desenvolver um sistema com estas características, o nascimento da área é normalmente associado à publicação, em 1992, do livro *Genetic Programming* de Koza (1992). A questão central para a qual este trabalho procura obter respostas é a seguinte: 'De que forma pode um computador aprender a resolver um determinado problema sem ser explicitamente programado para isso?"

Um algoritmo de programação genética evolui um conjunto de programas de computador ao longo de várias gerações. O objectivo é encontrar uma solução (i.e., um programa) que resolva um determinado problema. Tal como num AG, é necessário definir uma representação para as soluções, uma função de aptidão, um mecanismo de seleçcão e operadores genéticos. As abordagens iniciais recorreram a árvores para representar programas de computador. Actualmente existem diversas representações alternativas como, por exemplo, as lineares ou baseadas em grafos. O livro *A Field Guide to Genetic Programming* de Poli et al. (2008b) apresenta uma introdução clara e completa da área da programação genética, incluíndo uma descrição de aplicações práticas. Por sua vez, na página *Human Competitive Results*[24] é possível consultar um conjunto soluções obtidas por programação genética que, de acordo com um conjunto de critérios, são comparáveis ou melhores do que soluções propostas por humanos.

[22] http://www.sigevo.org/
[23] http://www.ieee-cis.org/
[24] http://www.genetic-programming.com/humancompetitive.html

7. Conclusões

Os AGs são métodos de resolução de problemas inspirados em modelos simplificados da evolução natural. O seu modo de actuação é extremamente simples de enunciar: um conjunto inicial de possíveis soluções para um determinado problema (a população inicial) vai sendo sucessivamente modificado através da aplicação de operadores inspirados nos mecanismos da selecção natural e da variação genética (i.e., o conjunto de soluções vai evoluindo ao longo de sucessivas gerações). Neste capítulo descrevemos as componentes básicas que constituem um AG, nomeadamente o método de selecção de progenitores e os operadores de transformação que dão origem a novas soluções. Identificámos ainda a definição da representação e da função de aptidão como duas decisões cruciais para o sucesso da optimização. Para tornar a exposição o mais clara possível, descrevemos, passo a passo, a aplicação de um AG a uma instância do KSP.

Uma das características que torna os AGs especialmente apelativos é a simplicidade do seu funcionamento. Considerando a metáfora biológica como ponto de partida, é trivial entender e implementar as etapas associadas ao processamento iterativo da população de soluções. Esta simplicidade está relacionada com o facto de um AG simples ser um método de optimização global. A aplicação a um determinado problema não exige conhecimento particular sobre a situação em causa, sendo suficiente definir um critério de qualidade que avalie o mérito relativo das soluções que são geradas. Esta característica torna os AGs especialmente robustos e com capacidade para lidar com espaços de procura de grande dimensão, irregulares, descontínuos e multimodais. É importante, no entanto, referir que estas características não inviabilizam que um AG possa ser combinado com um algoritmo que possua propriedades complementares. Pelo contrário, abordagens híbridas que combinem as características globais de um AG com propriedades mais locais e/ou especializadas de outros métodos podem ser extremamente eficazes na resolução de problemas difíceis. Este tópico será abordado num dos capítulos deste manual.

(Página deixada propositadamente em branco)

CAPÍTULO 3

Estratégias Evolutivas

Lino Costa [*] *Pedro Oliveira* [**]

[*] *Departamento de Produção e Sistemas*
Universidade do Minho

[**] *Instituto de Ciências Biomédicas Abel Salazar*
Universidade do Porto

Na natureza, de acordo com a Teoria da Evolução de Darwin, a evolução dos seres vivos ocorre devido à selecção natural e à adaptação ao ambiente. A selecção natural faz com que os seres vivos mais aptos, em relação ao ambiente, tenham maior probabilidade de sobrevivência. Por outro lado, para que seja possível a adaptação dos seres vivos a um ambiente continuamente em mudança, surgiram mecanismos na natureza que tornam possível a diversidade, isto é, a existência de seres vivos com características próprias que, potencialmente, os podem tornar mais aptos. A contínua evolução das espécies pode ser vista como um processo de optimização consistindo na adaptação dos seres vivos ao seu meio ambiente. Este modelo biológico inspirou o desenvolvimento de diversos Algoritmos Evolucionários[1], na década de 60 do século XX, nomeadamente, na Alemanha, as Estratégias Evolutivas (Rechenberg, 1973; Schwefel, 1995) e, nos Estados Unidos da América, os Algoritmos Genéticos (Holland, 1975;

[1] O termo Algoritmo Evolucionário é utilizado, genericamente, na descrição de sistemas computacionais para a resolução de problemas que utilizam como elementos chave na sua implementação, modelos computacionais baseados em mecanismos de evolução. De referir que estes mecanismos de evolução correspondem ou relacionam-se com processos evolutivos biológicos.

Goldberg, 1989) para serem aplicados a problemas de optimização[2].

As Estratégias Evolutivas (EEs) foram desenvolvidas com o objectivo de serem aplicadas em optimização numérica, revelando-se como algoritmos de optimização robustos e eficientes. As EEs são algoritmos iterativos que pesquisam com base em populações de indivíduos, que representam potenciais soluções para o problema de optimização. Cada iteração é, por analogia com os sistemas biológicos, chamada de *geração*. Em cada geração, os operadores genéticos (*recombinação* e *mutação*) actuam sobre a população de indivíduos permitindo a exploração do espaço de procura e controlar a diversidade dos indivíduos da população. O princípio de sobrevivência do mais apto é implementado pela *selecção* que garante a sobrevivência dos indivíduos que representam soluções de maior qualidade para o problema. A condições que se devem verificar para terminar o processo iterativo (e.g., a obtenção de uma solução de qualidade suficiente para a resolução do problema) constituem os chamados critérios de paragem. Um boa introdução às EEs, focando, com alguma profundidade, os fundamentos teóricos, é feita por Beyer e Schwefel (2002).

1. Optimização em Espaços Contínuos

Um grande número de aplicações de EEs a problemas de optimização com espaços de procura discretos e contínuos encontram-se descritas na literatura (Rechenberg, 1964, 1994; Schwefel, 1995; Bäck, 1996). No entanto, as EEs serão aqui apresentadas como algoritmos para a resolução de problemas de programação não linear em espaços de procura contínuos (Nocedal e Wright, 1999):

$$\min f(\mathbf{x}) \text{ onde } \mathbf{x} \in \Omega \tag{3.1}$$

sujeito a

$$g_j(\mathbf{x}) \geq 0 \text{ com } j = 1, \ldots, m$$

$$h_i(\mathbf{x}) = 0 \text{ com } i = m+1, \ldots, m+p$$

onde $\mathbf{x}$ é o vector das *variáveis de decisão*; $f(\mathbf{x})$ é a *função objectivo*, i.e., a função que se pretende minimizar[3]; $\mathbf{g}(\mathbf{x})$ é o vector das *restrições do tipo desigualdade* que devem ser satisfeitas; $\mathbf{h}(\mathbf{x})$ é o vector das *restrições do tipo igualdade* que devem ser satisfeitas.

Na formulação apresentada existem n variáveis reais, m restrições do tipo desigualdade e p restrições do tipo igualdade (o número total de restrições é $m+p$). O *espaço das variáveis*, $\Omega \subseteq \Re^n$, corresponde ao conjunto de todos os valores possíveis para as variáveis de decisão. A procura da solução de um problema de optimização, o ponto *óptimo* $\mathbf{x}^*$, é feita no espaço das variáveis[4]. As restrições de desigualdade são expressas em termos de desigualdades do tipo maior ou igual ($\geq$)[5]. Muitas vezes, poderão existir restrições de desigualdade especificando limites inferiores ($\underline{x_k}$) e superiores ($\overline{x_k}$) para as n variáveis x_k, i.e., $\underline{x_k} \leq x_k \leq \overline{x_k}$, para $k = 1, \ldots, n$.

Qualquer ponto $\mathbf{x} \in \Omega$ diz-se satisfazer uma restrição se, para essa restrição, o lado esquerdo da expressão calculado nesse ponto está de acordo com o lado direito, em termos do operador relacional.

[2] De notar que estas duas abordagens não são as únicas inspiradas na natureza para resolver problemas de optimização, existindo outras, tais como, a Programação Evolucionária (Fogel et al., 1966), a Programação Genética (Koza, 1992), os algoritmos de Colónia de Formigas (Dorigo et al., 1996) e os algoritmos de Enxames de Partículas (Kennedy e Eberhart, 1995).

[3] De notar que qualquer problema formulado em termos de maximização de uma função objectivo $f(\mathbf{x})$ pode ser reformulado da seguinte forma: $\max f(\mathbf{x}) = -\min(-f(\mathbf{x}))$.

[4] Em geral, os problemas de optimização são abordados pressupondo-se que o ponto óptimo $\mathbf{x}^*$ existe, é único, e pode ser localizado utilizando um algoritmo de optimização. Apesar de muitas vezes este ser o caso, existem situações em que tal não se verifica: se $f(\mathbf{x})$ não é limitada inferiormente, então $\mathbf{x}^*$ não existe ou, para certas funções objectivo, $\mathbf{x}^*$ poderá não ser único.

[5] Qualquer restrição expressa em termos da desigualdade menor ou igual pode ser facilmente transformada em termos da desigualdade maior ou igual bastando para isso multiplicá-la por –1.

Um ponto é dito *ponto admissível* se todas as restrições do tipo desigualdade e restrições do tipo igualdade forem satisfeitas nesse ponto. O conjunto de todos os pontos admissíveis constitui a *região admissível* e pode ser definido da seguinte forma:

$$A = \{\mathbf{x} \in \Omega : \mathbf{g}(\mathbf{x}) \geq 0 \wedge \mathbf{h}(\mathbf{x}) = 0\}\,.$$

Todos os pontos que não pertencem ao conjunto A são pontos não admissíveis. O óptimo $\mathbf{x}^*$, a solução do problema, pertence necessariamente à região admissível[6].

2. Características Gerais

Existem diferenças importantes relativamente à forma como as EEs e os algoritmos de optimização tradicionais (não evolucionários) resolvem os problemas de optimização contínua formulados em (3.1). Tal como as EEs, os algoritmos de optimização tradicionais são iterativos. Em geral, iniciam a procura a partir de uma aproximação inicial ao óptimo que se pretende encontrar. A partir da aproximação inicial são geradas, sucessivamente, novas estimativas do óptimo. O processo iterativo é repetido até que os critérios de paragem sejam verificados. Os diversos algoritmos de optimização distinguem-se pela forma como são calculadas as novas estimativas ao longo da procura. Quase todas as abordagens utilizam os valores da função objectivo, das restrições e, muitas vezes, das primeiras e segundas derivadas destas funções. Alguns algoritmos, para calcular uma nova aproximação ao óptimo, utilizam apenas a informação relativa à aproximação actual, enquanto que outros consideram, também, a informação recolhida durante as iterações passadas. Os algoritmos de optimização tradicionais podem ser caracterizados, em grande parte, pela descrição anterior. No entanto, convém separar os algoritmos tradicionais em dois grupos principais: os métodos directos e os métodos baseados em gradientes. Os métodos directos guiam a procura com base em informação relacionada com a função objectivo e/ou restrições. Enquanto que os métodos baseados em gradientes, para além desta informação, necessitam, também, das primeiras e/ou segundas derivadas. Por este motivo, em geral, os métodos baseados em gradientes não são eficientes quando, num problema, não se verificam as condições de diferenciabilidade e/ou continuidade das funções. De salientar que, nestes métodos, os resultados obtidos dependem grandemente das aproximações ao óptimo consideradas no início do processo iterativo e não são eficientes na resolução de problemas com espaços de procura de natureza discreta.

Em contraste com estas abordagens ditas tradicionais, as EEs:

- iniciam a procura a partir de uma população de potenciais soluções geradas de forma aleatória (caso seja conhecida podem partir de uma aproximação inicial);
- trabalham, ao longo das gerações, com populações de potenciais soluções, em vez de uma única aproximação ao óptimo por iteração;
- não utilizam nenhuma informação relativa às primeiras e/ou segundas derivadas da função objectivo e/ou restrições;
- não exigem nenhuma condição relativa à continuidade e convexidade do espaço de procura;
- podem utilizar mecanismos que permitem encontrar múltiplos óptimos locais (se existirem) numa única execução;

[6] Quando um ponto satisfaz uma restrição j do tipo desigualdade, duas situações podem ocorrer: o ponto está no limite da região admissível da restrição, i.e., $g_j(\mathbf{x}) = 0$ e neste caso a restrição j diz-se *activa*; ou o ponto está no interior da região admissível da restrição, i.e., $g_j(\mathbf{x}) > 0$ e neste caso a restrição j diz-se *inactiva*. Para qualquer ponto admissível $\mathbf{x}$, todas as p restrições do tipo igualdade estão activas.

- guiam a procura com base em mecanismos e/ou regras probabilísticas (estocásticas).

Estas características das EEs fazem com que sejam particularmente eficientes na resolução de problemas reais, onde os espaços de procura podem não ser convexos e, por isso, muitas vezes contêm um grande número de óptimos locais e/ou mais do que uma solução óptima. Por outro lado, muitos problemas reais apresentam espaços de procura de natureza discreta onde não é possível garantir as condições de diferenciabilidade e continuidade desejáveis para muitos dos algoritmos tradicionais.

3. Nomenclatura

As EEs foram desenvolvidas, inicialmente, por Rechenberg (1973), com o objectivo de resolver problemas de optimização, tendo como base as estruturas e os processos de optimização que ocorrem na natureza. Mais tarde, Schwefel (1981) desenvolveu novos esquemas evolutivos com base nos mesmos princípios.

Nas EEs é utilizada uma população com um determinado número de indivíduos. Em cada geração, a partir dos indivíduos presentes na população, os *progenitores*, são gerados novos indivíduos, os *descendentes*. Desta forma, ao longo das gerações, novas populações são sucessivamente geradas. As EEs trabalham directamente com a representação real das variáveis de decisão, pelo que cada indivíduo é um vector de números reais representando uma potencial solução para o problema de optimização.

Desde o seu aparecimento, tem vindo a ser utilizada uma nomenclatura própria para designar as diferentes EEs. Esta nomenclatura é baseada no número de progenitores, no número de descendentes e no tipo de selecção considerado. O número de progenitores é designado por μ e o número de descendentes por λ. Por outro lado, dois tipos de selecção foram originalmente descritos e designados por selecção '+' e selecção ','. A primeira e mais simples EE, desenvolvida por Rechenberg (1973), onde a selecção era feita sobre uma população de dois membros, i.e., $\mu + \lambda = 1 + 1$, é designada, na nomenclatura atrás apresentada, por EE-$(1 + 1)$. Posteriormente, o mesmo autor desenvolveu uma estratégia multimembros mais complexa, onde a selecção era feita sobre uma população de $\mu > 1$ indivíduos e um descendente; i.e., $\mu + \lambda = \mu + 1$, que é designada por EE-$(\mu + 1)$.

De uma forma mais genérica, numa EE-$(\mu + \lambda)$, numa determinada geração, existe uma população de μ progenitores que gera λ descendentes por mutação (Figura 3.1). Em seguida, os $\mu + \lambda$ indivíduos são avaliados e ordenados de acordo com os seus valores da função objectivo[7]. Finalmente, o processo de selecção faz com que apenas os μ melhores de todos os $\mu + \lambda$ indivíduos se tornem os progenitores da geração seguinte, i.e., a selecção é feita a partir dos $\mu + \lambda$ indivíduos. De referir que este mecanismo de selecção é determinístico, uma vez que apenas os melhores indivíduos são seleccionados para formarem a população de progenitores da geração seguinte.

Um outro modelo conceptual pode ser definido onde, numa determinada geração, uma população de μ progenitores gera, por mutação, λ descendentes (assumindo que $\lambda > \mu$). Em seguida, os λ descendentes são avaliados e ordenados de acordo com os seus valores da função objectivo. Então, os μ melhores dos λ descendentes gerados tornam-se os progenitores da próxima geração, i.e., os μ progenitores não são incluídos no processo de selecção. Esta EE que utiliza a selecção ',' é designada por EE-(μ, λ) (Figura 3.2). Os dois tipos de EEs atrás descritas, a EE-$(\mu + \lambda)$ e a EE-(μ, λ), diferem basicamente no procedimento de selecção.

Originalmente, as EEs baseavam-se num único operador, a mutação, para gerar novos indivíduos. Posteriormente, foi introduzido um outro operador, a recombinação, que era aplicado juntamente com a mutação (Schwefel, 1995). Mais adiante, nesta mesma secção, descrever-se-á detalhadamente este operador.

[7] Assumindo que os indivíduos representam pontos admissíveis. Mais tarde, serão abordados diversos mecanismos de tratamento de restrições.

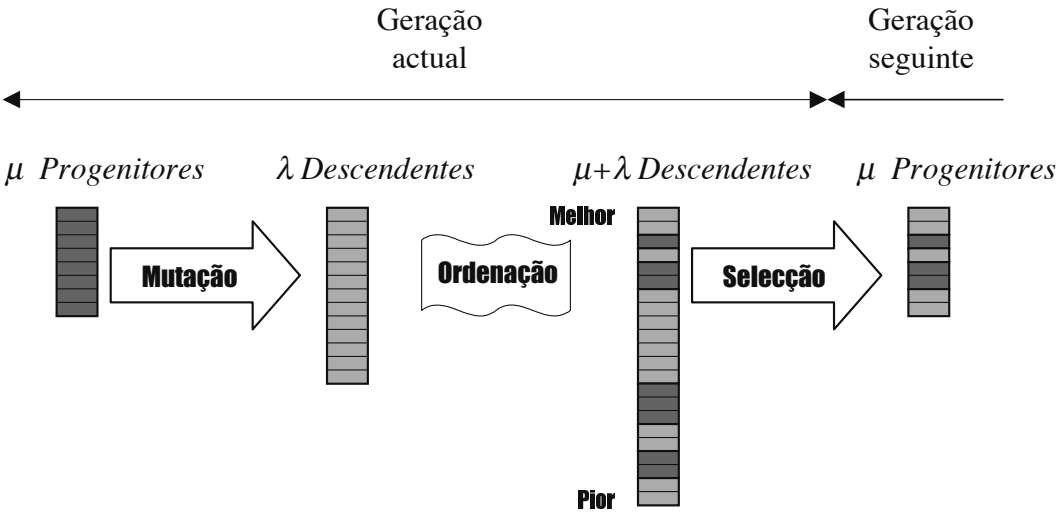


Figura 3.1: Aspecto Geral da Estratégia Evolutiva $(\mu + \lambda)$

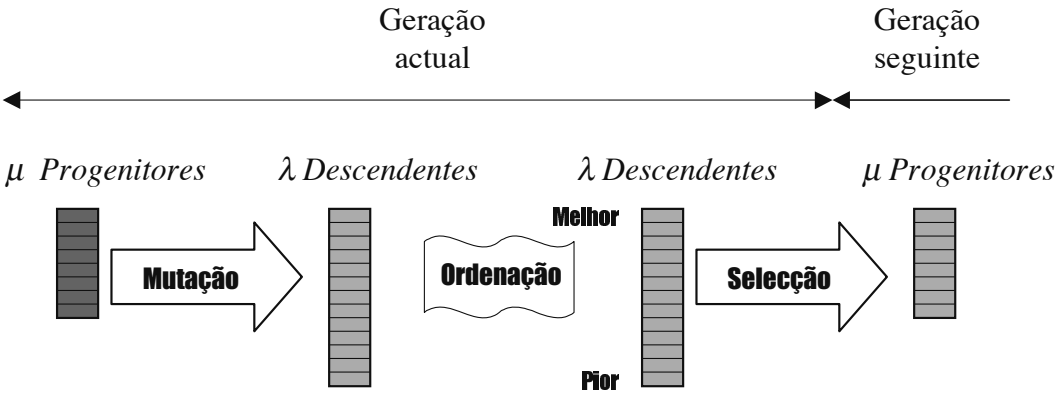


Figura 3.2: Aspecto Geral da Estratégia Evolutiva (μ, λ)

Em seguida, cada uma das EEs atrás referidas irá ser descrita com maior detalhe. Todos os algoritmos apresentados são formulados para problemas de minimização sem restrições. Mais adiante, ir-se-á focar diversos esquemas de tratamento de restrições quer do tipo desigualdade quer do tipo igualdade.

4. Estratégia Evolutiva (1+1)

Este esquema evolutivo com apenas 2 membros, designado por EE-$(1+1)$, foi proposto por (Rechenberg, 1964) para a resolução de problemas de optimização. Numa determinada geração, existe apenas um progenitor ($\mu = 1$) e um único descendente ($\lambda = 1$), e a selecção tem lugar apenas entre estes dois membros.

O funcionamento da EE-$(1+1)$ pode ser descrito pelo Algoritmo 1. A procura inicia-se a partir de um ponto inicial $\mathbf{x}_0$ (uma aproximação ao óptimo). Em seguida, um novo ponto $\mathbf{x}_N$ é gerado por mutação através da adição de uma quantidade aleatória normal com média 0 e variância σ^2. O desvio padrão σ está relacionado com o tamanho do passo e vai sendo adaptado ao longo da procura. Depois, os dois pontos são comparados e o melhor (com menor valor da função objectivo e satisfazendo todas as restrições) é seleccionado para progenitor da próxima geração. Este processo é repetido até que o critério de paragem seja verificado.

De seguida, vai-se analisar com mais detalhe cada um dos passos envolvidos neste algoritmo.

Algoritmo 1 Estratégia Evolutiva $(1+1)$

Require: $\mathbf{x}_0$, Δx
$t \leftarrow 0$
$\mathbf{x}^{(t)} \leftarrow \mathbf{x}_0$ [onde $\mathbf{x}^{(t)} = (x_1, x_2, \ldots, x_n)^{(t)}$]
$\sigma \leftarrow \frac{|\Delta x|}{\sqrt{n}}$
para $i = 1$ to $10n$ **faça**
  $sucessos[i] \leftarrow 0$
fim para
enquanto CP falso **faça** [// é o resto da divisão inteira]
  se $(t//n = 0 \wedge t \geq 10n)$ **então** [Regra de 1/5 sucessos]
    $P_s \leftarrow \frac{\sum_{i=1}^{10n} sucessos[i]}{10n}$
    $\sigma \leftarrow \begin{cases} \sigma/c & \text{se} \quad P_s > 1/5 \\ c\sigma & \text{se} \quad P_s < 1/5 \quad \text{com} \quad c = 0.85 \\ \sigma & \text{se} \quad P_s = 1/5 \end{cases}$
  fim se
  $\mathbf{x}_N \leftarrow \mathbf{x}^{(t)} + \mathbf{N}(0, \sigma^2)$ [Mutação]
  se $(f(\mathbf{x}_N) < f(\mathbf{x}^{(t)}))$ **então** [Selecção]
    $\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}_N$
    $sucessos[(t+1)//10n] \leftarrow 1$
  senão
    $\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)}$
    $sucessos[(t+1)//10n] \leftarrow 0$
  fim se
  $t \leftarrow t + 1$
fim enquanto
$\mathbf{x}_{final} \leftarrow \mathbf{x}^{(t)}$
retorno $\mathbf{x}_{final}$

Aproximação Inicial

Como já foi dito, para iniciar a procura é necessária uma aproximação inicial ao óptimo $\mathbf{x}_0$. Mas, para além disso, é necessário escolher os valores iniciais para o desvio padrão σ. O valor inicial típico para o desvio padrão σ pode ser expresso pela seguinte equação:

$$\sigma = \frac{\Delta x}{\sqrt{n}},$$

onde Δx é uma medida aproximada da distância esperada da aproximação inicial $\mathbf{x}_0$ ao óptimo e n é a dimensão do problema (o número de variáveis de decisão).

Ao longo da procura, devem-se garantir as seguintes duas condições para os valores dos tamanhos do passo σ:

1. $\sigma > 0$ para $i = 1, \ldots, n$;
2. os valores de σ devem ser suficientemente grandes para que pelo menos o dígito menos significativo da variável x_i seja alterado.

As condições anteriores podem ser escritas em termos dos seguintes limites inferiores para os tamanhos do passo: $\sigma \geq \varepsilon_1$ e $\sigma \geq \varepsilon_2 |x_i|$ para $i = 1, \ldots, n$, onde ε_1 e ε_2 dependem da precisão do computador utilizado, sendo $\varepsilon_1 > 0$ e $\varepsilon_2 > 0$.

Mutação

O operador de mutação consiste em gerar um novo ponto $\mathbf{x_N}$ através da adição de uma quantidade aleatória $\mathbf{z}$. A quantidade aleatória $\mathbf{z}$ traduz o tamanho do passo (ou comprimento do deslocamento), e a sua escolha deve ser feita de tal forma que pequenos deslocamentos ocorram frequentemente e grandes deslocamentos ocorram raramente. Com este propósito, em geral, as quantidades aleatórias $\mathbf{z}$ são geradas de acordo com uma distribuição Normal. Para além disso, as seguintes condições devem ser impostas à distribuição dos tamanhos do passo:

1. O valor esperado dos componentes z_i de $\mathbf{z}$, com $i = 1, \ldots, n$, deve ser nulo, i.e., $E[z_i] = 0$;
2. As variâncias σ^2, com $i = 1, \ldots, n$, devem ter valores pequenos.

Logo, os componentes aleatórios z_i podem ser calculados de acordo com uma distribuição Normal com média nula e variância σ^2, i.e., $z_i \sim N(0, \sigma^2)$. Para determinar números aleatórios normais a partir de número aleatórios uniformes, podem ser utilizadas as regras de transformação de Box e Muller (1958)[8].

Controlo do Tamanho do Passo

Para que o processo de optimização seja eficiente, os deslocamentos devem ser continuamente modificados. Se os deslocamentos forem demasiado pequenos, o número de iterações do processo de procura é desnecessariamente grande; pelo contrário, se forem demasiado grandes, poderá não se obter uma boa aproximação ao óptimo, ou mesmo, o processo poderá não convergir. Por isso, em todas as estratégias de optimização, o controlo do tamanho do passo é uma das componentes mais importantes no processo de procura.

[8] De acordo com estas regras, dois números aleatórios independentes normalmente distribuídos com média nula e variância unitária podem ser calculados partir de quaisquer dois números aleatórios gerados uniformemente no intervalo $[0, 1]$ da seguinte forma:$x_1 = \sqrt{-2\ln(y_1)}\sin(2\pi y_2)$ e $x_2 = \sqrt{-2\ln(y_1)}\cos(2\pi y_2)$ onde y_1 e y_2 são os dois números aleatórios gerados uniformemente no intervalo $[0, 1]$ e, x_1 e x_2 são os dois números aleatórios normalmente distribuídos com média nula e variância unitária. Para se obter dois números aleatórios z_1 e z_2 normalmente distribuídos com média nula e variância σ^2, basta multiplicar x_1 e x_2 pelo desvio padrão σ, i.e., $z_1 = \sigma x_1$ e $z_2 = \sigma x_2$.

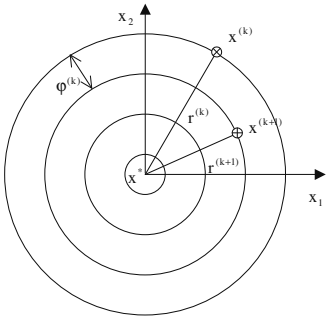


Figura 3.3: Razão de Progresso

Nas EEs, a grandeza das quantidades aleatórias z_i depende do desvio padrão σ. Quanto maior for o desvio padrão σ, maiores serão as quantidades aleatórias z_i e, consequentemente, os tamanhos do passo. Rechenberg (1973), baseando-se na aplicação da EE-$(1+1)$ a alguns tipos de problemas de optimização, formulou a chamada "Regra de 1/5 de Sucessos" para controlar os tamanhos do passo. Esta regra pode ser descrita da seguinte forma:

> "De tempos a tempos, ao longo do processo de procura, calcule-se a frequência de sucessos, i.e., a razão entre o número de sucessos e o número de iterações. Se a razão for maior que 1/5, aumente-se a variância, se for menor que 1/5, diminua-se a variância."

Assumindo-se que esta regra é aplicada periodicamente todas as Δ_t iterações, a sua expressão pode ser definida da seguinte forma para uma iteração k:

$$\sigma^{(k+1)} = \begin{cases} c_{inc}\,\sigma^{(k)} & \text{se} \quad P_s(\Delta_t) > 1/5 \\ c_{dec}\,\sigma^{(k)} & \text{se} \quad P_s(\Delta_t) < 1/5 \\ \sigma^{(k)} & \text{se} \quad P_s(\Delta_t) = 1/5 \end{cases},$$

onde $P_s(\Delta_t)$ é a proporção de sucessos nas últimas Δ_t iterações e, $c_{dec} < 1$ e $c_{inc} > 1$ são, respectivamente, os coeficientes de diminuição e de aumento do desvio padrão σ.

Defina-se a razão de progresso como o valor esperado da diferença radial após uma iteração k (Figura 3.3):

$$\varphi^{(k)} = E\left[\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\| - \left\|\mathbf{x}^* - \mathbf{x}^{(k+1)}\right\|\right] = E\left[r^{(k)} - r^{(k+1)}\right],$$

onde $\mathbf{x}^*$ representa o óptimo e, $\mathbf{x}^{(k)}$ e $r^{(k)}$ representam, respectivamente, a aproximação ao óptimo e a distância ao óptimo na iteração k.

Verifica-se que, para muitos problemas, a regra de 1/5 de sucessos mostra-se extremamente eficiente na manutenção, aproximadamente, da maior razão de progresso possível para o óptimo. No entanto, interessa definir qual a frequência com que o critério de sucesso deve ser testado (o valor do parâmetro Δ_t) e qual o factor de diminuição ou de aumento dos desvios padrão mais eficiente (os valores dos coeficientes c_{dec} e c_{inc}). Para tentar responder a esta questão, podem-se utilizar os resultados obtidos por Rechenberg (1973). A máxima razão de progresso é dada por:

$$\varphi_{\max} = c_1 \frac{r^{(k)}}{n} \text{ sendo } c_1 \cong 0.2025,$$

com variância comum σ^2, cujo valor σ^2_{opt} óptimo é dado por:

$$\sigma^2_{opt} = \left(c_2 \frac{r^{(k)}}{n}\right)^2 \text{ com } c_2 \cong 1.224,$$

para todos os componentes $z_i^{(k)}$ do vector aleatório $\mathbf{z}^{(k)}$. Nestas expressões, $r^{(k)}$ é a distância actual ao óptimo e n é o número de variáveis. Destas equações pode-se obter a relação para as variações nos comprimentos dos deslocamentos após uma geração, supondo a condição de máxima razão de convergência:

$$\frac{\sigma_{opt}^{(k+1)}}{\sigma_{opt}^{(k)}} = \frac{r^{(k+1)}}{r^{(k)}} = \frac{r^{(k)} - \varphi_{\max}}{r^{(k)}} = 1 - \frac{c_1}{n},$$

ou após n gerações:

$$\frac{\sigma_{opt}^{(k+n)}}{\sigma_{opt}^{(k)}} = \left(1 - \frac{c_1}{n}\right)^n.$$

Quando n for muito maior que 1, o factor do comprimento do deslocamento tende para uma constante:

$$\lim_{n\to+\infty} \left(1 - \frac{c_1}{n}\right)^n = e^{-c_1} \cong 0.817 \cong \frac{1}{1.224}.$$

Este resultado aplica-se ao caso limite em que existem muitas variáveis (n grande) e expressa que a razão de progresso é inversamente proporcional ao número de variáveis. O facto da razão de progresso perto do seu máximo ser bastante insensível a pequenas variações das variâncias, juntamente com o facto da probabilidade de sucesso só poder ser determinada pela média de diversas gerações, conduz à seguinte formulação, mais precisa, da regra de 1/5 de sucessos onde n é o número de variáveis de decisão do problema de optimização:

> "Após cada n gerações, verifique-se quantos sucessos ocorreram nas últimas $10n$ gerações. Se este número for inferior a $2n$, multipliquem-se os comprimentos dos deslocamentos pelo factor 0.85; senão, dividam-se os comprimentos dos deslocamentos pelo factor 0.85."

Esta regra permite que os comprimentos dos deslocamentos ou as variâncias dos deslocamentos aleatórios sejam controlados, fixando-se os seguintes valores $c_{dec} = 0.85$, $c_{inc} = 1/0.85$ e $\Delta_t = 10n$. No entanto, a probabilidade de sucesso não fornece nenhuma indicação de quão apropriadas são as razões das variâncias σ_i^2 relativamente umas às outras (para cada variável), pelo que os comprimentos dos deslocamentos só podem ser todos reduzidos em conjunto ou todos aumentados em conjunto. Por este motivo, em geral, considera-se que $\sigma_1^2 = \sigma_2^2 = \ldots = \sigma_n^2 = \sigma^2$.

Critério de Paragem

O critério de paragem determina quando o processo de procura deve ser terminado. Em geral, são utilizadas as condições:

$$\left|f(\mathbf{x}^{(k-\Delta_k)}) - f(\mathbf{x}^{(k)})\right| \leq \varepsilon_3 \text{ ou } \frac{\left|f(\mathbf{x}^{(k-\Delta_k)}) - f(\mathbf{x}^{(k)})\right|}{\left|f(\mathbf{x}^{(k)})\right|} \leq \varepsilon_4,$$

e $\Delta_k \geq 20n$, onde ε_3 e ε_4 dependem da precisão do computador utilizado, sendo $\varepsilon_3 > 0$ e $\varepsilon_4 > 0$. A condição $\Delta_k \geq 20n$ assegura que, no caso extremo, entre testes da regra de 1/5 de sucessos, os comprimentos dos deslocamentos são reduzidos ou aumentados, pelo menos pelo factor $(0.85)^{\pm 20} \cong (25)^{\mp 1}$, o que evita que a procura seja terminada apenas porque os comprimentos dos deslocamentos são forçados a variar muito frequentemente. Por outro lado, é evidente que o critério de convergência não precisa de ser verificado em todas as gerações. O procedimento usual é testá-lo apenas cada $20n$ gerações.

5. Estratégias Evolutivas Multimembros

Nestas EEs multimembros existem, em cada geração, μ progenitores e λ descendentes. A EE-(μ, λ) distingue-se da EE-$(\mu+\lambda)$ apenas na selecção que tem lugar, no primeiro caso, entre os λ descendentes enquanto que, no segundo caso, se faz a partir dos $\mu + \lambda$ indivíduos (Schwefel, 1995).

Caso seja conhecida uma aproximação ao óptimo, a procura pode ser iniciada gerando μ pontos a partir desse ponto inicial. Caso contrário, a população inicial pode ser gerada de forma aleatória. Em cada geração, λ pontos são gerados por mutação, como resultado da adição de números aleatórios normais $\mathbf{z}$. O desvio padrão da distribuição normal utilizada na geração das quantidades aleatórias normais tem um papel crucial numa vez que está relacionado com os deslocamentos ocorridos ao longo da procura. Várias regras podem ser adoptadas com o propósito de adaptar estes desvios padrão. Depois, os λ pontos gerados por mutação são ordenados de acordo com os seus valores da função objectivo. Finalmente, os μ melhores pontos são seleccionados para serem os progenitores na geração seguinte. Este processo é repetido até que o critério de paragem seja satisfeito.

De seguida, são apresentados mais detalhes relativos a cada um dos passos do procedimento atrás descrito. Serão, também, apresentados vários algoritmos correspondentes a diferentes variantes de estratégias evolutivas multimembros.

Aproximação inicial

De forma semelhante à EE-(1+1), a procura pode iniciar-se a partir de uma aproximação ao óptimo $\mathbf{x}_0$, que permitirá gerar os μ pontos da população inicial[9]. Caso, não seja conhecida nenhuma aproximação ao óptimo, a população inicial pode ser gerada de forma aleatória. Para além disso, é necessário escolher os valores iniciais para parâmetros associados ao esquema de auto-adaptação considerado[10]. Caso se conheça uma estimativa da distância ao óptimo, os valores iniciais típicos para o desvio ou os desvios padrão podem ser expressos pela mesma equação indicada para EE-$(1 + 1)$.

Mutação

O operador de mutação consiste em gerar novos pontos pela adição de quantidades aleatórias, tal como acontecia na EE-$(1 + 1)$. Contudo, nas EEs multimembros, cada progenitor produz, em média, λ/μ descendentes, de tal forma que λ descendentes sejam gerados. As mesmas condições acerca da normalidade das quantidades aleatórias $\mathbf{z}$ são consideradas neste tipo de estratégias. No entanto, dependendo do tipo de controlo de passo usado para fazer a auto-adaptação, as quantidades aleatórias normais são calculadas de diferentes formas como é descrito mais adiante.

Recombinação

Schwefel (1995) notou uma aceleração considerável na procura, bem como a facilitação da adaptação dos desvios padrão pela introdução do operador de recombinação nas EEs. Basicamente, a recombinação consiste em, antes da mutação, recombinar um conjunto de progenitores por forma a encontrar uma nova solução. Seja ρ o número de progenitores que participam na recombinação ($1 \leq \rho \leq \mu$). Estes ρ progenitores são escolhidos aleatoriamente de entre os μ indivíduos da população. Quando $\rho = 1$ então não existe recombinação. Logo, a nomenclatura das EEs pode ser estendida, e as EEs com recombinação são, em geral, referidas por EE-$(\mu/\rho + \lambda)$ ou EE-$(\mu/\rho, \lambda)$. Existem dois tipos de recombinação principais:

[9] Pode-se utilizar a aproximação ao óptimo como a média da distribuição de probabilidade para gerar os restantes pontos da população inicial, e.g., a média da distribuição normal.

[10] Estes parâmetros poderão ser o desvio padrão inicial σ comum a todas as variáveis, ou os desvios padrão iniciais σ_i distintos para cada variável ou ainda as rotações de direcções de procura relacionadas com a matriz de covariâncias.

- a Recombinação Intermédia, e
- a Recombinação Discreta.

Estes tipos de recombinação podem ser aplicados quer às variáveis de decisão quer aos desvios padrão σ. Em seguida, estes tipos de recombinação são descritos em termos das variáveis de decisão. A sua aplicação aos desvios padrão σ faz-se de forma similar.

Recombinação Intermédia

Neste tipo de recombinação, os componentes dos descendentes são obtidos através da média dos componentes dos progenitores correspondentes (escolhidos aleatoriamente a partir da população). Logo, considerando ρ progenitores escolhidos aleatoriamente, os descendentes $\mathbf{x}_D$ serão dados por:

$$\mathbf{x}_D = \frac{1}{\rho} \sum_{m=1}^{\rho} \mathbf{x}_m. \tag{3.2}$$

Se $\rho = \mu$, este esquema tende a gerar descendentes próximos do centróide da população.

Recombinação Discreta

Neste tipo de recombinação, cada componente dos descendentes é escolhido aleatoriamente a partir de um dos ρ progenitores. Logo, supondo ρ progenitores escolhidos aleatoriamente, os descendentes $\mathbf{x}_D$ serão dados por:

$$\mathbf{x}_D = (x_{u_1,1}, \ldots, x_{u_n,n}) \text{ com } u_1 \in U(0,\rho), \ldots, u_n \in U(0,\rho), \tag{3.3}$$

onde u_i, para $i = 1, \ldots, n$, são índices gerados aleatoriamente (uniformemente), indicando a partir de que progenitor (dos ρ progenitores) o valor da variável de decisão é copiado para o descendente. Este procedimento permite obter diferentes combinações dos valores das variáveis de decisão a partir das soluções existentes na população.

Controlo do Tamanho do Passo

Os desvios padrão podem ser actualizados de diversas formas. Este processo consiste na adaptação dos parâmetros da estratégia durante a procura (Schwefel, 1995). As implementações mais conhecidas são:

- a Adaptação Isotrópica,
- a Adaptação Não Isotrópica, e
- a Adaptação Não Isotrópica com Rotação

Em seguida, descreve-se sucintamente cada um destes esquemas de actualização dos desvios padrão.

Adaptação Isotrópica

Neste esquema, tal como na EE-$(1+1)$, é considerada uma variância σ^2 comum a todas as variáveis, i.e., $\sigma_1^2 = \sigma_2^2 = \ldots = \sigma_n^2$. O desvio padrão comum σ são agora actualizados pela equação:

$$\sigma^{(k+1)} = \sigma^{(k)} e^z \text{ com } z \sim N(0, \Delta_\sigma^2), \tag{3.4}$$

onde z é determinado de acordo com uma distribuição Normal com média zero e variância Δ_σ^2, onde Δ_σ é um parâmetro do algoritmo.

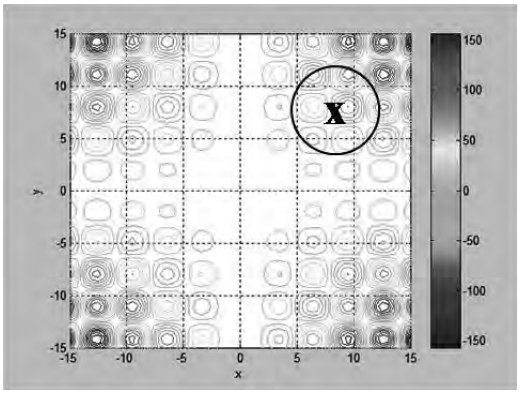


Figura 3.4: Adaptação Isotrópica

A Figura 3.4 ilustra o funcionamento deste esquema de adaptação para um problema com duas variáveis de decisão. Nesta figura, todos os pontos pertencentes à circunferência centrada no ponto x correspondem a pontos com igual probabilidade de serem gerados por mutação.

Em geral, o valor de Δ_σ é fixado em $1/\sqrt{n}$.Uma vez que apenas existe uma única variância comum σ^2, cada indivíduo da população possui a seguinte estrutura:

$$(\mathbf{x};\sigma) = (x_1, x_2, \ldots, x_n; \sigma).$$

Por sua vez, a mutação é dada por:

$$x_i' = x_i + z_i \text{ com } z_i \sim N(0,\sigma). \tag{3.5}$$

Algoritmo 2 Estratégia Evolutiva $(\mu/\rho\overset{+}{,}\lambda)$ com Adaptação Isotrópica

Require: μ, ρ, λ, Δ_σ

1: $t \leftarrow 0$
2: $\mathbf{P}^{(t)} \leftarrow \left\{(\mathbf{x}_m^{(t)}; \sigma^{(t)}; f(\mathbf{x}_m^{(t)})) : m = 1, \ldots, \mu\right\}$ [onde $\mathbf{x}_m^{(t)} = (x_1, x_2, \ldots, x_n)^{(t)}$]
3: **enquanto** CP falso **faça**
4: **para** $l = 1$ to λ **faça**
5: $\mathbf{g}_l \leftarrow agrupar(\mathbf{P}^{(t)}, \rho)$
6: $\sigma_l' \leftarrow recombinar_s(\mathbf{g}_l)$
7: $\mathbf{x}_l' \leftarrow recombinar_x(\mathbf{g}_l)$
8: $\sigma_l'' \leftarrow mutar_s(\sigma_l')$
9: $\mathbf{x}_l'' \leftarrow mutar_x(\mathbf{x}_l', \sigma_l'')$
10: **fim para**
11: $\mathbf{D}^{(t)} \leftarrow \left\{(\mathbf{x}_m^{''(t)}; \sigma_m^{''(t)}; f(\mathbf{x}_m^{''(t)})) : m = 1, \ldots, \mu\right\}$
12: $\mathbf{P}^{(t+1)} \leftarrow \begin{cases} \text{selecção}(\mathbf{D}^{(t)}) & \text{se } EE-(\mu/\rho,\lambda) \\ \text{selecção}(\mathbf{D}^{(t)} \cup \mathbf{P}^{(t)}) & \text{se } EE-(\mu/\rho+\lambda) \end{cases}$
13: $t \leftarrow t+1$
14: **fim enquanto**
15: $\mathbf{x}_{final} \leftarrow melhor(\mathbf{P}^{(t)})$
16: **retorno** $\mathbf{x}_{final}$

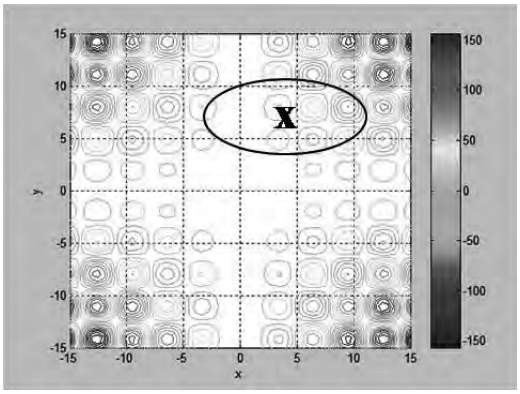


Figura 3.5: Adaptação Não Isotrópica

O Algoritmo 2 é o algoritmo da EE-$(\mu/\rho_{,}^{+}\lambda)$ com adaptação isotrópica. Neste algoritmo, a função *agrupar* selecciona aleatoriamente ρ indivíduos da população de progenitores $\mathbf{P}$. As funções $recombinar_s$ (para os desvios padrão) e $recombinar_x$ (para as variáveis de decisão) recombinam, respectivamente, os ρ indivíduos de acordo com a equação (3.2) ou a equação (3.3). A função $mutar_s$ muta os desvios padrão aplicando a regra isotrópica expressa na equação (3.4). Finalmente, a função $mutar_x$ corresponde à mutação das variáveis de decisão como é descrita na equação (3.5). O algoritmo devolve a solução correspondente ao indivíduo da população final com menor valor da função objectivo (a melhor aproximação ao óptimo).

Adaptação Não Isotrópica

Neste esquema, os desvios padrão σ_i são actualizados (um para cada variável de decisão) de acordo com a equação:

$$\sigma_i^{(k+1)} = \sigma_i^{(k)} e^{z_i} e^{z}, \tag{3.6}$$

onde $z_i \sim N(0, \Delta_\sigma^2)$, $\mathbf{z} \sim N(0, \Delta_{\sigma'}^2)$ e, Δ_σ e $\Delta_{\sigma'}$ são parâmetros do algoritmo. Em geral, os valores de Δ_σ e $\Delta_{\sigma'}$ são fixados, respectivamente, em $1/\sqrt{2\sqrt{n}}$ e $1/\sqrt{2n}$. Este esquema não isotrópico, ao contrário do anterior, permite adaptar os desvios padrão independentemente para cada variável de decisão.

A Figura 3.5 ilustra o funcionamento deste esquema de adaptação para um problema com duas variáveis de decisão. Neste figura, os pontos pertencentes à elipse são os descendentes com igual probabilidade de serem gerados a partir de um ponto x.

Neste esquema existe agora uma variância associada a cada variável σ_i^2, pelo que cada indivíduo da população passa a possuir a seguinte estrutura:

$$(\mathbf{x}; \sigma) = (x_1, \ldots, x_n; \sigma_1, \ldots, \sigma_n).$$

Para fazer a mutação, aplica-se a seguinte equação:

$$x_i' = x_i + z_i \text{ com } z_i \sim N(0, \sigma_i). \tag{3.7}$$

O Algoritmo 3 é o algoritmo da EE-$(\mu/\rho_{,}^{+}\lambda)$ com adaptação não isotrópica. Tal como anteriormente, neste algoritmo, as funções *agrupar*, $recombinar_s$ e $recombinar_x$ permitem seleccionar aleatoriamente ρ indivíduos da população de progenitores $\mathbf{P}$ e recombiná-los de acordo com a equação

Algoritmo 3 Estratégia Evolutiva $(\mu/\rho\overset{+}{,}\lambda)$ com Adaptação Não Isotrópica

Require: μ, ρ, λ, Δ_σ, $\Delta_{\sigma'}$

$t \leftarrow 0$

$\mathbf{P}^{(t)} \leftarrow \left\{(\mathbf{x}_m^{(t)}; \mathbf{s}_m^{(t)}; f(\mathbf{x}_m^{(t)}) : m = 1, \ldots, \mu\right\}$ [onde $\mathbf{x}_m^{(t)} = (x_1, x_2, \ldots, x_n)$ e $\mathbf{s}_m^{(t)} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$]

enquanto CP falso **faça**

  para $l = 1$ to λ **faça**

    $\mathbf{g}_l \leftarrow agrupar(\mathbf{P}^{(t)}, \rho)$

    $\mathbf{s}'_l \leftarrow recombinar_s(\mathbf{g}_l)$

    $\mathbf{x}'_l \leftarrow recombinar_x(\mathbf{g}_l)$

    $\mathbf{s}''_l \leftarrow mutar_s(\mathbf{s}'_l)$

    $\mathbf{x}''_l \leftarrow mutar_x(\mathbf{x}'_l, \sigma''_l)$

  fim para

  $\mathbf{D}^{(t)} \leftarrow \left\{(\mathbf{x}_m^{''(t)}; \mathbf{s}_m^{''(t)}; f(\mathbf{x}_m^{''(t)})) : m = 1, \ldots, \mu\right\}$

  $\mathbf{P}^{(t+1)} \leftarrow \begin{cases} \text{selecção}(\mathbf{D}^{(t)}) & \text{se} \quad EE - (\mu/\rho, \lambda) \\ \text{selecção}(\mathbf{D}^{(t)} \cup \mathbf{P}^{(t)}) & \text{se} \quad EE - (\mu/\rho + \lambda) \end{cases}$

  $t \leftarrow t + 1$

fim enquanto

$\mathbf{x}_{final} \leftarrow melhor(\mathbf{P}^{(t)})$

retorno $\mathbf{x}_{final}$

(3.2) ou a equação (3.3). Contudo, a função $mutar_s$ muta os desvios padrão aplicando agora a regra não isotrópica expressa na equação (3.6). Finalmente, a função $mutar_x$ corresponde à mutação das variáveis de decisão tal como é descrita na equação (3.7) de forma independente para variável de decisão.

Adaptação Não Isotrópica com Rotação

Neste esquema, para modelar a possível correlação entre as variáveis de decisão são consideradas $n(n-1)/2$ rotações. Os desvios padrão σ_i e as rotações α_j são actualizadas de acordo com a equação:

$$\begin{aligned} \sigma_i^{(k+1)} &= \sigma_i^{(k)} e^{z_i} e^{z} \\ \alpha_j^{(k+1)} &= \alpha_j^{(k)} + \beta N(0,1), \end{aligned} \tag{3.8}$$

onde $z_i \sim N(0, \Delta_\sigma^2)$, $\mathbf{z} \sim N(0, \Delta_{\sigma'}^2)$ e, Δ_σ e $\Delta_{\sigma'}$, e β são parâmetros do algoritmo. Em geral, os valores de Δ_σ e $\Delta_{\sigma'}$ são fixados, respectivamente, em $1/\sqrt{2\sqrt{n}}$, $1/\sqrt{2n}$ e $\beta = 5$. Este esquema não isotrópico com rotações, ao contrário do anterior, permite adaptar os desvios padrão tendo em conta a possível existência de correlações entre as variáveis de decisão. Neste esquema existe agora uma variância associada a cada variável σ_i^2 e rotações α_j, pelo que cada indivíduo da população passa a possuir a seguinte estrutura:

$$(\mathbf{x}; \sigma; \alpha) = (x_1, \ldots, x_n; \sigma_1, \ldots, \sigma_n; \alpha_1, \ldots, \alpha_r)$$

com $r = n(n-1)/2$.

A Figura 3.6 ilustra o funcionamento deste esquema de adaptação para um problema com duas variáveis de decisão. Neste figura, os pontos pertencentes à elipse sujeita a rotação são os descendentes com igual probabilidade de serem gerados a partir de um ponto x.

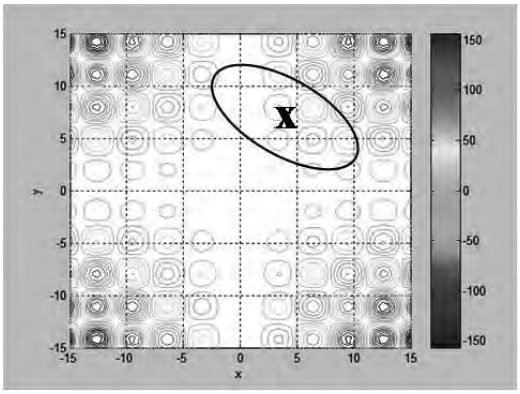


Figura 3.6: Adaptação Não Isotrópica com Rotação

A mutação é feita de acordo com a seguinte equação:

$$\begin{aligned}&\mathbf{x}' = \mathbf{x} + \mathbf{z} \text{ com } \mathbf{z} \sim N(0, \mathbf{C}')\\ &\mathbf{C}' = \begin{cases} \sigma_i^2 & \text{se } i = j \\ \frac{1}{2}(\sigma_i^2 - \sigma_j^2)\tan(2\alpha_{ij}) & \text{se } i \text{ e } j \text{ correlacionados} \\ 0 & \text{se } i \text{ e } j \text{ não correlacionados} \end{cases} .\end{aligned} \tag{3.9}$$

O Algoritmo 4 é o algoritmo da EE-$(\mu/\rho_{,}^{+}\lambda)$ com adaptação não isotrópica e com rotação. As funções *agrupar*, $recombinar_s$ e $recombinar_x$ são semelhantes às dos algoritmos anteriores. No entanto, a função $mutar_s$ muta os desvios padrão aplicando agora a regra não isotrópica com rotações expressa na equação (3.8). Finalmente, a função $mutar_x$ corresponde à mutação das variáveis de decisão tal como é descrita na equação (3.9) tendo em conta as covariâncias, i.e., as associações entre variáveis de decisão.

Critério de Paragem

O critério de convergência permite definir o término do processo de procura. Em geral, o critério de convergência adoptado para as estratégias evolutivas multimembros é terminar a procura quando as seguintes condições são verificadas:

$$\left|f_{\max}^{(k)} - f_{\min}^{(k)}\right| \leq \varepsilon_3 \text{ ou } \frac{\left|f_{\max}^{(k)} - f_{\min}^{(k)}\right|}{\left|\overline{f}^{(k)}\right|} \leq \varepsilon_4,$$

onde $f_{\max}^{(k)}$, $f_{\min}^{(k)}$ e $\overline{f}^{(k)}$ são, respectivamente, o maior e o menor valor da função objectivo e a média dos valores da função objectivo da população de μ progenitores, ε_3 e ε_4 dependem da precisão do computador utilizado, sendo $\varepsilon_3 > 0$ e $\varepsilon_4 > 0$.

6. Tratamento das Restrições

Os esquemas evolutivos, tal como foram apresentados, permitem resolver problemas de optimização sem restrições. É possível tratar problemas restrições de desigualdade do tipo $g_j(\mathbf{x}) \geq 0$ com $j = 1, \ldots, m$, utilizando um mecanismo de eliminação das soluções não admissíveis, i.e., em cada geração,

Algoritmo 4 Estratégia Evolutiva $(\mu/\rho\overset{+}{,}\lambda)$ com Adaptação Não Isotrópica com Rotação

Require: μ, ρ, λ, Δ_σ, $\Delta_{\sigma'}$, β

$t \leftarrow 0$

$\mathbf{P}^{(t)} \leftarrow \left\{(\mathbf{x}_m^{(t)}; \mathbf{s}_m^{(t)}; \mathbf{a}_m^{(t)}; f(\mathbf{x}_m^{(t)}) : m = 1, \ldots, \mu\right\}$ [onde $\mathbf{x}_m^{(t)} = (x_1, x_2, \ldots, x_n)$, $\mathbf{s}_m^{(t)} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ e $\mathbf{a}_m^{(t)} = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ com $k = n(n-1)/2$]

enquanto CP falso **faça**

  para $l = 1$ to λ **faça**

    $\mathbf{g}_l \leftarrow agrupar(\mathbf{P}^{(t)}, \rho)$

    $(\mathbf{s}_l, \mathbf{a}_l) \leftarrow recombinar_s(\mathbf{g}_l)$

    $\mathbf{x}'_l \leftarrow recombinar_x(\mathbf{g}_l)$

    $(\mathbf{s}'_l, \mathbf{a}'_l) \leftarrow mutar_s(\mathbf{s}_l, \mathbf{a}_l)$

    $\mathbf{x}''_l \leftarrow mutar_x(\mathbf{x}'_l, \mathbf{s}'_l, \mathbf{a}'_l)$

  fim para

  $\mathbf{D}^{(t)} \leftarrow \left\{(\mathbf{x}_m^{''(t)}; \mathbf{s}_m^{''(t)}; \mathbf{a}_m^{''(t)}; f(\mathbf{x}_m^{''(t)})) : m = 1, \ldots, \mu\right\}$

  $\mathbf{P}^{(t+1)} \leftarrow \begin{cases} \text{selecção}(\mathbf{D}^{(t)}) & \text{se } EE-(\mu/\rho, \lambda) \\ \text{selecção}(\mathbf{D}^{(t)} \cup \mathbf{P}^{(t)}) & \text{se } EE-(\mu/\rho + \lambda) \end{cases}$

  $t \leftarrow t + 1$

fim enquanto

$\mathbf{x}_{final} \leftarrow melhor(\mathbf{P}^{(t)})$

retorno $\mathbf{x}_{final}$

se a mutação ou recombinação gerarem um ponto que não satisfaz as restrições, este não é aceite. Desta forma, a procura restringe-se ao interior da região admissível. No entanto, é muitas vezes difícil especificar um vector inicial $\mathbf{x}_0$ que seja uma aproximação ao óptimo e que satisfaça todas as restrições (se for utilizado um vector inicial que não satisfaça as restrições poderá demorar muito tempo até que o processo de procura determine um ponto da região admissível). Um dos processos para a determinação de um vector inicial na região admissível é o descrito por Box (1965). Neste processo, uma função objectivo auxiliar $r(\mathbf{x})$ representando a soma dos valores das funções das restrições violadas é construída:

$$r(\mathbf{x}) = \sum_{j=1}^{m} g_j(\mathbf{x})\delta_j(\mathbf{x}),$$

onde

$$\delta_j(\mathbf{x}) = \begin{cases} -1 \text{ se } g_j(\mathbf{x}) < 0 \\ 0 \text{ se } g_j(\mathbf{x}) \geq 0 \end{cases}.$$

Um decréscimo no valor da função $r(\mathbf{x})$ representa uma aproximação à região admissível. Assim, quando $r(\mathbf{x}) = 0$ então $\mathbf{x}$ satisfaz todas as restrições e pode ser utilizado como vector inicial. Enquanto não é encontrado um ponto admissível, a procura é feita com base na função $r(\mathbf{x})$ definida anteriormente.

O esquema atrás descrito permite tratar apenas restrições do tipo desigualdade. As restrições do tipo igualdade $h_i(\mathbf{x}) = 0$ podem ser reformuladas como restrições de desigualdade da seguinte forma:

$$h_i(\mathbf{x}) \geq 0 \wedge -h_i(\mathbf{x}) \geq 0.$$

Para além disso, podem ser consideradas quantidades positivas e pequenas, $\underline{\varepsilon}$ e $\overline{\varepsilon}$ de tal forma que $-\underline{\varepsilon} \leq h_i(x) \leq \overline{\varepsilon}$, i.e.:

$$h_i(\mathbf{x}) + \underline{\varepsilon} \geq 0 \wedge -h_i(\mathbf{x}) + \overline{\varepsilon} \geq 0.$$

No entanto, o tratamento de restrições do tipo igualdade pode criar dificuldades às EEs, dado que encontrar novos pontos que sejam admissíveis pode traduzir-se num comportamento oscilatório.

Outras técnicas de tratamento de restrições podem ser implementadas. Coello Coello (2002) faz uma revisão, no contexto dos algoritmos evolucionários, de diversas técnicas para o tratamento de restrições do tipo desigualdade e igualdade.

7. Estratégias Evolutivas Avançadas

Nesta secção são referidas algumas abordagens baseadas em EEs que utilizam técnicas avançadas quer para a adaptação dos parâmetros de procura quer pela utilização de informação de procura não local.

Uma dessas abordagens são as Meta-EEs propostas por Rechenberg (1994) (também, chamadas de *Nested Evolution Strategies*). As Meta-EEs utilizam informação não local na procura. Para isso, a procura é hierarquicamente organizada em diversas EEs que fazem procura local por um período de γ gerações. Há uma EE externa cujas populações são constituídas por diversas EEs internas. Assim, a notação usada para descrever uma Meta-EE passa a ser EE-$[\mu'/\rho'^{+}_{,}\lambda'(\mu/\rho^{+}_{,}\lambda)^{\gamma}]$ onde os parêntesis rectos dizem respeito à EE externa e os curvos às EEs internas. Neste algoritmo, há μ' populações de progenitores de EE-$(\mu/\rho^{+}_{,}\lambda)$ que geram λ' EE-$(\mu/\rho^{+}_{,}\lambda)$ descendentes. Após γ gerações, a selecção correspondente à EE externa é feita para determinar as melhores μ' das EE-$(\mu/\rho^{+}_{,}\lambda)$ que servirão de base para a próxima população de λ' EE-$(\mu/\rho^{+}_{,}\lambda)$ descendentes. As Meta-EEs podem ser utilizadas em optimização global, multi-local, problemas inteiros mistos e, também, na optimização dos próprios parâmetros de procura das EEs internas.

Outras abordagens, são as propostas por Ostermeier et al. (1994) and Hansen e Ostermeier (2001), que utilizam informação do progresso ao longo das gerações para fazer a auto-adaptação dos parâmetros de procura. No algoritmo proposto por Ostermeier et al. (1994), designado por CSA-ES (*Cumulative Step-size Adaptation Evolution Strategy*), informação relativa ao percurso evolutivo percorrido ao longo das gerações é utilizada para adaptar os tamanhos do passo. No algoritmo CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*) desenvolvido por Hansen e Ostermeier (2001), o percurso evolutivo cumulativo é também utilizado para adaptar a matriz de covariâncias que permite a aplicação de mutação correlacionada. O estudo e a descrição aprofundada destas estratégias evolutivas avançadas sai fora do âmbito deste capítulo, podendo o leitor encontrar os detalhes destas propostas nas referências bibliográficas atrás indicadas.

(Página deixada propositadamente em branco)

CAPÍTULO 4

Programação Genética

Douglas A. Augusto *Helio J. C. Barbosa*

Laboratório Nacional de Computação Científica, Petrópolis, RJ

Este capítulo descreve a metaheurística evolucionária denominada *programação genética*, que visa a construção automática de programas de computador por meio de um processo iterativo inspirado na evolução por seleção natural.

A programação genética (PG), cujo desenvolvimento é atribuído a Koza (1992), é uma metaheurística estocástica de otimização global baseada no princípio darwiniano de seleção natural, sendo, pois, uma abordagem da computação evolucionária. A PG destina-se à evolução de programas de computador em linguagens arbitrárias, em outras palavras, a PG otimiza estruturas funcionais capazes de realizar operações, sejam elas lógicas, aritméticas, condicionais e de desvios, que normalmente mapeiam *entradas* em *saídas*.

Nesta concepção, submete-se uma população contendo um determinado número de indivíduos—programas candidatos criados aleatoriamente—ao processo simulado de evolução segundo a seleção natural. Neste processo iterativo figuram a cada geração: a seleção de indivíduos promissores para procriação, a formação de seus descendentes por meio de operações genéticas como cruzamento e mutação, e finalmente a inserção destes novos indivíduos na população, marcando-se o início de uma nova geração. Muito embora não existam garantias de progresso e tampouco de obtenção de soluções ótimas, como característico em qualquer metaheurística, essa dinâmica tende, ao longo das gerações, a produzir soluções candidatas incrementalmente mais adaptadas.

Destacam-se como características típicas da programação genética as seguintes qualidades:

- *Robustez* – o modelo populacional e o processo estocástico são os dois principais fatores por trás da programação genética responsáveis pela solidificação da tolerância a ruídos.
- *Exigência de pouco conhecimento* sobre o domínio – os requisitos acerca do domínio de aplicação podem ser relaxados; basicamente, espera-se apenas uma função capaz de comparar a qualidade relativa dos indivíduos com certa precisão.
- Produz *soluções simbólicas* – usualmente as soluções obtidas pela programação genética são imediatamente legíveis e interpretáveis ("código-fonte disponível"), em decorrência do processo evolutivo atuar diretamente no nível simbólico das estruturas.
- Possui *paralelismo natural* – as demandas computacionais da programação genética podem ser trivialmente particionadas em vários níveis, com emprego ou não de modelos distribuídos bioinspirados, permitindo-se assim redução no tempo de execução e/ou escalabilidade para problemas mais complexos.
- É facilmente *extensível/modificável* – a programação genética é uma metaheurística extremamente versátil, e dessa forma admite, por exemplo, hibridizações (funcionamento junto a outras técnicas), integração de outros modelos evolucionários (p. ex., co-evolução, nichos, múltiplos objetivos) e as mais diversas representações e linguagens para os programas (como representação formal por gramáticas).

O processo evolutivo da programação genética pode ser visualizado pelo fluxograma da Figura 4.1. Como exposto na figura, os componentes e etapas da PG, que são discutidos no decorrer deste capítulo,

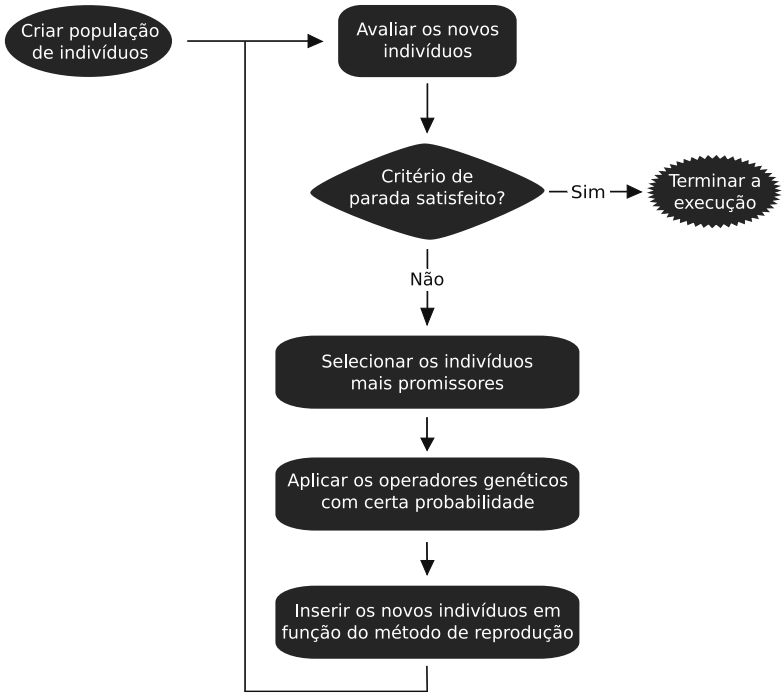


Figura 4.1: Fluxograma do funcionamento da programação genética.

podem ser sintetizados por:

1. *Criação da população* – determina quem e como são formados os primeiros indivíduos candidatos. Para tanto, dois conceitos devem estar definidos:
   (a) representação dos programas – a estrutura funcional na qual o processo evolutivo atuará, isto é, a entidade que será submetida às operações genéticas e avaliações.
   (b) conjuntos primitivos de funções e terminais – estabelece as instruções básicas ("palavras-chave") da linguagem dos programas que, quando devidamente combinadas, expressam a solução esperada.
2. *Avaliação dos indivíduos* – função que mede quão bem um determinado indivíduo executa a tarefa do problema, em outras palavras, é a função-objetivo a ser maximizada.
3. *Critério de parada* – determina quando o processo deve ser interrompido; normalmente por obtenção de uma solução satisfatória ou por limites de tempo de execução.
4. *Esquemas de seleção* – definem como os indivíduos mais bem adaptados são favorecidos na disputa pela procriação.
5. *Operadores genéticos* – são as técnicas de criação de novos indivíduos com base no material genético de seus genitores. Tipicamente subdividem-se em:
   (a) cruzamento – recombinação do material genético de dois indivíduos.
   (b) mutação – deriva um indivíduo como uma variação de um outro.
6. *Métodos de reprodução* – estipulam como os descendentes serão inseridos na população.

Aplicações

Teoricamente, a programação genética pode ser aplicada em qualquer problema que atenda duas propriedades fundamentais:

1. *que a solução possa ser expressa como um programa de computador em uma dada linguagem.*
2. *que seja possível definir uma medida capaz de, dadas duas soluções candidatas, dizer com razoável precisão quais delas é a melhor* ("função de avaliação").

Satisfeitas estas propriedades é possível, em tese, aplicar a programação genética. Na prática, porém, em problemas excessivamente complexos—como os que envolvem avaliações/simulações computacionalmente custosas, ou aqueles cuja estrutura da solução esperada é demasiadamente grande e intricada—a PG esbarra nos limites computacionais no que tange tempo e espaço de processamento, inviabilizando portanto o seu uso.[1] Nestes casos, contudo, uma possível alternativa é o uso da PG em ambientes computacionais de alto desempenho, servindo-se de implementações massivamente paralelas/distribuídas.

A despeito da restrição em certas classes de problemas, a programação genética tem sido usada em áreas diversas e expressivas. A relação apresentada a seguir lista, em caráter ilustrativo, uma pequena parcela dos domínios e suas aplicações na qual a PG tem sido bem sucedida.

Reconhecimento de padrões Neste domínio procura-se a evolução de programas capazes de identificar padrões normalmente implícitos em massas de dados. Exemplos constituem o diagnóstico automatizado de doenças mediante informações de exames do paciente, análise de crédito e cálculo de risco baseando-se em informações históricas e vigentes acerca do tomador de crédito, ou reconhecimento óptico de caracteres digitais/manuscritos.

[1] Naturalmente, à medida que a tecnologia avança e computadores mais rápidos são construídos, a programação genética amplia o seu leque de atuação.

Regressão simbólica Esta área trata da descoberta de expressões matemáticas (expressões simbólicas). Existem aplicações de ajuste de curvas, indução de sequências, derivação e integração simbólica, descoberta de identidades matemáticas, previsão de séries temporais, entre outras.

Robótica Busca-se desenvolver programas que controlem precisamente ações de robôs em diversas tarefas, tais como as de automação industrial.

Estratégias de jogos O objetivo é evoluir estratégias que, dado o estado atual do jogo—e possivelmente o histórico de jogadas—, decidam otimamente qual deve ser a próxima jogada.

Processamento de imagens Uma das aplicações envolve a compactação de imagens, isto é, a descoberta de programas compactos que quando executados recriem a imagem original.

Arte A evolução de programas que codificam (artificialmente) pinturas de quadros, por exemplo, é uma das possíveis aplicações da PG nesta área.

1. Descrição da Programação Genética

Representação dos Programas

Existe uma grande variedade de estruturas capazes de representar programas de computador no âmbito da programação genética.[2] Há no entanto três principais grupos conceituais de representação: *linear* (Oltean et al., 2009), por *árvore* (Koza, 1992) ou por *grafos* (Poli, 1999; Teller, 1996; Miller e Smith, 2006).

Dentre os grupos, a representação por árvore é a mais popular, status este possivelmente decorrente da sua tradição de uso na programação genética, aliada ao poder/espontaneidade de expressão de programas e relativa simplicidade. Todavia, esta classificação é *conceitual*, e difere da *representação computacional*; por exemplo, muito embora a árvore seja uma estrutura naturalmente hierárquica, ela pode ser implementada e operada linearmente sem que haja violação de suas propriedades (Pelikan et al., 1997; Augusto e Barbosa, 2000). Ainda, um terceiro nível de classificação pode ser introduzido: a *representação semântica*. Neste, a representação conceitual é complementada com regras/restrições que governam como a estrutura em questão pode ser operada; pode-se citar como membros desta classificação, por exemplo, a representação por gramática sobre a estrutura árvore (Whigham, 1995) e sobre a estrutura linear (O'Neill e Ryan, 2001), e restrições de tipo para a estrutura árvore (Montana, 1994).

A despeito do extenso leque de opções de representação, considera-se neste capítulo a representação pela estrutura de dados *árvore*, cujo emprego é bastante comum na PG. Nesta representação, o cromossoma de um indivíduo apresenta um aspecto gráfico como exemplificado na Figura 4.2. O indivíduo **A** descreve a expressão

$$\text{se } A \wedge B \text{ então } C, \text{ senão } (3.14 \times X),$$

enquanto o indivíduo **B** descreve a função matemática

$$\sin\left(\sqrt{0.987} - \cos(X \times Y)\right).$$

[2] Essas estruturas são muitas vezes referidas por *cromossoma* ou *genoma* do indivíduo.

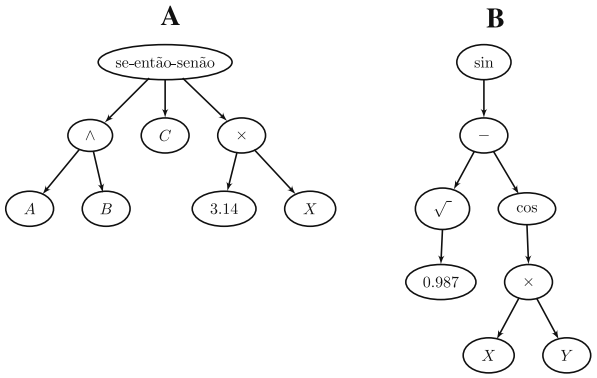


Figura 4.2: Indivíduos representados pela estrutura árvore.

Conjuntos Primitivos de Funções e Terminais

Essenciais ao funcionamento da programação genética, como originalmente concebida, são os denominados conjuntos de *funções* ($\mathbb{F}$) e *terminais* ($\mathbb{T}$), que são relativos ao domínio da aplicação. São eles que definem as ferramentas (as primitivas) que estarão à disposição do processo evolucionário para a construção de estruturas mais complexas, isto é, definem a linguagem—e o feitio do espaço de busca – onde a evolução poderá explorar.

O conjunto $\mathbb{F}$ é responsável pelo fornecimento de funções que requerem argumentos, ou seja, os *operadores*. Exemplos incluem operadores matemáticos ($+$, $-$, $\times$, $\div$, sin, cos, $\sqrt{}$, log, x^y), lógicos ($\vee$, $\wedge$, $\neg$), relacionais ($<$, $>$, $=$), condicionais (se x então y senão z) e laços iterativos (enquanto x faça y).

Por sua vez, o conjunto $\mathbb{T}$ provê os *operandos*, isto é, variáveis/atributos (x, y, *idade*, *salário*), constantes (π, e) e funções que não requerem argumentos (*random()*, *tempo_decorrido()*).

Os conjuntos de funções e terminais podem ser livre e recursivamente combinados; a única restrição é que seja obedecida a aridade[3] de cada primitiva. Portanto, todo fragmento ou estrutura de um programa na população, em qualquer estágio do processo evolutivo, nada mais é do que uma combinação particular entre elementos de $\mathbb{F}$ e $\mathbb{T}$. No entanto, para que estes conjuntos possam ser computacionalmente manipulados e permitam a expressão da solução procurada, duas propriedades devem ser satisfeitas:

Suficiência Do fato de que qualquer estrutura da população é um arranjo particular de $\mathbb{F}$ e $\mathbb{T}$, decorre que é condição *necessária* para a obtenção da solução esperada que esta possa ser expressa como uma combinação das primitivas definidas para o problema. Esta óbvia constatação é conhecida como *propriedade de suficiência*, e diz que os elementos dos conjuntos $\mathbb{F}$ e $\mathbb{T}$ devem ser suficientes para, em ao menos uma combinação específica, descrever a solução esperada.

Na prática, entretanto, a propriedade de suficiência não soa tão óbvia, devido ao fato de que em muitos problemas—especialmente os reais e de grande porte—não se conhece o aspecto da solução, e dessa forma pouco se pode cogitar a respeito da formação dos conjuntos de funções e terminais. Neste cenário, é tentador adicionar aos conjuntos tantas primitivas quanto se possa; contudo, tal procedimento aumenta enormemente o espaço de busca, e pode, portanto, seriamente degradar o progresso da evolução.

Consistência Uma segunda propriedade que concerne aos conjuntos de primitivas é a denominada *consistência*. Consistência é a condição de integridade, e requer que cada operador do conjunto

[3] *Aridade* é o número requerido de argumentos. Os terminais em $\mathbb{T}$ têm sempre aridade *zero*.

de funções seja suficientemente versátil e robusto para aceitar como argumento qualquer terminal em $\mathbb{T}$ ou valores de retorno de qualquer função em $\mathbb{F}$. Em outras palavras, os conjuntos de terminais e funções têm que ser tais que, quaisquer combinações entre $\mathbb{F}$ e $\mathbb{T}$, ou $\mathbb{F}$ e ele próprio, desde que respeitadas as aridades, sejam válidas.[4]

A consistência é uma propriedade demasiadamente relaxada no que se refere às relações e restrições entre as primitivas, isto é, no que tange a definição da linguagem dos programas. Ao mesmo tempo em que os conceitos dos conjuntos de funções e terminais da programação genética canônica proveem simplicidade e uniformidade, a ausência de restrições pode causar: (i) crescimento não justificado do espaço de busca – regiões conceitualmente sem sentido passam a fazer parte dele; e (ii) dificuldade de satisfazer a condição de consistência quando estão envolvidos diferentes tipos e estrutura de dados, tornando-a inviável ou, na melhor das hipóteses, forçando uma adaptação crassa.[5] Felizmente, na prática estes cenários não ocorrem com frequência. No entanto, para domínios de aplicação onde as deficiências e limitações da noção dos conjuntos de terminais/funções mostram-se preocupantes ou proibitivas, é possível adotar representações mais sofisticadas, como, entre outras, a representação formal por gramática (Whigham, 1995).

Criação da População

A criação da população visa formar uma amostragem representativa do espaço de programas, que seja preferencialmente bem distribuída e sem tendenciosidade a favor de configurações particulares. Entretanto, dado que o espaço de possíveis programas é geralmente infinito—quando não se impõe limites acerca do tamanho das soluções—, torna-se impossível amostragens estritamente uniformes; o que se pode fazer, todavia, é tentar evitar grandes distorções (Poli et al., 2008a).

Na prática, a criação dos indivíduos iniciais, e portanto da população, é realizada de maneira essencialmente *aleatória*, isto é, as instruções básicas dos conjuntos $\mathbb{F}$ e $\mathbb{T}$ são recursivamente combinadas ao acaso, em número e complexidade variáveis, na estrutura do indivíduo.[6] Embora o processo de criação seja não-determinístico, é preciso que se respeite a aridade de cada operador; em outras palavras, se uma determinada função possui n argumentos, então, considerando-se a estrutura árvore, o nó representando este operador terá exatamente n nós filhos, cada qual constituindo um argumento.

Na tentativa de se criar uma boa amostragem inicial, métodos que delineiam a forma como as árvores iniciais são construídas foram propostos. Alguns desses métodos são descritos a seguir.

Método de Criação Completa (*Full Creation Method*)

Na PG é interessante que se imponha limites nos tamanhos dos cromossomas (árvores), evitando assim, que as soluções candidatas cresçam indefinidamente, provocando esgotamento de memória e degradação no desempenho. Um destes limites está relacionado ao momento da criação dos indivíduos, e é geralmente especificado por profundidade máxima.[7]

Neste método de criação, exemplificado pela Figura 4.3, todo comprimento entre o nó raiz e qualquer nó terminal é sempre igual à profundidade máxima especificada. Isto significa que, até se alcançar a profundidade máxima, todos os nós terão um ou mais filhos (funções com mais de

[4] Uma das situações mais comuns envolve o operador de divisão, que normalmente precisa ser definido de modo a tratar a divisão por zero, p. ex. retornar a constante 1 neste caso.

[5] Por exemplo, não faz sentido conceitual a multiplicação de valores booleanos, no entanto, seria necessário garantir a factibilidade desta operação para que o processo evolutivo fosse viável.

[6] Em certas aplicações, no entanto, conhecendo-se alguns padrões úteis e promissores, é possível introduzi-los diretamente na população inicial, mas é preciso cautela para que estas sementes não causem convergência prematura.

[7] Profundidade é o maior comprimento (ramo) entre o nó raiz e os demais nós de uma árvore. Quando se tem noção da dimensão da solução do problema, além do limite máximo de profundidade, é usualmente também especificado um limite mínimo.

um argumento) e, ao se atingir a profundidade determinada, os nós não possuirão filhos (funções sem argumentos, variáveis ou constantes).

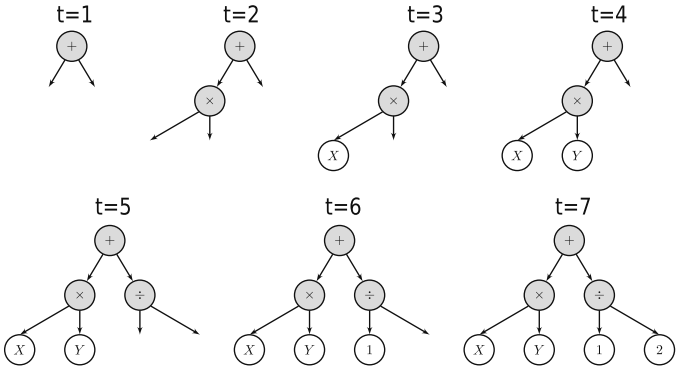


Figura 4.3: Exemplificação passo-a-passo do método de criação *completa*.

Método de Criação Livre (*Grow Creation Method*)

Diferentemente do método completo, este permite que nós com qualquer número de filhos (argumentos) sejam adicionados no momento de criação, assim, não é necessário que a árvore seja completa. Isto é, se ocorrer em algum ramo um nó sem filhos (terminal), este ramo será encerrado. A Figura 4.4 ilustra esse processo.

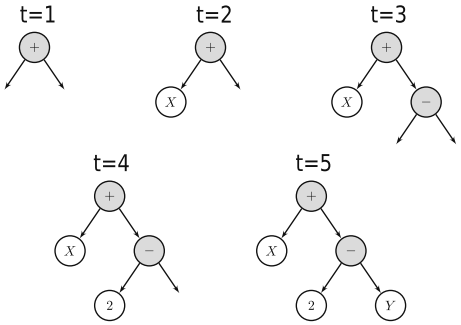


Figura 4.4: Exemplificação passo-a-passo do método de criação *livre*.

Efeito Escada (*Ramping*)

No momento da criação, a população é dividida em segmentos. A cada segmento é atribuído uma profundidade máxima, de modo que esta profundidade máxima varie entre a profundidade mínima real e a máxima real. Esta atribuição geralmente é feita em ordem crescente. Por exemplo, considere uma população com 50 indivíduos, e suponha que a profundidade mínima seja 2 e a máxima 6. Então, a população seria dividida em cinco segmentos, cada qual com dez indivíduos. Dessa forma, o primeiro segmento receberia uma profundidade máxima de 2, o segundo 3, e assim, até que o último receba a profundidade 6.

Método Misto de Criação (*Ramped Half-and-Half*)

Segundo Koza (1992), este é o método mais recomendado. É capaz de gerar uma ampla diversidade de indivíduos, produzindo várias formas e tamanhos distintos de árvores. A ideia consiste

em fazer com que cada metade de um segmento produzido por "*ramping*" seja criada ou com o método completo (*full*) ou com o método livre (*grow*). Isto é, metade de um segmento será criada pelo método completo e a outra metade com o método livre.

Avaliação dos indivíduos

A avaliação dos indivíduos, que depende fundamentalmente do domínio da aplicação, mede o êxito com que os programas realizam a tarefa alvo, isto é, define a *aptidão*. Tipicamente, cada programa da população é executado sobre conjuntos de argumentos, e então os valores de retorno são comparados com os valores esperados, que são conhecidos; quanto mais próximos os valores de retorno estão dos valores esperados, mais bem adaptado é o indivíduo em questão.

Em problemas de regressão, por exemplo, interessa minimizar a discrepância entre os valores esperados para os pontos fornecidos e os valores obtidos pela função sendo avaliada; portanto, uma medida de avaliação poderia ser o *inverso* da soma do quadrado dos erros:

$$f_{\text{aptidão}} = \left[\sum_{i=1}^{k}[f(p_i) - Y_i]^2\right]^{-1},$$

ou então o *negativo* da soma do quadrado dos erros:

$$f_{\text{aptidão}} = -\sum_{i=1}^{k}[f(p_i) - Y_i]^2,$$

onde k é o número de pontos, $p_i \in \mathbb{R}^n$ é o i-ésimo ponto, f é a função—codificada por um indivíduo qualquer—sendo avaliada ($f : \mathbb{R}^n \mapsto \mathbb{R}$), e $Y_i \in \mathbb{R}$ é o valor esperado para o ponto de índice i.

Já em problemas de classificação de dados[8] o objetivo primário é medir a acurácia de predição sobre um determinado conjunto de amostras de treinamento; dessa forma, a aptidão de uma árvore classificadora C de um indivíduo ($C : \mathbb{R}^n \mapsto \mathbb{N}$), normalizada para o intervalo $[0, 1]$, poderia ser dada pela *taxa de acertos*:

$$f_{\text{aptidão}} = \frac{1}{k}\sum_{i=1}^{k}\mathbb{I}[C(a_i) = Y_i],$$

onde k é o número total de amostras de treinamento, $a_i \in \mathbb{R}^n$ e $Y_i \in \mathbb{N}$, respectivamente, a i-ésima amostra e sua classe esperada, e $\mathbb{I}[\pi]$ retorna 1 se a condição π é satisfeita ou 0 caso contrário.

No entanto, nesses casos geralmente é útil ponderar a aptidão do indivíduo em função de sua complexidade/tamanho, visando assim a obtenção de soluções mais compactas, legíveis e potencialmente com maior poder de generalização. Uma medida comum é a introdução de um coeficiente de penalização por complexidade, que se propõe a reduzir a aptidão de um indivíduo proporcionalmente ao tamanho de sua estrutura.[9] A aptidão de um indivíduo pode ser, portanto, reescrita como:

$$f'_{\text{aptidão}} = f_{\text{aptidão}} - c \times \ell,$$

onde c é o coeficiente de penalização, que usualmente é uma constante, e ℓ é uma medida de complexidade, onde normalmente emprega-se o tamanho da estrutura do indivíduo como, por exemplo, o número de nós da árvore que representa o programa.

[8] A classificação de dados é a tarefa que, fornecido um conjunto de dados como treinamento, visa construir uma entidade com a propriedade de mapear atributos de entrada em um valor discreto que representa a classe/categoria de uma dada amostra. Esta entidade é denominada *classificador* ou *preditor*.

[9] Soluções mais sofisticadas para o controle de complexidade são discutidas em (Poli et al., 2008a).

Critério de parada

Idealmente, uma execução da programação genética deveria terminar quando, e somente quando, uma solução ótima fosse encontrada segundo a função de aptidão definida. Infelizmente, dependendo da complexidade do problema e dos recursos computacionais disponíveis, isto pode não ser viável. Nesses casos, o relaxamento desta condição no intuito de obtenção de soluções "suficientemente boas" pode ser admissível—ou ser de fato a única alternativa. A implementação deste conceito é feita pela adição de um ou mais critérios de parada, comumente:

- Aptidão dentro da faixa aceitável – o processo é interrompido quando uma solução aproximada é obtida.
- Número máximo de gerações ou avaliações – a execução é interrompida ao se atingir um número pré-estabelecido do gerações ou avaliações.
- Tempo limite de execução – análogo ao item anterior, mas é guiado pelo tempo despendido.
- Estagnação do processo – a execução termina quando é detectada estagnação do processo evolutivo, isto é, quando por um certo período nenhuma solução aprimorada é obtida.

Esquemas de seleção

A seleção é o instrumento pelo qual os algoritmos evolucionários conduzem a busca para as regiões mais promissoras do espaço e, que, efetivamente, permite o processo de otimização. Ela simula o efeito da *seleção natural*, sendo responsável por escolher os indivíduos da população que deixarão descendentes para a geração seguinte, ou seja, aqueles que participarão das operações genéticas. Portanto, é natural esperar que os esquemas de seleção favoreçam aqueles indivíduos mais bem adaptados, pois a transmissão de bom material genético potencializa o desenvolvimento de soluções aperfeiçoadas.

A diferença entre a probabilidade de seleção dos melhores indivíduos e a probabilidade de seleção dos piores denomina-se *pressão de seleção*. Quando a pressão de seleção é baixa, indivíduos de menor qualidade terão mais chance de disseminar seus genes; por outro lado, quando a pressão é alta, somente aqueles indivíduos muito bem avaliados, em média, têm oportunidade de ser selecionados. Embora possa parecer que, a princípio, a alta pressão de seleção é sempre vantajosa, na realidade ela pode levar o processo evolucionário rapidamente à indesejável convergência prematura. Isto ocorre, notoriamente, devido à degradação da diversidade na população, já que apenas um pequeno seleto grupo de indivíduos domina o processo.

Embora existam distintas maneiras de se implementar a seleção (Goldberg e Deb, 1991), cada qual diferindo particularmente no que tange a dinâmica da pressão de seleção e custo computacional, uma variação bastante simples, eficiente e conveniente, nomeada *torneio* (Brindle, 1981), é comumente empregada na programação genética. Neste esquema, k indivíduos são selecionados *aleatoriamente* da população, suas k aptidões são comparadas e o indivíduo detentor da melhor aptidão é selecionado. Nota-se que o valor de k, de fato, define a pressão de seleção, isto é, quanto maior o número de competidores mais rigorosa a seleção se torna.

Operadores genéticos

A função dos operadores genéticos é construir soluções derivadas a partir de outro(s) indivíduo(s). É através desse processo de experimentação de recombinações e/ou variações sobre padrões promissores que o espaço de busca é explorado.

Os dois principais operadores genéticos, descritos a seguir, são o *cruzamento*, que atua sobre dois indivíduos, e *mutação*, que atua sobre um único indivíduo.

Cruzamento

O operador de cruzamento propõe-se a recombinar partes de estruturas previamente selecionadas pelo processo de seleção, na esperança de se derivar estruturas ainda melhores. Devido ao status de operador primário na programação genética, o cruzamento é tipicamente aplicado com alta probabilidade.

Na representação por árvore, o operador de cruzamento comuta sub-árvores, cujos nós raiz são arbitrariamente escolhidos, de dois indivíduos, gerando dessa maneira dois novos descendentes. A Figura 4.5 exemplifica a aplicação do operador de cruzamento sobre os pais **A** e **B**, produzindo os filhos **A'** e **B'**.

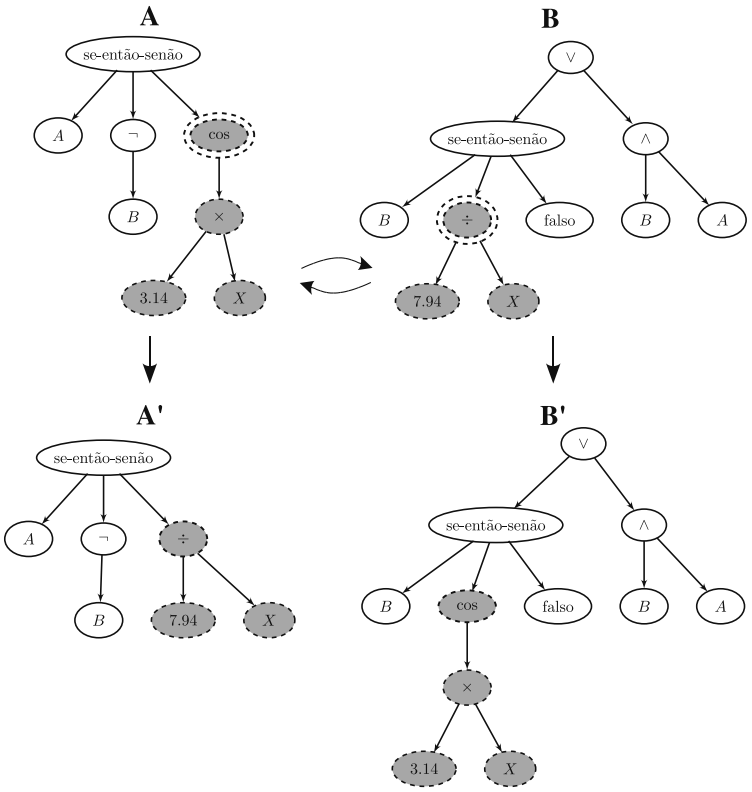


Figura 4.5: Cruzamento padrão.

Mutação

A mutação deriva um novo descendente mediante modificações na estrutura de um indivíduo selecionado. Estas modificações são normalmente de pequena escala ("perturbações"), e visam tanto a exploração de regiões vizinhas àquele indivíduo quanto a (re)introdução de material genético a fim de preservar a diversidade da população. As modificações sobre as estruturas são realizadas, tipicamente, de maneira absolutamente aleatória.[10]

A Figura 4.6 ilustra uma aplicação hipotética do operador padrão de mutação sob a representação por árvore. Primeiramente, uma sub-árvore do indivíduo **A** é escolhida ao acaso, então removida,

[10] Mas existem operadores especializados de mutação que agem deterministicamente; por exemplo, operadores de eliminação de redundância ou de simplificação de expressões em árvores.

e finalmente uma nova sub-árvore, criada aleatoriamente, é introduzida no local, gerando assim o descendente **A'**.

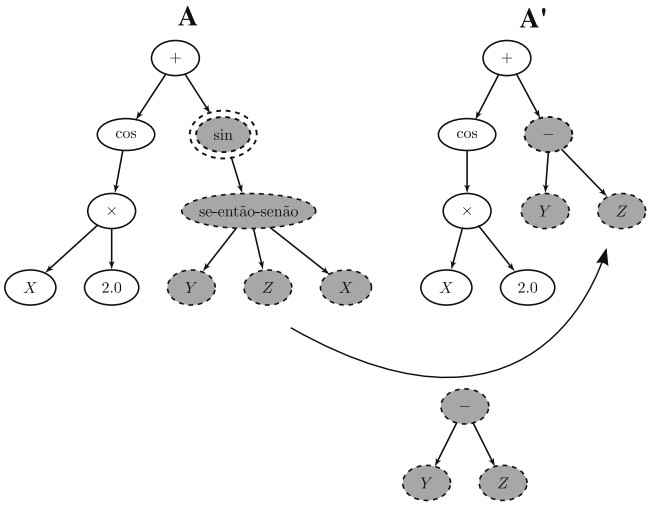


Figura 4.6: Mutação padrão.

Visando a promoção da diversidade da população e a exploração do espaço de busca por caminhos diferentes – às vezes inusitados –, variações do operador padrão de mutação foram desenvolvidas. Dentre estas, podem ser citadas a mutação tipo *nó*, *encolhimento* e *permutação*.

A mutação nó, como o nome indica, provoca uma alteração sobre algum nó da árvore. Assim, após selecionado um elemento, arbitrariamente, este é substituído por algum outro, de tal forma que o novo nó possua o mesmo número de argumentos que o original. A Figura 4.7 ilustra graficamente este processo.

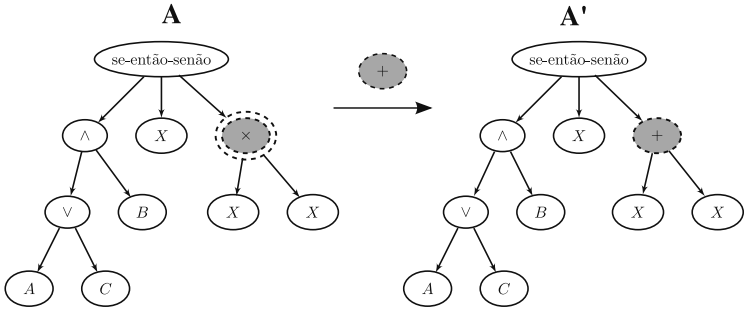


Figura 4.7: O processo de mutação tipo *nó*.

Por sua vez, a mutação tipo encolhimento objetiva reduzir o tamanho do cromossoma (árvore), inspirada no fato de que soluções menores (menos complexas) tendem a ser mais compreensíveis, e portanto desejadas. Seu procedimento consiste em, após escolhido um elemento aleatoriamente, selecionar um filho qualquer e, então, substituí-lo na posição do elemento escolhido (nó pai), excluindo-se todos os outros filhos e suas ramificações. Isto "compacta" a sub-árvore em questão, e portanto reduz o cromossoma como um todo. A Figura 4.8 ilustra visualmente a aplicação da mutação encolhimento.

Já a mutação por permutação provoca uma alteração na ordem (permuta) dos argumentos de uma

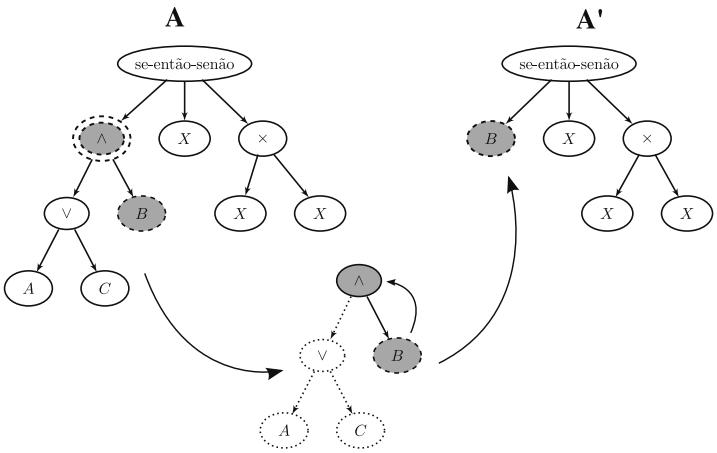


Figura 4.8: O processo de mutação tipo *encolhimento*.

função qualquer. Naturalmente, este operador só se aplica às funções com no mínimo dois argumentos. Um exemplo pode ser visto na Figura 4.9.

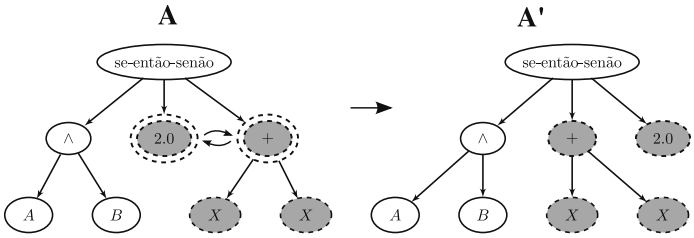


Figura 4.9: O processo de mutação tipo *permutação*.

Métodos de Reprodução

Os métodos de reprodução dizem respeito à política sob a qual os descendentes recém criados são introduzidos no processo evolucionário. Existem dois métodos fundamentais: *geracional* e *steady-state* (Syswerda, 1989; Whitley, 1989).

O método geracional foi o primeiro a surgir na literatura dos algoritmos de computação evolucionária; funciona basicamente assim: a cada geração os descendentes recém criados são alocados em uma população temporária; quando o número de indivíduos criados se iguala ao da população principal, a população temporária substitui a principal, e assim todos os indivíduos da geração passada são descartados—uma variação muitas vezes útil, conhecida por *elitismo*, é preservar os n melhores indivíduos (a "elite") da geração passada.

Uma desvantagem do método geracional, que pode ser decisiva em certas aplicações, é a necessidade de se alocar uma segunda população durante o processo evolutivo. O método de reprodução *steady-state*, por outro lado, insere os descendentes recém criados imediatamente na população atual, passando assim a coexistir com seus antecedentes. Quando um novo descendente é inserido, um indivíduo da população vigente deve ser eliminado para que o tamanho da população permaneça constante. Habitualmente, escolhe-se para ser substituído um indivíduo pouco adaptado (ou o menos adaptado), dessa forma, a evolução descarta soluções pouco promissoras e obtém-se, naturalmente, o

efeito de conservação dos melhores indivíduos análogo ao *elitismo* da reprodução geracional.

2. Algoritmo Protótipo

A programação genética, assim como a maioria das técnicas da computação evolucionária, não define passos algorítmicos precisos e imutáveis, sendo portanto, em sua essência, um modelo. Nesse sentido, é apresentado no Algoritmo 1 um protótipo em pseudo-código da PG segundo a reprodução do tipo geracional. Por sua vez, o Algoritmo 2 descreve a PG em sua variação de reprodução tipo *steady-state*.

A título de simplificação na listagem, ambos os algoritmos adotam como critério de parada tão somente um número máximo de iterações (gerações ou avaliações, dependendo do tipo de reprodução). Normalmente, entretanto, seria conveniente checar a cada avaliação de indivíduo se a solução candidata encontra-se dentro de uma faixa admissível de qualidade. Outros critérios de parada, como os descritos anteriormente, também podem ser usados.

Algoritmo 1 Programação Genética (reprodução geracional).

```
1: Criar aleatoriamente a população inicial P
2: Avaliar todos indivíduos de P
3: para geração ← 1 to N_G faça
4:    Copiar a elite de indivíduos de P para a população temporária P_tmp
5:    enquanto |P_tmp| < |P| faça
6:       Selecionar e copiar de P dois indivíduos bem adaptados, p_1 e p_2
7:       se [probabilisticamente] cruzamento então
8:          Cruzar p_1 com p_2, gerando os descendentes p'_1 e p'_2
9:          p_1 ← p'_1; p_2 ← p'_2
10:      fim se
11:      se [probabilisticamente] mutação então
12:         Aplicar operadores quaisquer de mutação em p_1 e p_2, gerando p'_1 e p'_2
13:         p_1 ← p'_1; p_2 ← p'_2
14:      fim se
15:      Avaliar p_1 e p_2 e inseri-los em P_tmp
16:   fim enquanto
17:   P ← P_tmp; descartar P_tmp
18: fim para
19: retorno melhor indivíduo encontrado
```

Parâmetros

Na Tabela 4.1 são descritos os parâmetros usuais cujos valores podem ser ajustados em execuções da PG para um certo problema. Em se tratando de valores ótimos para estes parâmetros, comumente diferentes problemas requerem diferentes valores de parâmetros; em outras palavras, dificilmente um conjunto ótimo de parâmetros para um problema será ótimo em outro problema.

Execução de Programas

Fundamental à avaliação dos indivíduos é a execução propriamente dita do programa codificado; por meio desta é possível medir quão bem o indivíduo desenvolve a tarefa em questão. O Algoritmo 3

Algoritmo 2 Programação Genética (reprodução *steady-state*).

1: Criar aleatoriamente a população inicial P
2: Avaliar todos indivíduos de P
3: **enquanto** número de *avaliações* $< N_A$ **faça**
4: Selecionar e copiar de P dois indivíduos *bem adaptados*, p_1 e p_2
5: **se** [probabilisticamente] *cruzamento* **então**
6: Cruzar p_1 com p_2, gerando os descendentes p'_1 e p'_2
7: $p_1 \leftarrow p'_1$; $p_2 \leftarrow p'_2$
8: **fim se**
9: **se** [probabilisticamente] *mutação* **então**
10: Aplicar operadores quaisquer de mutação em p_1 e p_2, gerando p'_1 e p'_2
11: $p_1 \leftarrow p'_1$; $p_2 \leftarrow p'_2$
12: **fim se**
13: Avaliar p_1 e p_2 e inseri-los em P, substituindo indivíduos *mal adaptados*
14: **fim enquanto**
15: **retorno** melhor indivíduo encontrado

Parâmetro	**Descrição**
Tamanho da população ($\|P\|$)	Número de indivíduos em evolução
Número de gerações (N_G)	Número máximo de gerações
Número de avaliações (N_A)	Número máximo de avaliações ("execuções")
Probabilidade de cruzamento	Probabilidade de aplicação do operador de cruzamento (tipicamente *alta*)
Probabilidade de mutação	Probabilidade de aplicação do operador de mutação (tipicamente *baixa*)
Pressão de seleção	Importância dada aos indivíduos mais bem adaptados durante o processo de seleção
Limites de profundidade	Profundidade máxima das árvores geradas
Tamanho da elite (geracional)	Quantidade de indivíduos mais bem adaptados que serão copiados para geração seguinte
Conjuntos de funções e terminais	Operadores e operandos primitivos que estabelecem as instruções básicas dos programas

Tabela 4.1: Parâmetros usuais da programação genética.

mostra em pseudo-código uma função recursiva para execução de programas codificados pela estrutura árvore.[11]

A execução de um programa inicia-se pela chamada da função Executa() sobre o nó raiz da árvore representada pelo indivíduo sendo avaliado. O algoritmo então testa o tipo do nó: se for *terminal* (nó folha) o seu valor é retornado imediatamente, e o percurso da recursão daquele ramo é finalizado; caso seja o tipo *função* (nó interno), o operador codificado pelo nó é aplicado sobre os seus argumentos[12] (nó filhos)—cujos valores são obtidos recursivamente—e finalmente o valor computado é devolvido. Comparando-se o valor final retornado por Executa() (ou os valores de sucessivas execuções) com o valor esperado obtém-se a discrepância do programa frente à tarefa alvo.

Um *terminal* pode codificar basicamente três espécies de valores: (i) constantes; (ii) variáveis; e (iii)

[11] Se o tempo de execução for crucial então uma versão não-recursiva do algoritmo seria mais apropriada, embora possivelmente menos elegante.
[12] O número de argumentos requerido pelo operador coincide com o número de filhos do nó.

Algoritmo 3 Execução de um programa.

EXECUTA(nó $raiz$)

1: **se** $raiz$.**tipo** = terminal **então**
2: **retorno** $raiz$.**valor**()
3: **fim se**
4: **retorno** $raiz$.**operador**(EXECUTA($raiz.filho_1$), EXECUTA($raiz.filho_2$), ...)

funções que não requerem argumentos. O tratamento da sentença "$raiz$.**valor**()" nos casos (i) e (iii) é trivial: basta que a constante seja retornada ou o valor da função, que não precisa de argumentos, seja computada. Porém, no caso (ii), o valor da variável precisa antes ter sido atribuído. Normalmente, um programa é executado sobre um conjunto finito de pontos (amostras de treinamento), e as variáveis, em cada execução, devem representar os valores do ponto em questão. Dessa forma, imediatamente antes de cada execução do programa os valores relativos ao ponto corrente são atribuídos, respectivamente, às variáveis do conjunto de terminais. Este procedimento assegura que os valores retornados das variáveis na i-ésima execução correspondem aos valores do i-ésimo ponto.

3. Exemplo de Aplicação: Regressão Simbólica

No intuito didático, e visando introduzir os conceitos da programação genética em uma aplicação prática, esta seção explica o que é regredir simbolicamente uma expressão, como os métodos tradicionais implementam a "regressão" e qual a vantagem da utilização da PG sobre estes métodos. Adicionalmente, uma execução hipotética passo-a-passo da PG em um problema simples de regressão simbólica é exemplificada.

Regressão Simbólica de uma Expressão

Regredir simbolicamente uma expressão é o ato de inferir simbolicamente uma expressão algébrica, dispondo apenas de um conjunto finito de amostras numéricas.[13] Basicamente, procura-se uma curva que melhor se ajuste aos dados fornecidos. O objetivo é encontrar uma função que minimize uma medida dos erros (ou discrepâncias) entre os valores previstos pela função e aqueles de fato observados.

Seja F o conjunto de todas as funções $f : \mathbb{R}^n \mapsto \mathbb{R}$ admissíveis para o problema, $\{p_1, p_2, \ldots, p_k\}$ o conjunto de pontos conhecido como amostragem e $Y \in \mathbb{R}^k$ o vetor cuja componente Y_i é o valor observado em $p_i \in \mathbb{R}^n$, onde k é o número de pontos fornecidos. Então, regredir simbolicamente é buscar por uma função f^* tal que

$$f^* = \arg\min_{f \in F} d(f, Y),$$

onde $d(f, Y) \geq 0$ mede a discrepância entre os valores previstos por f e aqueles observados. A função f^* é dita a mais adaptada (ou melhor ajustada) aos pontos fornecidos. Uma possibilidade é fazer

$$f^* = \arg\min_{f \in F} \left[\sum_{i=1}^{k} |f(p_i) - Y_i|^q \right]^{\frac{1}{q}} \qquad q \in \mathbb{R} \mid q \geq 1.$$

O caso $q = 2$ corresponde ao popular ajuste por mínimos quadrados. Outra possibilidade é

$$d(f, Y) = \max_{1 \leq i \leq k} |f(p_i) - Y_i|,$$

[13] Amostras são também conhecidas como conjunto de treinamento ou dados experimentais.

e ainda outra é

$$d(f, Y) = med\{|f(p_1) - Y_1|, |f(p_2) - Y_2|, \ldots, |f(p_k) - Y_k|\},$$

onde $med\{\}$ denota a mediana do conjunto.

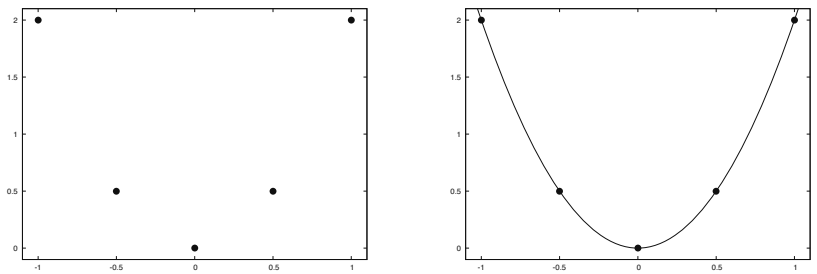
Exemplo de regressão simbólica

Como exemplo, suponha que se deseja descobrir qual a melhor função que se ajusta aos pontos (x_i, y_i) mostrados na Tabela 4.2.

x_i	y_i
-1.0	2.0
-0.5	0.5
0.0	0.0
0.5	0.5
1.0	2.0

Tabela 4.2: Pontos discretos da função exemplo.

São representadas graficamente na Figura 4.10 as amostras cujas coordenadas encontram-se na Tabela 4.2. Na Figura 4.10a são exibidos os pontos discretos, enquanto que a Figura 4.10b ilustra o ajuste exato desses pontos por meio da função $f(x) = 2x^2$.



(a) Pontos discretos

(b) Curva baseada na forma exata da função $f(x) = 2x^2$

Figura 4.10: Exposição gráfica dos pontos e da função exata com base nas coordenadas da Tabela 4.2.

Métodos Clássicos *versus* Programação Genética

Há na literatura várias técnicas de ajustes de funções, como a regressão linear, quadrática e polinomial, porém, estas limitam-se ao ajuste de coeficientes. Dessa forma, requerem um modelo de expressão definido, sendo fortemente dependentes da classe das funções, e por isso são específicas e pouco flexíveis. Isto é, considerando-se o exemplo em questão, seria fornecido ao método a forma pré-definida $f(x) = cx^2$ e este ajustaria apenas o coeficiente real c. Ou ainda, em um ajuste mais complexo, seria provido o modelo pré-estabelecido $f(x) = c_1 x^{c_2}$ e então o método, além do ajuste do coeficiente c_1, também seria responsável por ajustar o expoente c_2.

Muitas vezes, entretanto, não é suficiente ou desejável apenas a regressão linear, quadrática ou mesmo polinomial. Busca-se em muitos casos regredir, também, funções não lineares e sem qualquer forma ou complexidade pré-definida. Em outras palavras, deseja-se identificar simultaneamente a *estrutura* e os *coeficientes* da função, o que torna a tarefa vastamente mais árdua devido ao aumento

considerável do espaço de busca. A programação genética é capaz de gerenciar grandes e intricados espaços de busca, e não requer uma estrutura de solução definida, sendo portanto adequada a problemas de regressão simbólica de complexidade arbitrária.

Execução hipotética passo-a-passo

Esta seção apresenta uma execução hipotética passo-a-passo, extensamente simplificada, da programação genética no problema de regressão simbólica da função algébrica que se ajusta aos pontos da Tabela 4.2. Em outras palavras, o objetivo é evoluir a função $f(x) = 2x^2$ ou alguma identidade desta.

Não são relevantes neste exemplo didático alguns parâmetros e definições, são eles: o número máximo de gerações, a função específica de avaliação dos indivíduos, o modo de criação da população, o esquema e pressão de seleção, as probabilidades de aplicação dos operadores genéticos, e os limites de tamanho. Os parâmetros que possuem alguma relevância neste exemplo, no entanto, são listados na Tabela 4.3, onde é possível notar a simplicidade dos valores a fim de manter o problema ilustrativo e gerenciável. Foi escolhido para o conjunto primitivo de funções as operações aritméticas básicas, enquanto que o conjunto de terminais possui apenas a constante numérica 1 e, naturalmente, a variável X.

Parâmetro	**Valor**
Tamanho da população	4 indivíduos
Tipo de reprodução	geracional
Tamanho da elite	0 (sem elitismo)
Operadores genéticos	cruzamento e mutação padrão
Conjunto de funções	$\mathbb{F} = \{+, -, \times, \div\}$
Conjunto de terminais	$\mathbb{T} = \{1, X\}$
Critério de parada	discrepância nula (ajuste perfeito)

Tabela 4.3: Parâmetros do exemplo hipotético.

Sejam os quatro programas **A**, **B**, **C** e **D**, codificados por árvore, exibidos na Figura 4.11. Eles representam a população inicial, e portanto foram criados aleatoriamente por um método qualquer de criação. Imediatamente abaixo de cada um dos indivíduos é possível visualizar graficamente quão próximos seus ajustes estão da curva alvo representada pela função $f(x) = 2x^2$; isto fornece uma avaliação visual da qualidade do indivíduo.[14] Pode-se notar que os indivíduos **A** e **B** codificam a mesma função $f(X) = X$, embora o indivíduo **B** seja menos parcimonioso—o que não é necessariamente ruim, pois o material genético "redundante" pode ser útil ao processo evolucionário. Curiosamente, o indivíduo **C** não emprega a variável X, que representa a abscissa dos pontos do problema, ou seja, retorna sempre uma constante (no caso $1 + 1 = 2$) seja qual for o ponto; dessa forma, é muito pouco provável que estruturas como estas tenham prosperidade ao longo do processo evolutivo, exceto talvez como formas intermediárias ("blocos de construção"). Em termos de discrepâncias, percebe-se que os ajustes de **A**, **B**, e **C** são razoáveis, enquanto que o ajuste de **D** é nitidamente inferior aos dos demais.

Com base na qualidade dos ajustes, suponha que os indivíduos **B** e **C**, que pode-se dizer são relativamente bem adaptados, foram selecionados para cruzarem e assim deixar descendentes; processo este ilustrado na Figura 4.12a, onde os nós escurecidos denotam os pontos de cruzamento sorteados[15].

[14] Naturalmente, em problemas reais de regressão simbólica não se conhece a função que se deseja regredir—do contrário o problema já estaria resolvido—, e portanto a avaliação de um indivíduo ficaria restrita à medição da discrepância nos pontos fornecidos.

[15] Um dos pontos aleatoriamente escolhido foi o nó raiz do indivíduo **C**. Neste caso, a árvore inteira substitui o nó (ou sub-árvore, se fosse um nó interno) selecionado do indivíduo **B**, enquanto que o descendente **C'** será o próprio nó (ou

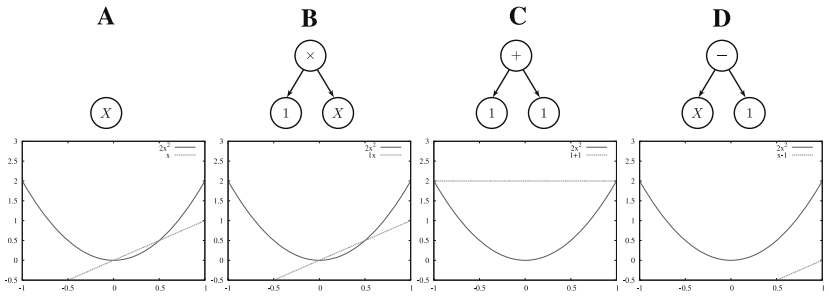
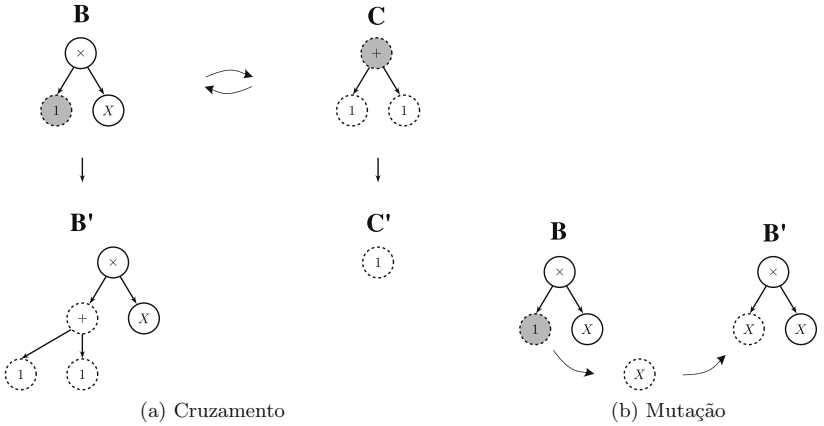


Figura 4.11: População inicial.



(a) Cruzamento (b) Mutação

Figura 4.12: Aplicação de operadores genéticos sobre a população inicial.

Esta recombinação deu origem aos indivíduos **B'** e **C'**, que então foram alocados na próxima geração (segunda geração, Figura 4.13) com os rótulos **A** e **B**, respectivamente.

Considere também que o indivíduo **B** da população inicial foi novamente selecionado, no entanto, desta vez ele foi submetido ao operador de mutação, como consta na Figura 4.12b, onde o primeiro operando da multiplicação (a constante 1) foi substituído pela variável X, dando origem à expressão $f(X) = X^2$. O programa derivado (**B'**) foi então alocado na geração seguinte com índice **C** (terceiro descendente gerado).

Suponha ainda que, dada a natureza probabilística de aplicação de operadores genéticos, que o indivíduo **A** ($f(X) = X$), da população inicial, tenha sido selecionado, mas ao invés de gerar uma estrutura derivada, foi simplesmente copiado sem modificação—isto é, clonado—à geração seguinte; em outras palavras, nem o cruzamento nem a mutação foram aplicados. Este indivíduo foi portanto inserido na próxima geração com rótulo **D**.

Percebe-se que, devido à baixa aptidão do indivíduo **D** da população inicial, este em nenhuma oportunidade foi escolhido pelo processo de seleção como genitor, portanto, o seu material genético não foi passado adiante—este processo de procriação a favor dos mais aptos é uma característica fundamental dos algoritmos da computação evolutiva.

Com a população da segunda geração lotada (quatro indivíduos), mostrada na Figura 4.13, o processo iterativo dá início à formação da geração seguinte, valendo-se de sucessivas aplicações dos operadores genéticos sobre os indivíduos mais bem adaptados. Nota-se pelos gráficos que os indivíduos

sub-árvore) escolhido de **B**.

da segunda geração são melhor adaptados quando comparados à geração anterior (população inicial)—esta *tendência* de aperfeiçoamento das soluções ao decorrer da evolução é uma propriedade das técnicas evolucionárias.

Dando sequência à execução hipotética, suponha agora que, da segunda geração, foram selecionados os indivíduos promissores **A** e **C** para cruzamento. Há uma boa chance de que os pontos escolhidos de cruzamento sejam os mostrados pela Figura 4.14 (nós escurecidos). Esta recombinação produziu os indivíduos **A'** e **C'**. É fácil perceber, no entanto, que o descendente **A'** codifica a solução esperada para o problema, isto é, $f(X) = (1 + 1) \times (X \times X) = 2X^2$. Este ajuste é apresentado graficamente na Figura 4.15, onde, obviamente, a curva da função alvo e da função evoluída coincidem. Como a discrepância neste caso é nula, isto é, o critério de parada foi satisfeito, o programa **A'** é retornado como solução desta execução ilustrativa.

4. Conclusões

Este capítulo apresentou a metaheurística evolucionária chamada programação genética, cuja especialidade é a otimização de estruturas funcionais capazes de realizar operações (lógicas, aritméticas, condicionais e de desvios), ou seja, programas de computador. A PG é uma técnica de otimização global que é *robusta*, *exige pouco conhecimento acerca do domínio*, produz *soluções simbólicas*, possui *paralelismo natural*, e é facilmente *extensível* e *modificável*.

O objetivo final da programação genética é a implementação plena da ambiciosa virtude que permitiria ao usuário, frente à tarefa alvo, especificar ao computador simplesmente *o que se quer* que seja feito (isto é, resolvê-la) em vez de *como* fazer (codificação manual dos passos/instruções) para solucioná-la. Naturalmente, em decorrência das limitações tecnológicas atuais e mesmo das limitações da metaheurística em si—que está em constante evolução e apresenta pontos de potencial aperfeiçoamento—, esta virtude ainda é um ideal a ser alcançado no que concerne à maioria dos problemas de interesse.

No entanto, dada a importância e apelo da tarefa de auto-programação, a PG tem atraído nos últimos anos esforços substanciais de pesquisa. Não é difícil imaginar que, ao passo que os computadores tornam-se mais acessíveis e poderosos, a PG torna-se capaz de resolver problemas mais complexos e extraordinários, por conseguinte despertando o interesse geral e inevitavelmente estimulando o desenvolvimento da área—que por sua vez expande o domínio de aplicação da metaheurística e novamente alavanca interesses e esforços de pesquisas, realimentando o círculo. A despeito deste futuro promissor, atualmente a PG já goza de um currículo expressivo de resultados obtidos que são tão bons ou melhores do que os produzidos por humanos, dentre estes descobertas patenteáveis (Poli et al., 2008a).

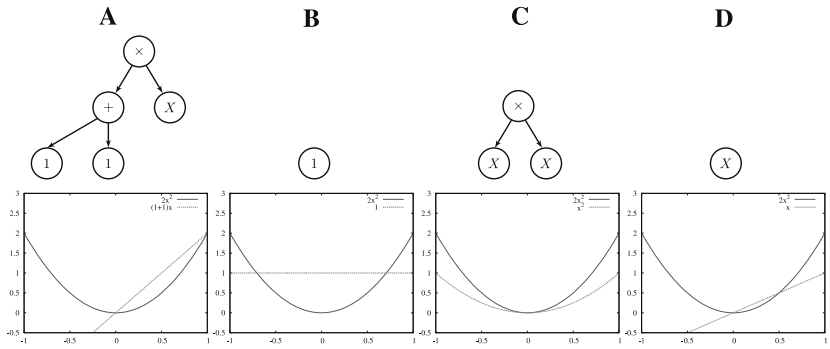


Figura 4.13: Segunda geração de indivíduos.

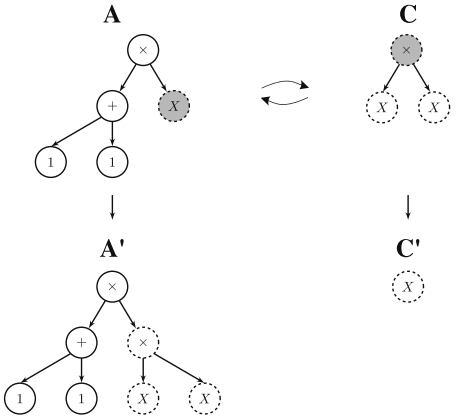


Figura 4.14: Aplicação do operador de cruzamento.

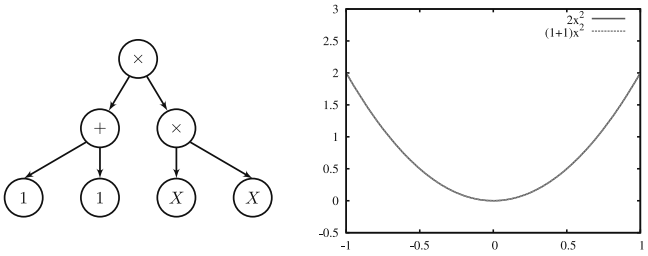


Figura 4.15: Terceira e última geração (solução encontrada).

CAPÍTULO 5

Colônia de Formigas

Priscila V. Z. C. Goliatt *Jaqueline S. Angelo* *Helio J. C. Barbosa*

Laboratório Nacional de Computação Científica, Petrópolis, RJ

Aborda-se neste capítulo uma metaherística de inspiração natural relativamente recente: a Otimização por Colônia de Formigas (em inglês *Ant Colony Optimization*, ou ACO). Para isto apresenta-se inicialmente o comportamento de uma colônia de formigas que deu origem à ACO, os principais experimentos realizados e os modelos matemáticos desenvolvidos na literatura. Antes de passar ao primeiro algoritmo da "família ACO"coloca-se o Problema do Caixeiro Viajante (PCV) – o primeiro a ser abordado pela ACO – que fornece o cenário ideal para a apresentação da técnica. Em seguida, as principais variantes da ACO são abordadas, referindo-se sempre ao PCV. Algumas aplicações da ACO em outros problemas são listadas incluindo-se possíveis extensões da técnica. O caso importante de problemas de otimização com restrições é exemplificado em um problema de otimização de estruturas de barras. Em seguida discute-se como estender a ACO ao caso de múltiplos objetivos. Finalmente, algumas ideias sobre como tratar o caso, não previsto originalmente, de variáveis de decisão contínuas são apresentadas, e conexões com outras metaheurísticas são evidenciadas.

1. Aprendendo com as Formigas Reais

Quando olhamos para uma única formiga pensamos em sua fragilidade física e em suas limitações de ações e tomadas de decisão. Entretanto, pense agora em um conjunto de formigas. Uma colônia

desses insetos sociais é capaz de organizar-se de forma a realizar tarefas de extrema complexidade.

Uma colônia de formigas é organizada em castas cujas funções variam de acordo com o tamanho do inseto (*e.g.* proteger a colônia, buscar por alimentos, transportar alimentos). A interação entre os indivíduos ocorre através de um fenômeno denominado por estigmergia [1], termo introduzido pelo zoólogo francês Pierre-Paul Grassé em 1959 (Grassé, 1959). Estigmergia refere-se a noção de que uma ação de um determinado agente deixe sinais no meio ambiente, e este sinal poderá ser percebido por outros agentes (geralmente da mesma espécie) de forma a incitar ou determinar suas ações subsequentes. Em diversas espécies de formigas esta sinalização (ou comunicação) é feita através da deposição de feromônio (ferormônio ou feromona) no meio ambiente.

O feromônio[2] é uma substância química de reconhecimento usada para a sinalização de, por exemplo, alimento, perigo e maturação sexual. No caso da busca por alimento, Goss et al. (1989) realizaram uma experiência conectando o ninho de uma colônia de formigas a uma fonte de alimento através de uma ponte dupla. Foram realizados testes com dois módulos idênticos da ponte dupla variando apenas a relação $r = \frac{l_M}{l_m}$ entre o comprimento dos braços menor (l_m) e maior (l_M) da ponte, iniciando com $r = 1$. No teste com $r = 2$, ou seja $l_M = 2l_m$ (Figura 5.1), no instante inicial da exploração a primeira formiga saía do ninho escolhendo aleatoriamente um dos braços da ponte e, ao encontrar o alimento, esta retornava ao ninho também selecionando uma rota ao acaso (Figura 5.1a). Chegando ao ninho, as demais formigas eram recrutadas para a exploração de rotas até o alimento. No início do recrutamento, as formigas permaneciam escolhendo aleatoriamente qual braço da ponte seguir (Figura 5.1b). Após certo tempo, a maioria das formigas escolhiam a menor rota e, eventualmente, algumas formigas exploravam o trajeto mais longo (Figura 5.1c).

A explicação proposta é que ao iniciar a experiência, como nenhuma formiga percorreu nenhum dos trajetos (não havendo assim nenhum feromônio depositado no meio), os dois braços da ponte possuem a mesma chance de serem percorridos. Encontrando o alimento a formiga retorna ao seu ninho depositando feromônio que, ao longo do tempo, sofre uma evaporação natural. Como no braço de menor comprimento a deposição de feromônio é rapidamente realimentada (*feedback* positivo) por um número cada vez maior de formigas, após algum tempo esta rota torna-se mais atrativa para as formigas subsequentes.

Quanto a observação de que algumas formigas eventualmente percorriam outra rota, podemos ver em Dorigo e Stützle (2004) uma discussão sobre os resultados da experiência de Goss et al. (1989) com relação ao papel do feromônio. A evaporação do feromônio ocorre muito lentamente o que significa que uma vez depositado diferentes regiões podem ser exploradas. Desta forma, possíveis trilhas subótimas podem ser preteridas pela colônia quando uma melhor rota for encontrada.

2. Construindo Formigas Artificiais

Até então sabemos que na natureza as formigas conseguem otimizar a busca por alimento de acordo a quantidade de feromônio depositada no ambiente por formigas que percorreram este caminho anteriormente. Com base nestas informações, obtidas com a experiência da ponte dupla, um modelo estocástico foi desenvolvido no intuito de tentar reproduzir a dinâmica de uma colônia de formigas em busca por alimento (Goss et al., 1989; Deneubourg et al., 1990).

Neste modelo estocástico, Φ formigas por segundo cruzam a ponte depositando uma unidade de feromônio cada. A escolha entre os braços menor (m) e maior (M) da ponte a partir de um ponto/nó $i \in \{1, 2\}$, onde $i = 1$ refere-se ao caminho/arco ninho-alimento e $i = 2$ ao caminho/arco alimento-ninho, é dada por $p_{i,m}(t)$ e $p_{i,M}(t)$. Esta decisão depende das quantidades de feromônio em cada braço, denotadas por $\phi_{i,m}(t)$ e $\phi_{i,M}(t)$. Por exemplo, desconsiderando a evaporação do feromônio, no

[1] Estigmergia: palavra proveniente do grego stigma (marca, sinal) e érgon (ação, trabalho).

[2] Feromônio: palavra proveniente do grego féro (transportar, transmitir) e órmon, particípio presente de órmao (excitar).

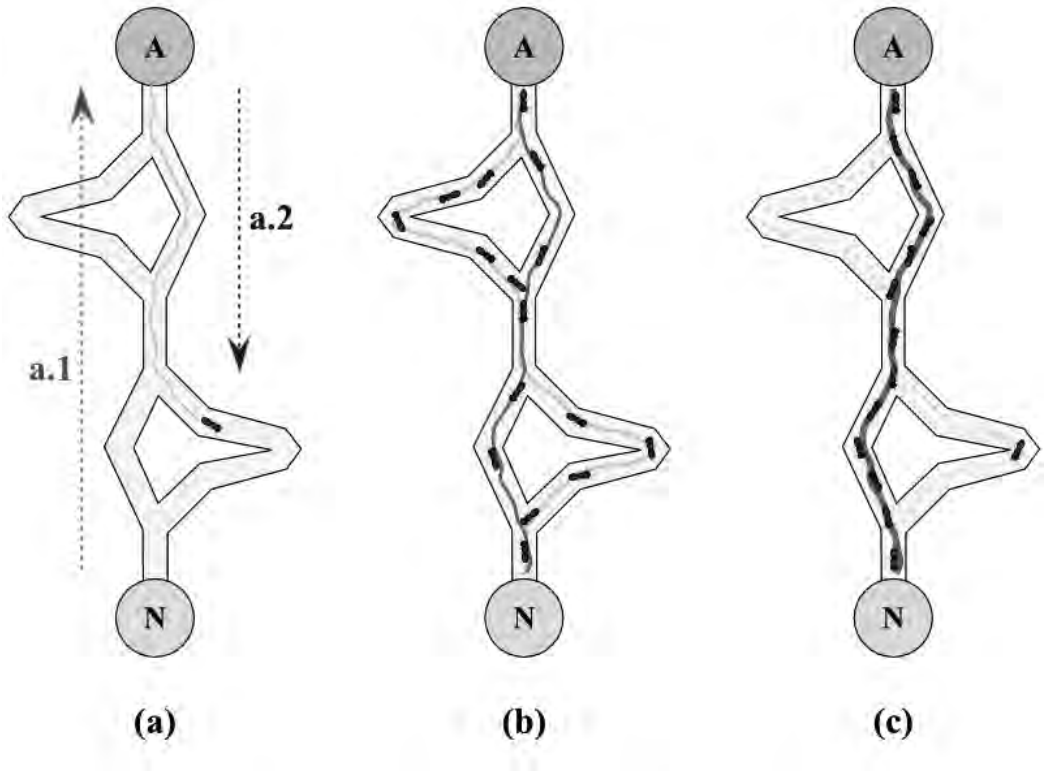


Figura 5.1: Experiência da ponte dupla com braços de tamanhos distintos. Esquema adaptado de Goss *et al* (1989). (a) Formiga inicia a exploração do meio (**a.1**) em busca por alimento. Ao encontrar o alimento (**A**), esta retorna ao ninho (**N**) depositando feromônio (**a.2**) ao longo da rota. (b) Demais formigas em **N** são recrutadas para a exploração de caminhos até **A**. (c) Após certo tempo, a maioria das formigas escolhem a menor rota (em vermelho). Eventualmente, algumas formigas são liberadas para explorar o território (linha tracejada).

instante t a probabilidade de se escolher o braço menor da ponte é dada por

$$p_{i,m}(t) = \frac{(k + \phi_{i,m}(t))^{\alpha}}{(k + \phi_{i,m}(t))^{\alpha} + (k + \phi_{i,M}(t))^{\alpha}} \qquad (p_{i,m} + p_{i,M} = 1) \tag{5.1}$$

onde os valores $k = 20$ e $\alpha = 2$ são derivados de observações experimentais e simulações computacionais, usando o método de Monte Carlo, realizados por Goss et al. (1989) e Deneubourg et al. (1990). As equações diferenciais que descrevem a dinâmica deste sistema são

$$\frac{d\phi_{i,m}}{dt} = \psi p_{i',m}(t - k) + \psi p_{i,m}(t), \qquad (i = 1, i' = 2; i = 2, i' = 1), \tag{5.2}$$

$$\frac{d\phi_{i,M}}{dt} = \psi p_{i',M}(t - kr) + \psi p_{i,M}(t), \qquad (i = 1, i' = 2; i = 2, i' = 1), \tag{5.3}$$

onde a constante $\psi = 0,5$ representa o fluxo de formigas (número de formigas por segundo). A constante k na equação (5.2) representa o tempo necessário para que as formigas atravessem o braço menor, enquanto kr na equação (5.3) expressa o mesmo mas para o braço maior. Na Figura 5.2 são apresentados os resultados da simulação de Monte Carlo, realizados por Goss et al. (1989), usando o modelo estocástico apresentado nas Equações (5.1), (5.2) e (5.3). Podemos observar que em 1.000

simulações de Monte Carlo, para $r = 1$ (braços da ponte de mesmo tamanho) os dois braços da ponte são escolhidos com igual probabilidade, enquanto para $r = 2$ (um dos braços é duas vezes maior que o outro) na maioria das simulações o braço menor foi preferencialmente escolhido.

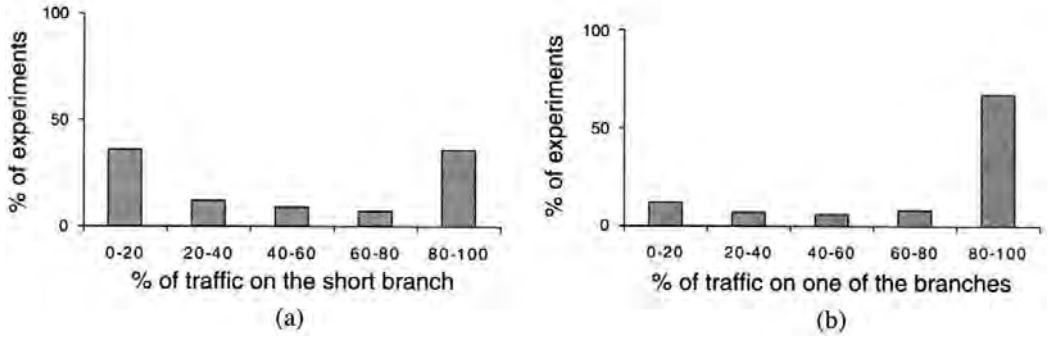


Figura 5.2: Resultado de 1.000 simulações de Monte Carlo usando o modelo estocástico dado pelas Equações (5.1), (5.2) e (5.3), para $\psi = 0,5$, $r = 1$ em (a) e $r = 2$ em (b). Figura retirada de Dorigo e Stützle (2004).

Visto que um conjunto de equações diferenciais consegue reproduzir as observações experimentais do comportamento de formigas reais em busca de alimento, usando caminhos mais curtos em um esquema de grafo simplificado (com apenas dois arcos), apresentaremos, na próxima seçao, os conceitos básicos para a construção de um algoritmo, inspirado neste comportamento das formigas reais, visando a solução de problemas de otimização que podem ser formulados em grafos/matrizes mais complexos.

3. Otimização por Colônia de Formigas

Computacionalmente, a metaheurística ACO é um método de busca construtivo no qual uma população de agentes (formigas artificiais) constroem cooperativamente soluções candidatas para um dado problema. A construção é probabilística, guiada pela heurística do problema e por uma memória compartilhada entre os agentes, contendo a experiência de iterações anteriores. Esta memória consiste de uma trilha artificial de feromônio, baseada na atribuição de pesos às variáveis das soluções candidatas. As variáveis do problema podem ser representadas por matrizes/grafos contendo os valores de sua informação heurística. Esta matriz é associada a uma matriz de taxa de feromônio como exemplificado na Figura 5.3.

Para a aplicação da ACO a um problema de otimização combinatorial, Dorigo e Stützle (2004) distinguem 6 passsos, sendo os 4 primeiros cruciais para um bom desempenho do algoritmo resultante:

1. Representar o problema usando conjuntos de componentes e transições, ou um grafo ponderado no qual as formigas construirão as soluções candidatas.

2. Definir adequadamente o que seriam as trilhas de feromônio (matriz τ) para o problema em questão.

3. Definir o que seria a informação heurística (matriz η) associada ao problema tratado, que cada formiga levaria em conta ao efetuar suas decisões.

4. Sempre que possível, implementar um algoritmo de busca local eficiente para o problema em questão.

5. Escolher uma das variantes de ACO já existentes (com ou sem modificações).

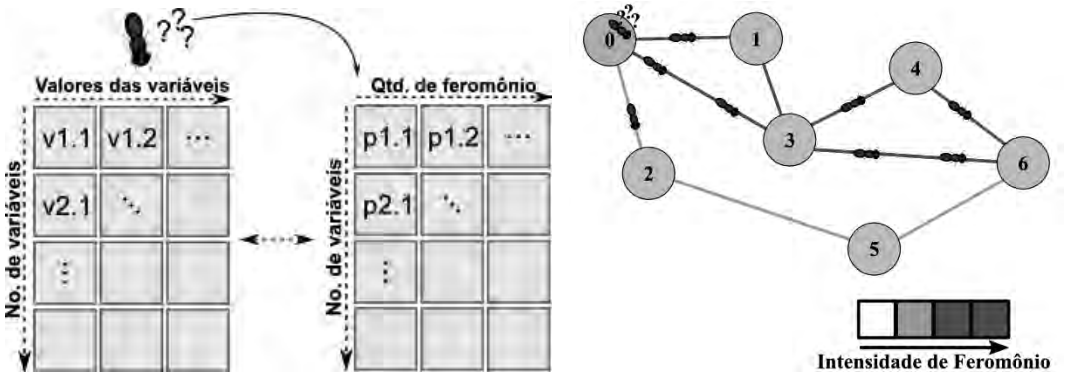


Figura 5.3: Esquema da representação e escolha das variáveis para a construção de uma rota (solução) em ACO, de acordo com a taxa de feromônio a elas associadas. Baseado em Di Caro e Dorigo (1997).

6. A partir de valores previamente utilizados em aplicações similares, ajustar os parâmetros da ACO para o problema em questão.

Evidentemente, o processo de solução de um dado problema é iterativo, e à medida que se vai ganhando conhecimento sobre o mesmo, é provável que decisões iniciais venham a ser modificadas, tendo em vista as informações obtidas no processo. Antes de abordarmos os algoritmos inspirados no comportamento de uma colônia de formigas, convém introduzir, como exemplo de aplicação, o problema do caixeiro viajante.

O Problema do Caixeiro Viajante

No Problema do Caixeiro Viajante (PCV) (em inglês *Traveling Salesman Problem (TSP)*), um vendedor, partindo de uma cidade inicial, deseja percorrer o menor caminho para atender seus clientes nas cidades vizinhas, retornando por fim à cidade de onde ele partiu, visitando cada cidade uma única vez.

A representação do PCV é feita através de um grafo completamente conectado $G = (N, A)$, onde N é o conjunto de nós (vértices), que representam as cidades, e A é o conjunto de arestas que ligam todos os nós do grafo. Cada aresta $a_{ij} \in A$ possui um peso d_{ij} pré-determinado, representando a distância entre as cidades i e j, com $i, j \in N$. A Figura 5.4 apresenta um exemplo do PCV com 4 cidades, onde a distância entre as cidades é representada por uma matriz de custo e a função objetivo é descrita em (5.4). Neste exemplo, fixando uma cidade inicial podemos gerar 6 possíveis soluções, representadas por $s_1, ..., s_6$, e o valor ótimo é alcançado pelas soluções s_3 e s_5.

O PCV pode ser simétrico ou assimétrico, onde no primeiro caso a distância entre as cidades i e j é a mesma entre j e i, ou seja $d_{ij} = d_{ji}$, e no segundo a direção utilizada para percorrer os nós é levada em consideração, desta forma pode existir pelo menos uma aresta a_{ij} tal que $d_{ij} \neq d_{ji}$. O objetivo do PCV é encontrar o menor ciclo Hamiltoneano do grafo, onde este ciclo é um caminho fechado que visita cada um dos $n = |N|$ nós [3] do grafo G apenas uma vez.

A solução ótima do PCV é portanto uma permutação π dos nós de índice $\{1, 2, ..., n\}$, tal que o tamanho $f(\pi)$ seja mínimo. A formulação do problema pode ser escrita matematicamente como:

$$\begin{aligned} &\text{minimizar } f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \\ &\text{sujeito a } \pi \in \Pi\{1, 2, ..., n\} \end{aligned} \tag{5.4}$$

[3] $|C|$, denota a cardinalidade do conjunto C, ou seja, o número de elementos do conjunto.

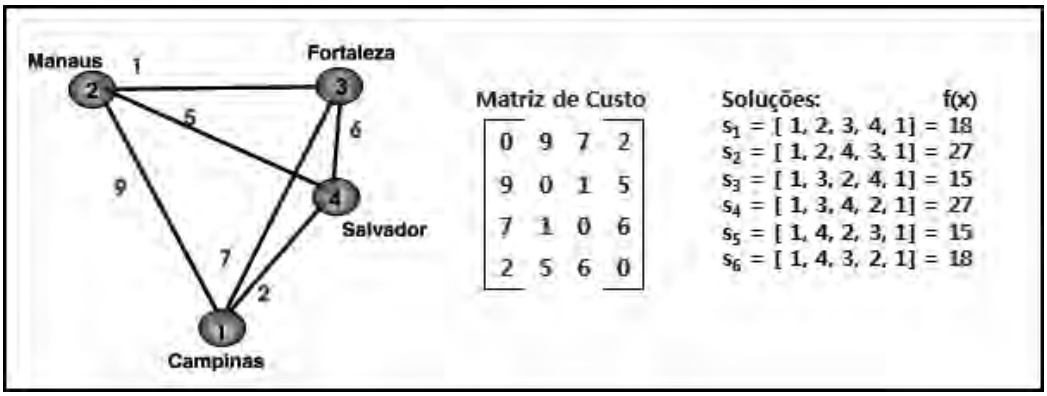


Figura 5.4: Exemplo de um PCV com 4 cidades. Figura retirada de Angelo (2008).

onde n é o número de cidades e $\Pi\{1, 2, .., n\}$ denota o conjunto de todas as possíveis permutações de $\{1, 2, .., n\}$.

ACO Aplicado ao PCV

A ACO pode ser aplicado ao PCV definindo o grafo $G = (N, A)$, onde o conjunto A conecta todos os componentes de N, que representa o conjunto de todas as cidades, e onde cada aresta tem um comprimento pré-determinado d_{ij}.

Os algoritmos ACO são essencialmente construtivos pois, a cada iteração, cada formiga (artificial), ao percorrer o grafo, constrói uma solução para o problema. Cada nó do grafo representa uma possível opção de caminho que a formiga pode percorrer. Este movimento está associado a duas formas de informação passadas para a formiga: (a) a trilha de ferômonio τ_{ij}, representada por uma matriz, associada à aresta a_{ij}, que corresponde à preferência da escolha do movimento para o nó j partindo do nó i; (b) a informação heurística, definida por $\eta_{ij} = 1/d_{ij}$, representada pela visibilidade do movimento da formiga seguir do nó i para j, que é inversamente proporcional à distância entre estas duas cidades.

O processo básico de construção da rota realizado por cada formiga é iniciado quando uma formiga artificial é posicionada, de acordo com algum critério, numa cidade inicial (ponto de partida). Em seguida, utilizando o feromônio e a informação heurística, de forma probabilística, ela constrói a rota através da inclusão sucessiva das cidades que ainda não foram visitadas. O processo termina quando a formiga, após visitar todas as cidades, retorna à cidade inicial. Depois que todas as formigas completaram seu caminho, elas adicionam uma quantidade de feromônio na rota percorrida. A estrutura de um algoritmo ACO é descrita a seguir.

Algoritmo 1 Pseudocódigo da Metaheurística ACO

```
1: Inicializar parâmetros
2: Inicializar matriz de feromônio
3: enquanto condições de parada não satisfeitas faça
4:     ConstruirSoluções
5:     AplicarBuscaLocal (opcional)
6:     AtualizarFeromônio
7: fim enquanto
```

Na rotina *ConstruirSoluções*, iniciando de um ponto de partida, as formigas se movem de acordo

com uma política de decisão que faz uso de informações locais relacionados aos nós visitados (trilha de feromônio e informação heurística). A cada passo uma componente de solução (cidade) é adicionada à rota até que todos os nós sejam visitados. Após a construção da solução e antes da atualização do feromônio, é comum a realização de busca local, através da rotina *AplicarBuscaLocal*, para melhorar a qualidade da solução encontrada. Ao final, a rotina *AtualizarFeromônio* realiza o processo de modificação da trilha (matriz) de feromônio, inspirado na deposição e/ou evaporação do feromônio nas trilhas de formigas reais.

Uma ideia muito útil para melhorar o desempenho da ACO no PCV seria a de restringir –de maneira adequada– a escolha da próxima cidade a ser visitada pela formiga. Ao invés de considerar *todas* as cidades ainda não visitadas, a formiga escolheria num subconjunto bem reduzido destas cidades.

Há pelo menos duas fortes razões para adotar listas de candidatas. Uma delas seria a redução no esforço computacional decorrente da redução do número de transições cujas probabilidades teriam que ser calculadas. É fácil ver que isto vai se tornando mais crítico à medida que o número de opções disponíveis cresce. Outra razão seria uma potencial aceleração da convergência do processo no sentido de que é de se esperar que a solução ótima (no PCV, por exemplo) conecte cada cidade sempre a alguma outra não muito distante dela. De certa forma, alguma informação heurística (a distância entre cidades vizinhas no PCV) passa a ser introduzida de maneira mais direta, além da maneira usual (probabilística) via a matriz η. Na realidade está se reduzindo drasticamente o grafo sobre o qual as formigas construirão suas rotas, focando-se a busca nas componentes mais promissoras do espaço de busca.

No caso do PCV, a informação heurística é conhecida de antemão e não se modifica durante a busca, podendo assim ser (pré-)computada uma única vez. Em outras situações, tal informação depende das componentes já presentes na solução parcial construída, e a lista de componentes candidatas deverá ser criada a cada passo.

4. Histórico dos Algoritmos ACO

Em 1991, foi desenvolvido o *Ant System* (AS), o algoritmo que deu início a metaheurística ACO. Inicialmente, o AS foi implementado como um algoritmo de otimização discreta para resolver o PCV (Dorigo et al., 1991; Dorigo, 1992).

O algoritmo AS passou por diversas modificações visando a melhoria dos resultados obtidos, a diminuição do esforço computacional e sua extensão para problemas multiobjetivo e de domínios contínuos. Os principais algoritmos ACO estão listados na Tabela 5.1 e uma breve comparação entre os algoritmos AS, RBAS, MMAS e ACS é apresentada na Tabela 5.2.

Algoritmo	Primeiras Referências
Ant System (AS)	Dorigo et al. (1991)
Elitist AS (EAS)	Dorigo (1992)
Ant-Q	Gambardella e Dorigo (1995)
Ant Colony System (ACS)	Dorigo e Gambardella (1997a,b)
Max-Min AS (MMAS)	Stützle e Hoos (1996)
Rank-Based AS (AS_{rank}/RBAS)	Bullnheimer et al. (1997, 1999)
ANTS	Maniezzo (1999); Maniezzo e Carbonaro (2000)
Best-Worst AS (BWAS)	Cordón et al. (2000)
Hyper-Cube AS (HCAS)	Blum et al. (2001)

Tabela 5.1: Principais variantes do algoritmo ACO. Baseado em Dorigo et al. (2006).

Assim como no algoritmo AS, os algoritmos RBAS e ACS inicializam a matriz de feromônio da seguinte forma:

$$\tau_{ij} = \tau_0 = \frac{k}{C^n}, \quad \forall(i,j) \tag{5.5}$$

onde τ_{ij} é a matriz de feromônio, k é o número de formigas e C^n está associada com a qualidade da solução encontrada de acordo com a heurística do problema.

No MMAS, cada valor da matriz de feromônio é limitada inferior e superiormente por τ_{min} e τ_{max} ($\tau_{min} \leq \tau_{ij} \leq \tau_{max}$) na tentativa de se evitar possíveis estagnações do algoritmo. Devido à evaporação do feromônio, o nível máximo possível para a trilha de feromônio é limitado por $\tau_{max} = 1/\rho C^*$, onde C^* é a qualidade da solução ótima de cada problema. Na prática, o MMAS usa $\tau_{max} = 1/\rho C^{bs}$, onde C^{bs} é a qualidade da melhor solução global. Desta forma, toda vez que a melhor solução global for atualizada, um novo valor de τ_{max} é definido. O valor do limite inferior é dado por $\tau_{min} = \tau_{max}/a$ onde a é um parâmetro que pode ser definido pelo usuário de acordo com o problema abordado. Maiores detalhes de como determinar o valor de τ_{min} podem ser encontrados em Stützle e Hoos (2000).

Outra importante alteração no algoritmo MMAS em relação ao AS ocorre na etapa de inicialização da matriz de feromônio. No MMAS o valor de τ_0 é estimado com base no limite superior τ_{max}. Se após algumas iterações não houver melhora na solução encontrada (ou após um número fixo de iterações), a matriz de feromônio pode ser reinicializada com base no valor de $\tau_{max} = 1/\rho C^{best}$ onde $C^{best} = C^{ib}$ representa a qualidade da melhor solução encontrado na última iteração, ou $C^{best} = C^{bs}$ representando a qualidade da melhor solução global.

Ant System (AS)	**AS$_{rank}$ ou RBAS**	**Max-Min AS (MMAS)**	**Ant Colony System (ACS)**

para cada colonia **faça**
  para cada formiga **faça**
  ConstruirSolução $^{(a)}$
  AplicarBuscaLocal $^{(c)}$
 	
  fim para
 	
  AtualizarFeromônio
fim para
	para cada colonia **faça**
	  para cada formiga **faça**
	  ConstruirSolução $^{(a)}$
	  AplicarBuscaLocal $^{(c)}$
	
	  fim para
	  OrdenarSoluções
	  AtualizarFeromônio
	fim para
		para cada colonia **faça**
		  para cada formiga **faça**
		  ConstruirSolução $^{(a)}$
		  AplicarBuscaLocal $^{(c)}$
		
		  fim para
		  OrdenarSoluções
		  AtualizarFeromônio
		fim para
			para cada colonia **faça**
			  para cada formiga **faça**
			  ConstruirSolução $^{(b)}$
			  AplicarBuscaLocal $^{(c)}$
			  AtualizarFeromônio
			  fim para
			  OrdenarSoluções
			  AtualizarFeromônio
			fim para

AtualizarFeromônio: 1. Evaporação: todas as arestas. $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in \mathcal{L},$ onde $0 < \rho \leq 1$ é a taxa de evaporação. 2. Deposição: arestas visitadas. $\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \forall (i,j) \in \mathcal{L},$ onde $\Delta\tau_{ij}^{k} = 1/C^{k}$ é a quantidade de feromônio depositada pela formiga k nas arestas que visitou.	*AtualizarFeromônio*: 1. Evaporação: idêntica ao proposto no algoritmo AS. 2. Deposição: realizada nas arestas das rotas das $w-1$ formigas melhor ordenadas (r) e nas arestas da melhor rota global (T^{bs}), sendo $\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1} (w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs},$ onde $\Delta\tau_{ij}^{r} = 1/C^{r}$ e $\Delta\tau_{ij}^{bs} = 1/C^{bs}$.	*AtualizarFeromônio*: 1. Evaporação: idêntica ao proposto no algoritmo AS. 2. Deposição: é dada por $\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best},$ onde $\Delta\tau_{ij}^{best} = \Delta\tau^{ib} = 1/C^{ib}$ representa a deposição nas arestas da melhor rota da iteração atual (T^{ib}) e $\Delta\tau_{ij}^{best} = \Delta\tau^{bs} = 1/C^{bs}$ é a deposição nas arestas da melhor rota global (T^{bs}). As duas formas podem ser usadas individualmente ou alternadamente dependendo do aspecto de convergência desejado.	*AtualizarFeromônio*: 1. Atualização Local: iterativamente assim que a aresta é visitada. $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\tau_{0},$ onde τ_0 possui os mesmos valores atribuídos na inicialização da matriz de feromônio. 2. Atualização Global: somente nas arestas pertencentes a melhor rota global T^{bs}. $\tau_{ij} \leftarrow (1-\rho_2)\tau_{ij} + \rho_2\Delta\tau_{ij}^{bs}, \forall (i,j) \in T^{bs},$ onde $0 < \rho_2 \leq 1$ é a taxa extra de evaporação e $\Delta\tau_{ij}^{bs} = 1/C^{bs}$.

(a) Nos algoritmos AS, RBAS e MMAS a rotina *ConstruirSolução* é dada por $p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in \mathcal{N}_i^k}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}}$, se $j \in \mathcal{N}_i^k$, onde α e β são parâmetros que regulam a influência da matriz de feromônio τ_{ij} e da informação heurística local $\eta_{ij} = 1/d_{ij}$, e $\mathcal{N}_i^k$ é a vizinhança factível da formiga k situada em i.

(b) No algoritmo ACS a rotina *ConstruirSolução* é dada por $j = \begin{cases} \text{argmax}_{l \in \mathcal{N}_i^k} \left\{\tau_{il}[\eta_{il}]^{\beta}\right\}, & \text{se } q < q_0; \\ J, & \text{caso contrário;} \end{cases}$ onde $q \in [0,1]$ é uma variável aleatória uniformemente distribuída, $0 \leq q_0 \leq 1$ é um parâmetro de decisão escolhido pelo usuário e $J = p_{ij}^{k}$ é dado pela equação apresentada em (a).

(c) A rotina *AplicarBuscaLocal* é opcional em todos os algoritmos.

Tabela 5.2: Comparação entre os algoritmos *Ant System*, *Rank Based AS*, *Max-Min AS* e *Ant Colony System*. Baseado em Dorigo et al. (2006).

Por tratar-se de uma metaheurística, a ACO pode ser aplicado a um grande leque de problemas. Algumas aplicações dos algoritmos derivados da ACO estão listadas na Tabela 5.3.

Tipo	Problema
Roteamentos	PCV, Rotas de Veículos, Ordenamento Sequencial.
Atribuições	QAP, Tabela Horário, Coloração de Grafos.
Agendamentos	Projetos, Total de Atrasos, Abertura de Lojas.
Subgrupos	Número Máximo de Cliques, Cardinalidade de Árvores.
Biologia Computacional	Enovelamento Protéico, Docagem Ligante-Proteína, Multi-Alinhamento de Sequências.
Outros	Problemas Multiobjetivo, Criação de Música, Otimização Estrutural, Segmentação de Imagens.

Tabela 5.3: Exemplos de aplicações dos algoritmos baseados na ACO. Baseado em Blum (2005); Dorigo et al. (2006); Capriles et al. (2007)

5. ACO Aplicada a Problemas com Restrições

No problema do caixeiro viajante, uma formiga não pode partir da cidade atual e decidir mover-se para uma cidade já visitada. Ela não está completamente livre para decidir; há restrições. No caso, ela deve formar ao final uma rota (ciclo hamiltoneano) como solução candidata. Em outras palavras, o espaço de busca está restrito às permutações da lista de cidades $\{1, 2, \ldots, n\}$. O problema de otimização é, de fato, restrito. O leitor está convidado a ler, nesta obra, o capítulo 14, que é especialmente dedicado a este tema.

Mas, no caso do PCV, há uma maneira simples de fazer com que as formigas construam sempre uma rota válida. Basta equipar cada formiga com uma memória que lista todas as cidades já visitadas por ela. Assim, a sua escolha é feita somente entre as cidades ainda não visitadas.

Mas nem sempre as restrições permitem um tratamento tão simples. Para exemplificar uma situação que requer cuidados especiais, consideraremos aqui o problema do projeto estrutural de uma estrutura de barras – denominada treliça na engenharia estrutural – muito comum na prática.

O projeto estrutural ótimo busca produzir a melhor solução estrutural no sentido de encontrar uma relação satisfatória entre custo e desempenho. Entende-se por desempenho uma série de aspectos desejáveis de um bom projeto, que vão desde objetivos funcionais (como um peso mínimo, que possibilite maximizar a carga útil num dado veículo) à beleza, no caso de uma estrutura exposta que integra um dado conjunto arquitetônico.

O projetista buscará então um conjunto de opções e valores das chamadas variáveis de projeto que minimizem (ou maximizem) uma ou mais funções objetivo satisfazendo também um conjunto de restrições sobre tais variáveis e também sobre certas medidas de desempenho da estrutura.

Na prática, as áreas das seções transversais das barras que compõem estas estruturas, exemplificadas na Figura 5.5, são muitas vezes escolhidas dentre os tamanhos disponíveis no mercado, caracterizando assim um problema de otimização discreta, onde deseja-se encontrar o conjunto de áreas $\mathbf{a} = \{A_1, A_2, ..., A_n\}$ que minimize o peso da treliça

$$w(\mathbf{a}) = \sum_{k=1}^{N} \gamma A_k L_k \tag{5.6}$$

onde γ é o peso específico do material e L_k é o comprimento da k-ésima barra.

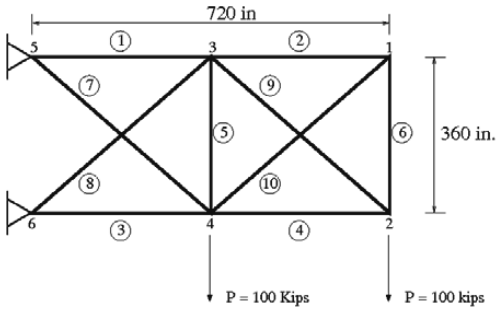
Para um desempenho aceitável, a treliça está sujeita a restrições de tensão normal nas barras (para evitar uma falha localizada)

$$\frac{|s_{j,l}|}{s_{adm}} - 1 \leq 0 \qquad 1 \leq j \leq N, \quad 1 \leq l \leq N_L \tag{5.7}$$

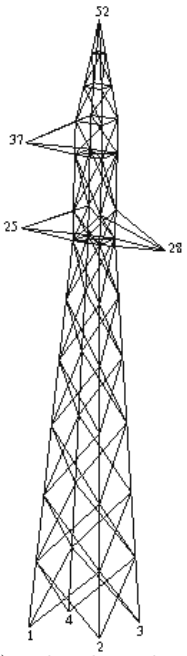
e de deslocamentos nos nós (para garantir que a sua geometria não se modifique significativamente)

$$\frac{|u_{i,l}|}{u_{adm}} - 1 \leq 0 \qquad 1 \leq i \leq M, \quad 1 \leq l \leq N_L \tag{5.8}$$

onde u_i é o deslocamento do i-ésimo grau de liberdade translacional, s_j é a tensão na j-ésima barra, s_{adm} é a tensão admissível, u_{adm} é o deslocamento máximo admissível, M é o número de graus de liberdade translacionais, N é o número total de barras e N_L é o número de casos de carregamento considerados.



(a) Treliça de 10 barras.



(b) Treliça de 160 barras.

Figura 5.5: Exemplo de treliças. (a) Estrutura de referência (*benchmark*) usada em estudos de otimização de peso de treliça. (b) Exemplo real de treliça usada como torre de transmissão. Figura retirada de Capriles et al. (2007).

Deve-se observar que as restrições acima são funções *implícitas* das variáveis de projeto **a**. De fato, é necessário resolver o problema estrutural, correspondente à solução da equação de equilíbrio discretizada

$$\mathbf{K(a)u}_l = \mathbf{f}_l \qquad 1 \leq l \leq N_L \tag{5.9}$$

onde $\mathbf{K}$ é a matriz de rigidez da estrutura construída a partir da contribuição matricial $K_j(\mathbf{a})$ de cada barra.

Para cada caso de carregamento o sistema é resolvido para o campo de deslocamentos u

$$\mathbf{u}_l = [\mathbf{K(a)}]^{-1}\,\mathbf{f}_l \tag{5.10}$$

e a tensão na j-ésima barra é calculada como $s_{j,l} = E\varepsilon(\mathbf{u}_l)$ onde E é o módulo de Young do material e ε é a deformação longitudinal unitária da barra.

Pode-se então verificar se as restrições (5.7) e (5.8) são satisfeitas para a treliça/solução candidata construída pela formiga.

Mas o que fazer se a treliça encontrada tem um bom (baixo) peso mas é inviável, isto é, viola uma ou mais das restriçoes (5.7) e (5.8)? Uma técnica simples consiste em *penalizar* tal solução: ela tem seu peso real penalizado (aumentado) via:

$$W(\mathbf{a}) = \quad w + \kappa \left(\sum_{l=1}^{N_L} \sum_{j=1}^{N} \left[\frac{|s_{j,l}|}{s_{max}} - 1 \right]_+^2 + \sum_{l=1}^{N_L} \sum_{i=1}^{M} \left[\frac{|u_{i,l}|}{u_{max}} - 1 \right]_+^2 \right) \tag{5.11}$$

onde κ é um coeficiente de penalização e $[x]_+ = x$, se $x > 0$, e 0, caso contrário.

Alternativamente, um esquema de penalização multiplicativa pode ser usado:

$$W(\mathbf{a}) = \quad w \left(1 + \sum_{l=1}^{N_L} \sum_{j=1}^{N} \left[\frac{|s_{j,l}|}{s_{max}} - 1 \right]_+ + \sum_{l=1}^{N_L} \sum_{i=1}^{M} \left[\frac{|u_{i,l}|}{u_{max}} - 1 \right]_+ \right)^{\epsilon} \tag{5.12}$$

onde $\epsilon \geq 1$ é um expoente de penalização. Uma técnica adaptativa que não requer do usuário a definição dos parâmetros de penalização pode ser encontrada em Lemonge e Barbosa (2004), mas o leitor deve consultar o capítulo 14 sobre tratamento de restrições desta obra para se familiarizar com as várias maneiras de se lidar com este problema.

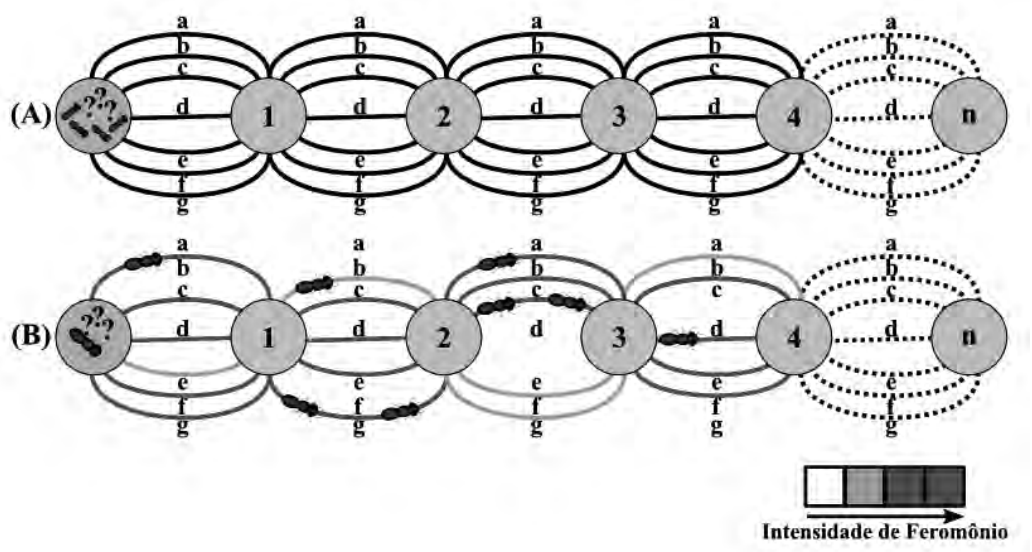


Figura 5.6: Esquema da representação e escolha das variáveis para a construção de uma rota em ACO para o problema de minimização de peso de treliças. **(A)** Para cada uma das n variáveis, cada formiga pode escolher um conjunto de áreas $A = \{\text{a,b,}\ldots\text{,g}\}$ com igual probabilidade. **(B)** Após certo tempo, algumas áreas terão maior probabilidade de serem escolhidas de acordo com a quantidade de feromônio associada.

A aplicação da ACO em problemas de otimização de peso de treliças pode ser observado em Capriles et al. (2007), onde para cada variável (barra) $k = 1 \ldots, N$, cada formiga pode escolher dentro de um conjunto de áreas A. Inicialmente, todas as áreas possuem igual probabilidade de escolha mas, após certo tempo, algumas áreas terão maior probabilidade de serem escolhidas de acordo com a quantidade de feromônio associada (esquema apresentado na Figura5.6).

Os autores fizeram testes em treliças com 10, 25, 52, 120 e 160 barras comparando variações (propostas pelos autores) do algoritmo RBAS, da seguinte forma: (i) variando os valores dos parâmetros do algoritmo RBAS (e RBAS$^+$) e considerando um esquema de penalização aditiva como apresentado na Equação 5.11; (ii) implementando uma etapa de atualização local da matriz de feromônio (RBAS$_{LU}$) e aplicando o esquema de penalização multiplicativa (Equação 5.12); (iii) modificando o algoritmo RBAS$_{LU}$ de forma a serem executadas duas fases de penalização multiplicativa e redução do espaço de busca (RBAS$_{LU,2}$). Os resultados destes testes mostraram que o algoritmo RBAS$_{LU,2}$ foi o que apresentou, na média, os melhores resultados. Os parâmetros adotados em cada variante estão descritos na Tabela 5.4.

Parâmetros	RBAS	RBAS$^+$	RBAS$_{LU}$	RBAS$_{LU,2}$
α	0,20	1,00	1,00	1,00
β	0,43	0,20	0,20	0,20
σ	0,60–1,10	0,67	0,60	0,60
σ_2	0,10–0,80	0,33	0,10	0,10
ρ	0,92	0,50	0,50	0,50
q_0	0,60–0,70	0,60	0,60	0,60
ξ	–	–	0,67	fase1: 0,67/fase2: 0,33
κ	10^7	10^7	–	–
ϵ	–	–	1,00	fase1: 1,00/fase2: 4,00
τ_0	0.50	0.50	$\frac{1}{w_{min}}$	$\frac{1}{w_{min}}$

Tabela 5.4: Parâmetros usados pelo *Rank Based Ant System* (RBAS), e suas variantes, em problemas de otimização de peso de treliças. α e β ponderam, respectivamente, as informações de feromônio e heurística, σ e σ_2 são as taxas de deposição de feromônio, ρ e ρ_2 são as taxas de evaporação de feromônio e q_0 é o parâmetro que controla a probabilidade de escolher uma determinada variável.

Nestes testes, o algoritmo RBAS inicializa a matriz de feromônio usando um valor $\tau_0 = 0.50$ em todas as entradas da matriz, enquanto no RBAS$_{LU}$ e RBAS$_{LU,2}$, a matriz de feromônio é inicializada com um valor constante dado por $\tau_0 = [w_{min}]^{-1}$, onde w_{min} é o peso da treliça quando a menor área é atribuída a cada um dos seus membros. Convém ressaltar que, desconsideradas as restrições impostas, este é o menor peso que a estrutura pode atingir.

No algoritmo RBAS$_{LU}$, quando o peso final da Equação 5.12 não se altera por um certo número consecutivo de ciclos, considera-se que a colônia de formigas convergiu para uma solução. Neste ponto completa-se a primeira fase. Para encorajar a colônia a buscar por melhores soluções em uma segunda fase, inicia-se um procedimento de busca local na vizinhança da melhor solução encontrada na primeira fase. O espaço de busca é reduzido limitando-se o número de seções transversais disponíveis na primeira fase. Esta limitação é feita da seguinte maneira: (i) para cada variável, o limite inferior do novo espaço de busca é a área transversal imediatamente menor àquela da melhor solução da primeira fase; (ii) da mesma forma, o limite superior é a área transversal imediatamente maior àquela da melhor solução da primeira fase. Esta redução do espaço de busca permite uma busca mais refinada na vizinhança e uma possibilidade de melhorar a solução encontrada. Adicionalmente, na segunda fase do algoritmo RBAS$_{LU,2}$ o coeficiente de atualização local ξ é reduzido promovendo uma busca mais vigorosa.

Na segunda fase, o parâmetro de penalização ϵ é aumentado, o que afeta diretamente o mecanismo de construção das rotas e a atualização da matriz de feromônio. Além disso, a matriz de feromônio na segunda fase é novamente inicializada com os valores de $\tau_0 = [w_{min}]^{-1}$.

6. ACO Aplicada a Problemas Multiobjetivo

Otimização Multiobjetivo

Os Problemas de Otimização Multiobjetivo (POM) buscam encontrar soluções ótimas que apresentem a melhor relação entre diversos objetivos, onde cada um precisa ser minimizado ou maximizado. A formulação matemática de um problema multiobjetivo, pode ser escrita da seguinte forma:

$$\begin{array}{l} \text{minimizar } \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x})] \\ \text{sujeito a } \mathbf{x} = (x_1, x_2, ..., x_n) \in \mathcal{S} \end{array} \tag{5.13}$$

onde $k(\geq 2)$ é o número de funções objetivo, $\mathcal{S}$ é chamado conjunto ou região viável, $\mathbf{f}$ é o vetor das funções objetivo com $f_i : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x}$ é o vetor de decisão, e se $\mathbf{x} \in \mathcal{S}$, então $\mathbf{x}$ é uma solução viável. O conjunto $\mathcal{Z}(= f(\mathcal{S}))$ corresponde ao conjunto imagem da região viável, e cada elemento de $\mathcal{Z}$, chamado de objetivo, é escrito como $\mathbf{z} = (z_1, z_2, ..., z_k)$, onde $z_i = f_i(\mathbf{x})$ para todo $i = 1, ..., k$, são chamados valores dos objetivos.

Em geral, os objetivos são conflitantes e não-comensuráveis, não sendo possível encontrar uma única solução que otimiza todos os objetivos. Lança-se mão então do conceito de otimalidade de Pareto (vide mais detalhes no capítulo 17).

Dado $\mathbf{x} \in \mathcal{S}$ um vetor de decisão qualquer, diz-se que $\mathbf{x}$ *domina* um outro vetor de decisão $\mathbf{x}' \in \mathcal{S}$, e escreve-se $\mathbf{x} \prec \mathbf{x}'$, se e somente se

$$f_i(\mathbf{x}) \leq f_i(\mathbf{x}') \quad \forall i \in \{1, ..., k\} \wedge \exists j \in \{1, ..., k\} : f_j(\mathbf{x}) < f_j(\mathbf{x}')$$

onde k é o número de objetivos. O vetor de decisão $\mathbf{x}$ é dito *não-dominado* com relação a um conjunto $\mathcal{S}' \subseteq \mathcal{S}$, se e somente se, não existir nenhum vetor em $\mathcal{S}'$ que domina $\mathbf{x}$. O vetor objetivo $\mathbf{z} \in \mathbb{Z}$ é dito não-dominado, se o seu vetor de decisão $\mathbf{x}$ correspondente é não-dominado. Um vetor de decisão $\mathbf{x}$ é um ótimo de Pareto, se e somente se, $\mathbf{x}$ é não-dominado com relação a $\mathbb{S}$. O vetor de objetivos $\mathbf{z} \in \mathbb{Z}$ é dito ótimo de Pareto, se o seu vetor de decisão $\mathbf{x}$ correspondente é um ótimo de Pareto. A melhoria de um vetor ótimo de Pareto em qualquer objetivo provoca a degradação em pelo menos algum outro objetivo.

O conjunto ótimo de Pareto contém todas as soluções ótimas de Pareto e o conjunto de vetores objetivos correspondente é denominado fronteira ótima de Pareto (Zitzler et al., 2000). A Figura 5.7 apresenta o exemplo de um problema de minimização com dois objetivos e duas variáveis de decisão.

O Problema do Caixeiro Viajante Multiobjetivo

O Problema do Caixeiro Viajante Multiobjetivo (PCVMO) pode ser apresentado da seguinte forma:

$$\begin{array}{l} \text{minimizar } \mathbf{f}(\pi) = \left\{ \begin{array}{l} f_1(\pi) = \sum_{i=1}^{n-1} d^1_{\pi(i)\pi(i+1)} + d^1_{\pi(n)\pi(1)} \\ f_2(\pi) = \sum_{i=1}^{n-1} d^2_{\pi(i)\pi(i+1)} + d^2_{\pi(n)\pi(1)} \\ \vdots \\ f_k(\pi) = \sum_{i=1}^{n-1} d^k_{\pi(i)\pi(i+1)} + d^k_{\pi(n)\pi(1)} \end{array} \right\} \\ \text{sujeito a } \pi \in \Pi\{1, 2, ..., n\} \end{array} \tag{5.14}$$

Em uma instância do PCV com k-objetivos, k grafos são utilizados, e cada um deles possui um custo diferente d^k_{ij} para a mesma aresta a_{ij}. O custo pode representar distância, tempo, gastos ou outro

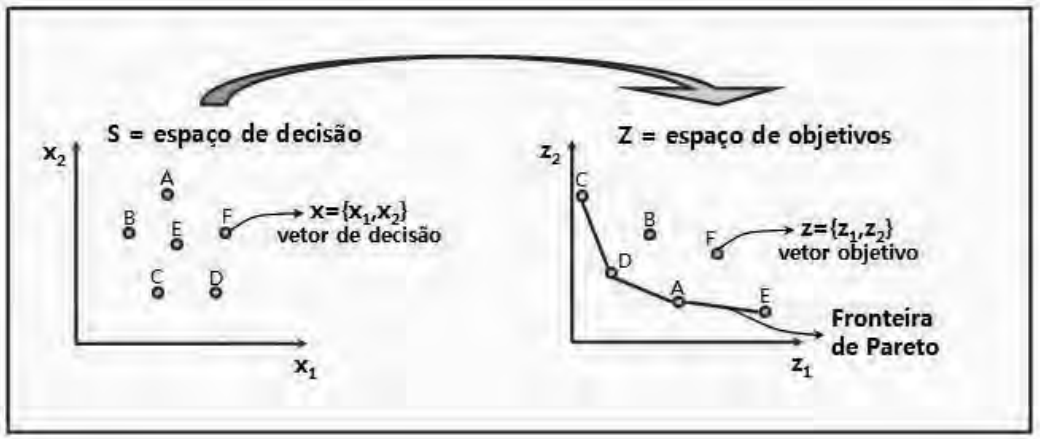


Figura 5.7: Espaço de decisão, espaço dos objetivos e a fronteira de Pareto em um problema de minimização com dois objetivos e duas variáveis de decisão, onde A, C, D e E são os vetores ótimos de Pareto ou o conjunto de soluções não-dominadas. Baseado em Zitzler et al. (2004).

parâmetro de interesse. A Figura 5.8 apresenta o mesmo exemplo da Figura 5.4, porém com dois objetivos, sendo agora as distâncias representadas por duas matrizes de custo, uma para cada objetivo e as funções objetivos são descritas em (5.14), com $k = 2$. Neste exemplo o valor ótimo no primeiro objetivo é alcançado pelas soluções s_3 e s_5 e no segundo pelas soluções s_1 e s_6.

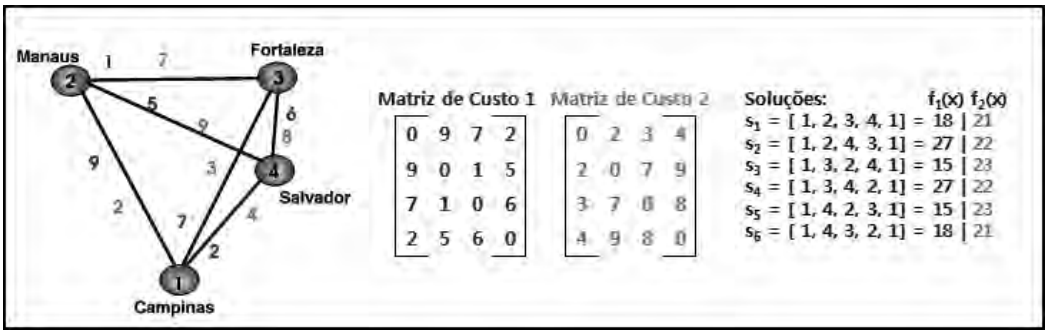


Figura 5.8: Exemplo de um PCV multiobjetivo com 4 cidades. Figura retirada de Angelo (2008).

Algoritmos ACO Multiobjetivo

García-Martínez et al. (2007) classificam os algoritmos ACO multiobjetivo (ACOMO) de acordo com a utilização de uma ou várias informações heurísticas, e a utilização de uma ou várias matrizes de feromônio, como exibido na Tabela 5.5. Em comparação aos algoritmos ACO mono-objetivo, os algoritmos ACO multiobjetivos apresentam algumas características semelhantes, porém adaptadas ao problema multiobjetivo, como por exemplo, na construção da solução e na atualização de feromônio. Para aqueles que constroem o conjunto de soluções ótimas de Pareto, esta etapa precisa ser adicionada ao algoritmo cuja estrutura básica pode ser escrita como

A seguir, os algoritmos MACS, MONACO e BicriterionAnt serão detalhados como exemplos da ACO para o caso multiobjetivo:

	Uma informação heurística	**Várias informações heurísticas**
Uma matriz de feromônio	MOACOM, MOACO-ALBP, ACOAMO.	MOAQ, MACS.
Várias matrizes de feromônio	P-ACO, MONACO.	BicriterionAnt, UnsortBicriterion, BicriterionMC, COMPETants, MACS-VRPTW, MO-PACO.

Tabela 5.5: Classificação dos algoritmos ACO multiobjetivo. Baseado em García-Martínez et al. (2007).

Algoritmo 2 Pseudocódigo da Metaheurística ACO Multiobjetivo (ACOMO)

1: Inicializar parâmetros
2: Inicializar matriz de feromônio
3: **enquanto** condições de parada não satisfeitas **faça**
4: *ConstruirSolução*
5: *AplicarBuscaLocal* (opcional)
6: *AtualizarConjuntoPareto*
7: *AtualizarFeromônio*
8: **fim enquanto**

Multiple Ant Colony System (MACS): Barán e Schaerer (2003) desenvolveram o algoritmo MACS, baseado no ACS, para a resolução de um problema de roteamento de veículos. No MACS, apenas uma colônia τ é utilizada para a geração do conjunto de Pareto, considerando três objetivos: número de veículos, tempo total da viagem e tempo total de entrega. Inicialmente, duas informações heurísticas são utilizadas, η_{ij}^0 e η_{ij}^1, referentes ao tempo total da viagem e tempo total de entrega, respectivamente. A escolha do próximo passo é feita através da seguinte regra:

$$j = \begin{cases} \arg\max_{j \in \mathcal{N}_i^h} \{\tau_{ij}[\eta_{ij}^0]^{\lambda\beta}[\eta_{ij}^1]^{(1-\lambda)\beta}\}, & \text{se } q \leq q_0 \\ \hat{i} & \text{caso contrário;} \end{cases} \tag{5.15}$$

onde β é o parâmetro que regula a influência da trilha de feromônio, λ é calculado para cada formiga h onde $\lambda = h/m$, m representa o número total de formigas e $\hat{i}$ é dado pela seguinte regra de decisão probabilística:

$$p_{ij}^h = \frac{\tau_{ij}[\eta_{ij}^0]^{\lambda\beta}[\eta_{ij}^1]^{(1-\lambda)\beta}}{\sum_{l \in \mathcal{N}_i^h} \tau_{il}[\eta_{il}^0]^{\lambda\beta}[\eta_{il}^1]^{(1-\lambda)\beta}}, \quad \text{se } j \in \mathcal{N}_i^h \tag{5.16}$$

A cada passo dado por uma formiga, a seguinte regra de atualização local de feromônio é aplicada

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\tau_0 \tag{5.17}$$

Inicialmente τ_0 é calculado como $1/\overline{f^0}.\overline{f^1}$ usando-se a média de cada uma das funções objetivo, f^0 e f^1. Após a construção de cada solução, esta é comparada com as soluções armazenadas no conjunto de Pareto. A solução que for não-dominada é incluída e a dominada é excluída do conjunto de Pareto. Ao final de cada iteração, τ_0 é recalculado de acordo com a expressão já vista. Caso $\tau_0' > \tau_0$, então a trilha de feromônio é reinicializada para o novo valor $\tau_0 \leftarrow \tau_0'$, caso contrário, a atualização global de feromônio é realizada, utilizando no cálculo cada solução S armazenada no conjunto de Pareto, de forma que

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \frac{\rho}{f^0(S)f^1(S)} \tag{5.18}$$

Multi-objective Network ACO (MONACO): O algoritmo MONACO foi apresentado por Cardoso et al. (2003), para resolver o problema dinâmico de otimização multiobjetivo do tráfego de mensagens um uma rede. Apresentaremos uma adaptação do MONACO para o problema estático, onde a disposição dos nós na rede não é modificada.

Este algoritmo utiliza uma informação heurística $\eta_{ij} = \sum_{k=1}^{K} d_{ij}^k$ e várias matrizes de ferômonio τ^k em sua formulação, onde K é o número de objetivos. Cada formiga, considerada como uma messagem, escolhe o próximo nó da rede de acordo com a seguinte regra de decisão probabilística

$$p_{ij}^h = \frac{\eta_{ij}^{\beta} \prod_{k=1}^{K} [\tau_{ij}^k]^{\alpha_k}}{\sum_{l \in \mathcal{N}_i^h} \eta_{il}^{\beta} \prod_{k=1}^{K} [\tau_{il}^k]^{\alpha_k}} \tag{5.19}$$

Nesta equação, os parâmetros α_k e β representam a importância relativa as matrizes de feromônio e a informação heurística, respectivamente. Ao final de cada iteração, as formigas que construiram sua solução, atualizam a trilha de feromônio da seguinte forma

$$\tau_{ij}^k = (1 - \rho_k)\tau_{ij}^k + \Delta\tau_{ij}^k \quad \text{com} \quad \Delta\tau_{ij}^k = Q/f^k(s_h) \tag{5.20}$$

onde ρ_k são os parâmetros relativos à taxa de evaporação para cada objetivo k, Q é uma constante relativa à quantidade de feromônio liberada pelas formigas e s_h a solução construída pela formiga h. Nesta adaptação do MONACO, as soluções não-dominadas, que formam o conjunto de Pareto, são armazenadas em um arquivo externo.

Ant Algorithm for Bicriterion Optimization (BicriterionAnt): Iredi et al. (2001) desenvolveram alguns métodos ACO para o problema de minimização do tempo total de atraso numa única máquina incluindo custos de conversão (*Single machine total tardiness problem with change over cost*). Nesta seção, apenas um dos algoritmos propostos por Irede *et al* será apresentado, aquele que considera apenas uma colônia de formigas. Este algoritmo, chamado BicriterionAnt, utiliza duas matrizes de feromônio τ e τ' e duas informações heurísticas η e η', referentes a cada um dos objetivos.

A cada iteração, cada uma das m formigas da colônia constrói uma solução, e a escolha do próximo passo utiliza a seguinte regra de decisão probabilística

$$p_{ij}^h = \frac{\tau_{ij}^{\lambda\alpha} \tau_{ij}'^{(1-\lambda)\alpha} \eta_{ij}^{\lambda\beta} \eta_{ij}'^{(1-\lambda)\beta}}{\sum_{l \in \mathcal{N}_i^h} \tau_{il}^{\lambda\alpha} \tau_{il}'^{(1-\lambda)\alpha} \eta_{il}^{\lambda\beta} \eta_{il}'^{(1-\lambda)\beta}}, \quad \text{se } j \in \mathcal{N}_i^h \tag{5.21}$$

onde α e β são os parâmetros que regulam a influência de τ e η, respectivamente, e $\mathcal{N}_i^h$ são os vizinhos viáveis (cidades ainda não visitadas) da formiga h. Para forçar que cada formiga busque por soluções em diferentes regiões na fronteira de Pareto, λ é calculado para cada formiga $h \in \{1, ..., m\}$, via

$$\lambda_h = (h - 1)/(m - 1) \tag{5.22}$$

Assim, nos casos extremos, a formiga 1, com $\lambda = 0$, considera apenas o primeiro objetivo e a formiga m, com $\lambda = 1$, considera apenas o segundo.

Após a construção das soluções, a trilha de feromônio é evaporada, aplicando a seguinte regra em todas as arestas a_{ij}

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} \quad , \quad \tau'_{ij} = (1 - \rho)\tau'_{ij} \tag{5.23}$$

onde $\rho \in (0, 1]$ representa a taxa de evaporação do feromônio.

Em seguida, as soluções não-dominadas na atual iteração são armazenadas no conjunto de Pareto e apenas estas podem atualizar a matriz de feromônio, acrescentando uma quantidade de $\tau_{ij} \leftarrow \tau_{ij} + 1/l$ na trilha, onde l é a quantidade de formigas que podem atualizar o feromônio na atual iteração.

7. ACO Aplicada a Problemas com Variáveis Contínuas

A extensão das ideias da ACO a problemas de otimização contínua exige diversas modificações, sendo poucos os trabalhos disponíveis na literatura, alguns deles brevemente comentados a seguir. Observa-se que alguns algoritmos são apenas inspirados em ACO, sendo conceitualmente diferentes do ACO original aplicado a problemas discretos. Dentre estes, as principais classes são *simulação direta* e *discretização do domínio*. Recentemente alguns artigos foram publicados focando nos conceitos principais de um algoritmo ACO, sendo classificados como *amostra probabilística*. A seguir, será apresentada uma breve descrição dos principais algoritmos destas classes.

Simulação Direta

Continuous ACO (CACO):

Foi o primeiro algoritmo proposto por Bilchev e Parmee (1995). Este algoritmo estabelece uma vizinhança ao ninho através de representação por finitos vetores direção. O procedimento de busca local é realizado usando estes vetores e usa algoritmos genéticos para executar a busca global (busca por diferentes ninhos).

Pachycondyla API*calis* (API):

Assim como no CACO, no algoritmo API (Monmarché et al., 2000) as formigas movem-se na vizinhança do ninho. Cerca de 20-30% das formigas exploram esta vizinhança de acordo com uma amplitude, enquanto as demais formigas exploram outras regiões aleatoriamente.

Continuous Interacting Ant Colony (CIAC): Proposto por Dréo e Siarry (2004), no algoritmo CIAC a interação entre os agentes ocorre de forma heterárquica onde um indivíduo influencia os demais por fluxos de informação. Esta comunicação considera o número de indivíduos em cada lado do canal de comunicação que é dividida em duas partes: (a) Trilha: distribuição pontual do feromônio. As formigas movem-se para o centro de gravidade destes pontos (semelhante ao algoritmo PSO). (b) Inter-indivíduo: comunicação P2P. Se a mensagem incluir melhora na função objetivo, o receptor migra para a região.

Discretização do Domínio

Adaptive Ant Colony Algorithm (AACA): No algoritmo AACA (Li e Wu, 2003) as soluções candidatas são representadas por vetores binários. Após completar a rota, este vetor é convertido e a atualização dos valores de feromônio é feita por um método de auto-ajuste, baseado nos valores da função objetivo.

Amostra Probabilística

Extended ACO (EACO ou ACO*): O algoritmo EACO, proposto por Socha (2004), é uma extensão direta do ACO estruturado para trabalhar com problemas mistos. As formigas usam função densidade de probabilidade (FDP) para uma escolha probabilística dos componentes da solução. O feromônio é atualizado por modificações nas FDP[4] Entretanto, por possuir somente um máximo, uma única FDP normal não é capaz de descrever uma situação onde duas áreas disjuntas em um espaço de busca são promissoras. Para resolver este problema, é usado um conjunto de FDP normais, sendo cada FDP deste conjunto definida por

$$P_{jk}(x_j) = g(x_j, \mu_k, \sigma_k) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{(x_j-\mu_k)^2}{2\sigma_k^2}} \tag{5.24}$$

[4] Principais Características da FDP: A função normal (gaussiana) é uma das funções mais aplicadas como FDP para estimar distribuições. É definida por média e desvio padrão e permite a geração de números aleatórios a partir de uma distribuição uniforme.

onde μ_k é a média e σ_k é o desvio padrão da k-ésima FDP.
Continuous Ant Colony System (CACS): É uma extensão direta da variante ACS. Assim como na EACO, no algoritmo CACS (Pourtakdoust e Nobahari, 2004) as formigas usam FDP normal para a construção de soluções candidatas. Por não usar um conjunto de FDP, não é capaz de trabalhar com múltiplos mínimos. Porém foi estruturado para trabalhar com um menor número de parâmetros.
Aggregation Pheromone System (APS): Considerado uma generalização do EACO, o APS (Tsutsui, 2004) usa FDP para agregar feromônio ao conjunto solução. As soluções são ordenadas e a os valores de feromônio são atualizados em uma elite através de uma FDP. O total de feromônio emitido pela elite é igual a um valor constante. Assim como no EACO, o APS trabalha com a decomposição de sub-FDP, porém, através de uma abordagem mais complexa (via decomposição de Cholesky).
Direct ACO (DACO): Proposto por Kong e Tian (2006), a DACO usa dois tipos de feromônio, um para a média e outro para o desvio padrão. Estes valores são usados para construir as soluções candidatas sendo cada variável do problema é associada a uma respectiva FDP. A atualização é feita considerando-se distintamente a média μ e o desvio padrão σ. Sendo a evaporação dada por

$$\vec{\mu}(t) = (1-\rho)\vec{\mu}(t-1) \qquad \vec{\sigma}(t) = (1-\rho)\vec{\sigma}(t-1) \tag{5.25}$$

onde $\rho \in [0,1]$ a taxa de evaporação, e a intensificação é dada por

$$\vec{\mu}(t) = \vec{\mu}(t) + \vec{\rho} \qquad \vec{\sigma}(t) = \vec{\sigma}(t) + \vec{\rho}|\vec{x} - \vec{\mu}(t-1)| \tag{5.26}$$

onde $\vec{x}$ é a melhor solução global.
Continuous RBAS (CRBAS):

Proposto por Capriles et al. (2006), o algoritmo CRBAS é uma extensão direta da variante *Rank Based Ant System* (RBAS) aplicado a problemas de minimização de peso de treliças. Segue a implementação e metodologia proposta pelo EACO, sendo a construção de soluções candidatas baseada em conjuntos de FDP. O processo de escolha de FDP e atualização do feromônio seguem o mesmo mecanismo apresentado pelo RBAS original, aplicado a problemas discretos.

8. Conclusões

Este capítulo apresentou a metaheurística ACO, inspirada no comportamento de uma colônia de formigas, originalmente proposta para problemas formulados em grafos. Discutiu-se sua extensão aos casos de múltiplos objetivos, problemas com restrições e variáveis contínuas. É importante lembrar que a ACO continua em desenvolvimento e sua aplicação em outras situações, como otimização em dois níveis e programação automática (*ant programming*), pode ser encontrada na literatura, bem como sua implementação em arquiteturas paralelas e distribuídas.

(Página deixada propositadamente em branco)

CAPÍTULO 6

Algoritmos Imunoinspirados

Leandro Nunes de Castro

Faculdade de Computação e Informática
Universidade Mackenzie

Quando comparada à maioria das ciências naturais e sociais, a computação é vista como uma ciência ainda jovem, possuindo algumas poucas décadas de pesquisa e desenvolvimento. Apesar disso, uma característica importante da computação é que ela não apenas envolve pesquisas científicas teóricas de base, mas também serve como ferramenta ou plataforma para a investigação e modelagem de muitas outras ciências, como a biologia, a física e a química. Dentro da grande área da computação surgiram subáreas específicas caracterizadas por uma maior proximidade com a natureza e que, em conjunto, são denominadas de *computação natural* (de Castro, 2006). Essa proximidade com a natureza pode ocorrer de diferentes maneiras, por exemplo, a natureza pode servir como fonte de inspiração para o projeto de algoritmos capazes de resolver problemas complexos; a natureza pode fornecer matéria-prima para o projeto e implementação de novas máquinas para computar; e a computação pode ser utilizada para sintetizar fenômenos naturais. Exemplos de técnicas de computação natural incluem as redes neurais artificiais, os algoritmos evolutivos, os métodos de inteligência de enxame, os sistemas imunológicos artificiais, a geometria fractal, a vida artificial, a computação molecular e a computação quântica (de Castro, 2007).

A pesquisa em computação atravessa um período de importantes avanços, mas também de grandes desafios. No cenário nacional brasileiro, foram identificados cinco Grandes Desafios da Pesquisa em Computação (http://www.sbc.or.br):

Desafio 1: Gestão da informação em grandes volumes de dados multimídia distribuídos;

Desafio 2: Modelagem computacional de sistemas complexos artificiais, naturais e socioculturais e da interação homem-natureza;

Desafio 3: Impactos para a área da computação da transição do silício para novas tecnologias;

Desafio 4: Acesso participativo e universal do cidadão brasileiro ao conhecimento;

Desafio 5: Desenvolvimento tecnológico de qualidade: sistemas disponíveis, corretos, seguros, escalados, persistentes e ubíquos.

A perspectiva da computação natural apresentada acima faz com que ela contribua de alguma forma em praticamente todos os grandes desafios, pois técnicas de computação natural podem ser aplicadas para gerenciar informação em grandes volumes de dados, modelar computacionalmente sistemas complexos, propor novas tecnologias com as quais computar e desenvolver tecnologias de qualidade. Assim, a computação natural é um exemplo de tendência da área no novo milênio.

Os sistemas imunológicos artificiais (de Castro e Timmis, 2002), tema central deste capítulo, compõem um dos muitos paradigmas da computação natural. Seu enfoque é quase sempre o desenvolvimento de algoritmos inspirados no sistema imunológico dos animais vertebrados com o objetivo final de resolver problemas complexos em áreas diversas, como navegação autônoma de robôs, controle de sistemas e processos, descoberta de conhecimentos em bases de dados, aprendizagem de máquina, vida artificial, reconhecimento de padrões e otimização.

O sistema imunológico dos vertebrados é composto por uma coleção de células, moléculas e órgãos distribuídos que, como um todo, têm o papel importante de manter um estado interno de equilíbrio dinâmico em nossos corpos. Sua complexidade foi comparada a do cérebro em muitos aspectos: sistemas imunológicos são capazes de reconhecer sinais estranhos e internos ao corpo; controlar a ação de componentes imunológicos; influenciar o comportamento de outros sistemas, como o nervoso e o endócrino; e aprender a combater agentes que causam doenças e extrair informação dos mesmos.

Este capítulo apresenta os fundamentos dos sistemas imunológicos artificiais. Ele foi concebido considerando três fontes principais de referência, a tese de doutorado de um dos autores (de Castro, 2001), o livro intitulado "Artificial Immune Systems: A New Computational Intelligence Approach" e o Capítulo 6 do livro intitulado "Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications", ambos de autoria de um dos autores (de Castro e Timmis, 2002; de Castro, 2006). No que tange a organização do capítulo, ele começa com uma revisão dos fundamentos imunológicos necessários ao desenvolvimento e compreensão de sistemas imunológicos artificiais e segue com a descrição de uma estrutura para projetar sistemas imunológicos artificiais (SIA), denominada de *Engenharia Imunológica* (de Castro, 2001). Na sequência são apresentados alguns dos algoritmos imunológicos pioneiros e um exemplo de aplicação de sistemas imunológicos artificiais a um problema de mineração de dados. O capítulo é concluído posicionando os sistemas imunológicos artificiais no contexto dos algoritmos evolutivos, tema central deste livro.

1. O Sistema Imunológico

A palavra *imunologia* é derivada do Latim *immunis* ou *immunitas*. Indivíduos que não sucumbem a uma doença quando infectados são ditos *imunes* e o status de uma resistência específica a uma determinada doença é chamado de *imunidade*. A *imunologia* pode ser entendida como o ramo da biologia responsável pelo estudo das reações de defesa que conferem resistência às doenças (Klein e Horejsi, 1990). O *sistema imunológico* é aquele responsável pela defesa do animal contra o ataque constante de *microorganismos* (Tizard, 1995).

O sistema imunológico é fundamental para a sobrevivência do animal e, por isso, precisa atuar de forma efetiva. Existe uma grande quantidade de componentes e mecanismos distintos atuando no sistema imunológico. Alguns destes elementos são otimizados para defender contra um único invasor enquanto outros são direcionados contra uma grande variedade de *agentes infecciosos*. A defesa tem

que acontecer em muitos níveis e tem que cobrir o corpo inteiro. O sistema imunológico pode ser dividido em *sistema imune inato* e *sistema imune adaptativo*, composto de diversas linhagens de células, moléculas e órgãos que trabalham em conjunto para proteger o organismo.

O sistema imune inato é a primeira linha de defesa contra vários tipos de elementos causadores de doenças (patógenos) e também é crucial para a regulação do sistema imune adaptativo. Células que pertencem ao sistema imune inato são capazes de reconhecer padrões moleculares genéricos que só estão presentes em elementos patogênicos e nunca podem ser achadas nas células do hospedeiro. Quando um elemento patogênico é reconhecido por uma célula do sistema imune inato, esta célula sinaliza outras células imunes, incluindo aquelas do sistema imune adaptativo, para começar a combater o patógeno. Assim, o sistema imune inato tem o papel de fornecer sinais coestimulatórios para o sistema adaptativo. Sinais coestimulatórios normalmente são fornecidos pelo sistema imune inato quando o organismo está sendo lesado de algum modo, como quando células estão sendo mortas por vírus. Para a maioria dos patógenos o sistema imune adaptativo não pode agir sem os sinais coestimulatórios providos pelo sistema imune inato.

Nem todos os patógenos podem ser reconhecidos e combatidos eficientemente pelo sistema inato. Alguns patógenos específicos só são reconhecidos por células e moléculas do sistema imune adaptativo. Depois que um patógeno é eliminado pelo sistema adaptativo, o sistema inato tem o papel de sinalizar o sistema adaptativo que o patógeno foi derrotado. A imunidade inata também é importante para a imunidade adaptativa porque esta última demora mais tempo para iniciar sua ação. O sistema imune adaptativo, como o próprio nome diz, é capaz de se adaptar ao patógeno e criar anticorpos específicos para combater o mesmo patógeno futuramente. Esta capacidade adaptativa de nos proteger contra patógenos conhecidos vem sendo explorada pelos imunologistas e médicos há mais de dois séculos. O princípio da vacinação é baseado nesta capacidade de aprendizagem do sistema imunológico. Inoculando um indivíduo saudável com amostras atenuadas ou mortas de patógenos faz com que o sistema imunológico gere anticorpos capazes de combater aquele patógeno específico futuramente, sem sujeitar o organismo aos sintomas desagradáveis da doença.

Fisiologia

Os tecidos e órgãos que compõem o sistema imunológico são conhecidos como *órgãos linfóides* e estão distribuídos por todo o organismo. Nos órgãos linfóides, os linfócitos interagem com diversos tipos de células, seja durante seu processo de maturação, seja durante o início de uma resposta imune adaptativa. Os órgãos linfóides podem ser divididos em *primários* (ou *centrais*), responsáveis pela produção e maturação de linfócitos, e *secundários* (ou *periféricos*) nos quais os linfócitos encontram os estímulos antigênicos, iniciando as respostas adaptativas.

Os órgãos linfóides (Figura 6.1) e suas principais funções incluem:

1. **Órgãos linfóides primários:**

   - *Medula óssea*: é o local da geração dos elementos celulares do sangue, incluindo as *hemácias* e os linfócitos;
   - *Timo*: órgão localizado na porção superior do tórax onde ocorre o desenvolvimento das células T. Algumas células migram para o timo a partir da medula óssea e lá se multiplicam e amadurecem, transformando-se em células T.

2. **Órgãos linfóides secundários:**

   - *Amígdalas* e *Adenoides*: constituem grandes agregados de células linfoides organizadas como parte do sistema imune associado a mucosas ou ao intestino;

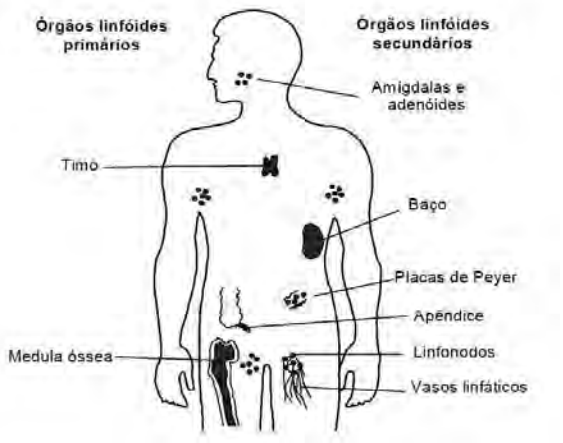


Figura 6.1: Anatomia do sistema imunológico (órgãos linfóides).

- *Linfonodos*: atuam como regiões de convergência dos vasos linfáticos e é compõem também os ambientes nos quais ocorrem a resposta imune adaptativa;
- *Apêndice* e *Placas de Peyer*: linfonodos especializados contendo células imunológicas destinadas a proteção do sistema gastrointestinal;
- *Baço*: é o único órgão linfoide entreposto na corrente sanguínea constituindo-se, portanto, no local onde os linfócitos combatem os organismos que invadem a corrente sanguínea;
- *Vasos linfáticos*: rede de canais que transporta a linfa para o sangue e órgãos linfoides.

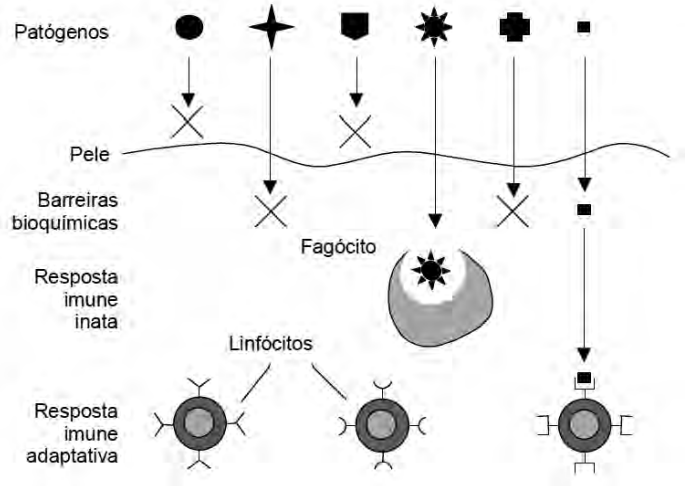


Figura 6.2: Estrutura multicamadas do sistema imunológico.

O sistema imunológico possui uma arquitetura de múltiplas camadas, com mecanismos de regulação e defesa espalhados em vários níveis (Figura 6.2). Além do sistema imune inato e do adaptativo, brevemente discutidos acima, as camadas de proteção podem ser divididas em *barreiras físicas* (pele,

pêlos, membranas e mucosas), e *barreiras bioquímicas* (saliva, suor, lágrimas, ácidos estomacais, pH, temperatura, etc.) (Janeway et al., 2000; Rensberger, 1996; Hofmeyr e Forrest, 2000; Hofmeyr, 1997).

Resposta Imunológica Adaptativa

Junto com a teoria de *maturação de afinidade* de anticorpos (Nossal, 1993; Storb, 1998), a *seleção clonal* forma o centro de uma resposta imune adaptativa e ambos foram usados na literatura de sistemas imunológicos artificiais para projetar sistemas adaptativos para solução de problemas. De acordo com a teoria de seleção clonal, o sistema imunológico é composto por conjuntos de células e moléculas discretas que permanecem em repouso até que um agente patogênico invada o organismo (Burnet, 1959). Depois da invasão, algum subconjunto destas células imunes, células B ou células T, é capaz de reconhecer o antígeno invasor e ligar-se a ele. Este processo de reconhecimento estimula as células imunes capazes de reconhecer o antígeno para começar a se reproduzir. A reprodução neste caso é assexuada; um processo de divisão mitótica de células, também chamado de *clonagem* ou *expansão clonal* (Figura 6.3). Assim, um subconjunto de células (clone) capaz de reconhecer um tipo específico de antígeno é gerado.

Como em todos os eventos reprodutivos, o processo de divisão (ou reprodução) das células B pode estar sujeito a um erro (*mutação*). Uma característica particular do sistema imunológico é que a mutação acontece com taxas inversamente proporcionais à afinidade da célula com o antígeno. Isto é muito interessante, pois sugere que a mutação no sistema imunológico é regulada pelo grau de reconhecimento de um antígeno por um anticorpo (Allen et al., 1987). Além disso, há autores (Kepler e Perelson, 1993) que sugerem que taxas elevadas de mutação se intercalam com períodos de baixa taxa de mutação para permitir seleção e expansão clonal.

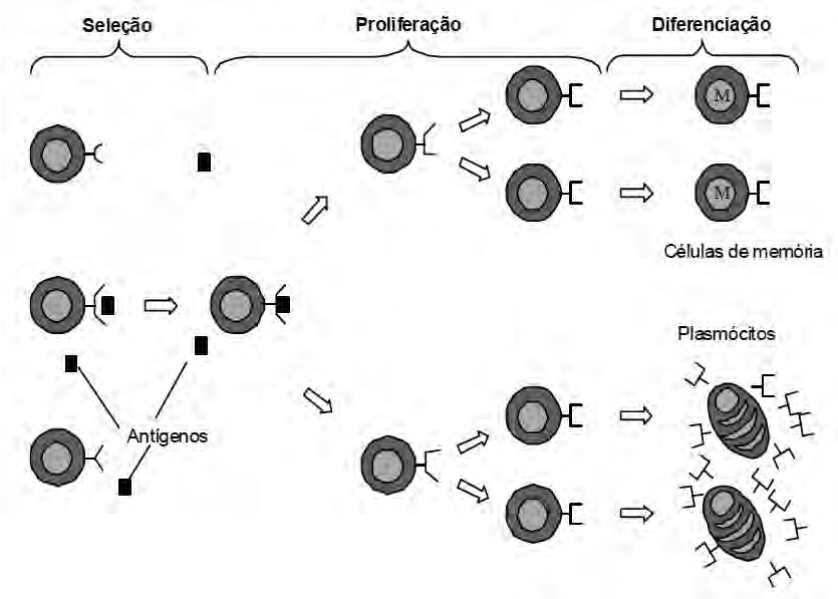


Figura 6.3: Seleção clonal, expansão e maturação de afinidade. Do repertório de células B, aquelas que apresentam afinidades mais altas com o antígeno são selecionadas e estimuladas a proliferar. Durante cada fase de proliferação, pode acontecer mutação, permitindo assim que o receptor de células se torne mais adaptado ao antígeno apresentado. Essas células imunes transformadas com afinidades mais altas com o antígeno são selecionadas para se tornar células de memória ou secretoras (plasmócitos) de anticorpos.

Adaptação por Seleção Clonal

O reconhecimento antigênico não é suficiente para a proteção do organismo, também é preciso que haja recursos suficientes para montar uma resposta imunológica efetiva contra os agentes patogênicos. O tamanho da subpopulação de linfócitos, ou seja, o *tamanho do clone* específico para o antígeno, em relação ao tamanho da população de antígenos, é crucial na determinação do resultado da infecção. A *aprendizagem imunológica* envolve o aumento do tamanho da população e afinidade Ag-Ab (antígeno-anticorpo) de linfócitos que reconheceram determinado antígeno. Como o número total de linfócitos do sistema imunológico é regulado, um aumento no tamanho de alguns clones específicos resulta na redução do tamanho de outros clones. Entretanto, o número total de linfócitos não permanece absolutamente constante (Perelson e Weisbuch, 1997).

Durante a evolução do sistema imunológico, um organismo encontra um dado antígeno repetidas vezes. Uma resposta imune adaptativa à exposição inicial de um dado antígeno é composta por um conjunto pequeno de clones de células B, cada um produzindo anticorpos de diferentes especificidades (*afinidades*). A eficiência da resposta adaptativa a encontros secundários é consideravelmente aumentada através do armazenamento de células produtoras de anticorpos com alta afinidade àquele antígeno, denominadas de *células de memória*, de forma que se tenha um grande clone inicial nos encontros subsequentes (Ada e Nossal, 1987). Ao invés de "partir do começo" toda vez que um dado estímulo antigênico é apresentado, essa estratégia garante que a velocidade e eficácia da resposta imunológica aumente após cada infecção (Perelson e Oster, 1979; Farmer et al., 1986).

Para ilustrar a maturação da resposta imunológica, considere que um antígeno Ag_1 é introduzido em um animal em um tempo 0. Poucos anticorpos específicos a Ag_1 estarão presentes no soro e, após uma fase de *latência*, os anticorpos contra o antígeno Ag_1 começam a aumentar em concentração e afinidade até um certo nível e, assim que a infecção é eliminada, sua concentração começa a cair (*resposta primária*). Quando outro antígeno Ag_2 (diferente de Ag_1) é introduzido, o mesmo padrão de resposta é apresentado, mas para um tipo de anticorpo de especificidade distinta daquela apresentada pelos anticorpos que reconheceram Ag_1, demonstrando a especificidade da resposta imune adaptativa. Por outro lado, uma característica importante da memória imunológica é sua associatividade: células B adaptadas a um certo tipo de antígeno Ag_1 apresentam uma *resposta secundária* mais rápida e eficiente não somente a Ag_1, mas também a um antígeno estruturalmente relacionado como, por exemplo, Ag_1'. Este fenômeno é chamado de *reação imunológica cruzada*, ou *resposta reativa cruzada* (*cross-reactive response*) (Hoffmann, 1986; Ada e Nossal, 1987; Sprent, 1994; Smith et al., 1997; Hodgkin, 1998; Mason, 1998). Esta memória associativa é chamada de *capacidade de generalização*, ou simplesmente generalização, na literatura de *redes neurais artificiais* (Haykin, 1999). A Figura 6.4 ilustra as respostas primária, secundária e reativa cruzada.

A aprendizagem e memória imunológica obtidas a partir da seleção clonal podem ser adquiridas através dos seguintes mecanismos (Ahmed e Sprent, 1999; Janeway et al., 2000):

Aprendizagem:

- Exposição repetida ao estímulo antigênico;
- Aumento do tamanho de clones específicos; e
- Maturação de afinidade do receptor.

Memória:

- Existência de linfócitos com períodos prolongados de vida que persistem em um estado de repouso até um segundo encontro antigênico;
- Exposição repetida ao antígeno mesmo na ausência de infecção ou em infecções crônicas; e

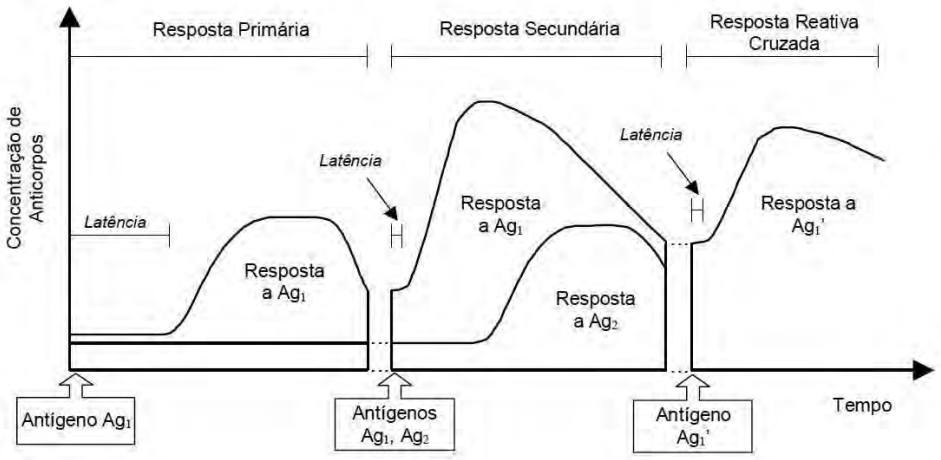


Figura 6.4: Respostas imunológicas primária, secundária e reativa cruzada. Depois que um antígeno que Ag_1 é visto uma vez (resposta primária), encontros subseqüentes com o mesmo antígeno (resposta secundária), ou um Ag'_1 semelhante (reativa cruzada), não só promoverão uma resposta mais rápida e mais efetiva a Ag_1, mas também para Ag'_1. Respostas primárias para antígenos Ag_1 e Ag_2 são qualitativamente equivalentes.

- Reatividade cruzada.

Seleção Clonal e Evolução Darwiniana

Basicamente, a teoria neo-Darwiniana de evolução declara que um procedimento evolutivo pode ser definido como aquele que envolve uma população de indivíduos capazes de se reproduzir sujeitos à variação genética seguida de seleção natural. Ao observar os princípios de seleção clonal e maturação de afinidade de uma resposta imune adaptativa é fácil notar que a seleção clonal é um tipo de processo evolutivo que ocorre em uma escala de tempo bem mais curta que a evolução das espécies. Em (de Castro e Timmis, 2002), os autores sugeriram que a seleção natural pode agir no sistema imunológico em dois níveis distintos. Primeiro, a seleção clonal e a maturação de afinidade podem ser vistas como um processo microevolutivo. Segundo, o sistema imunológico contribui de certo modo para a seleção natural, pois os indivíduos com sistema imunológico inadequado têm menor probabilidade de sobrevivência, e consequentemente de reprodução, que os indivíduos normais.

Distinção Próprio/Não-Próprio

Uma pergunta que por muito tempo intrigou os cientistas em vários campos é: como o sistema imunológico diferencia as células do organismo, conhecido como *próprio*, e os elementos estranhos ao organismo capazes de causar doenças, conhecidos como *não-próprio*? Há várias teorias que tentam chegar a uma resposta e uma delas envolve o conceito de *seleção negativa* de células T dentro do timo. Outras propostas menos ortodoxas são a idéia que o sistema imunológico evoluiu para discriminar entre não-próprio infeccioso e próprio não infeccioso (Janeway, 1992), e a teoria do perigo que sugere que o sistema imunológico é capaz de reconhecer o perigo (Matzinger, 1994).

Quando uma célula imune encontra um antígeno, várias coisas podem acontecer. Por exemplo, foi discutido que se o antígeno é não-próprio e causador de doenças, então acontecerá a expansão clonal dessas células bem sucedidas no reconhecimento e ligação ao antígeno. Porém, o reconhecimento não é imediato, um *segundo sinal*, também chamado *sinal coestimulatório*, é requerido antes que uma

resposta imune adaptativa possa ser lançada. Mas e se o antígeno é um antígeno próprio? Há algumas possibilidades, sendo uma delas a *seleção negativa* de células T dentro do timo. De forma simplificada, se um antígeno próprio é reconhecido por uma célula T imatura dentro do timo, esta célula é eliminada do repertório de células T e se torna uma célula *imunocompetente*, sendo liberada para circular pelo organismo a procura de antígenos não-próprios. Este processo, chamado *seleção negativa* de células T, só é possível porque o timo é protegido por uma barreira sanguínea que filtra qualquer molécula que não pertença ao próprio. Assim, todas as moléculas dentro do timo são moléculas próprias e as células T imaturas aprendem a serem tolerantes (a não responderem) às moléculas próprias enquanto dentro do timo.

A Teoria de Rede Imunológica

Ao invés de explicar os processos de sinalização celular e a interação de anticorpos, células e seus mecanismos efetores, a *teoria da rede imunológica* inicialmente proposta (Jerne, 1974) apresentava um novo ponto de vista sobre a atividade linfocitária, a produção de anticorpos, a seleção do repertório pré-imune, a distinção próprio/não-próprio e a tolerância imunológica, a memória e a evolução do sistema imunológico (Varela e Coutinho, 1991). Foi sugerido que *o sistema imunológico é composto por uma rede regulada de células e moléculas que se reconhecem mesmo na ausência de antígenos.* Este ponto de vista estava em conflito com a teoria da seleção clonal existente naquela época, que assumia que o sistema imunológico era composto por um conjunto discreto de clones celulares originalmente em repouso, sendo que a atividade apenas existiria quando um estímulo externo se apresentasse ao organismo.

Na nova proposta da rede imunológica, o termo 'região de ligação do anticorpo' foi mudado para *paratopo*, e 'determinante antigênico' substituído por *epítopo* . Os epítopos e os paratopos são considerados então como as duas características essenciais para o reconhecimento imunológico. Foi demonstrado experimentalmente que as moléculas de anticorpo também apresentavam epítopos, que poderiam exercer alguma funcionalidade. Um *idiotipo* foi definido como o conjunto de epítopos apresentados pelas moléculas de anticorpo e um *idiotopo* era cada epítopo idiotípico. A figura 6.5 ilustra a molécula de anticorpo com destaque para o idiotopo, o paratopo, e as respostas positiva e negativa.

(a) (b)

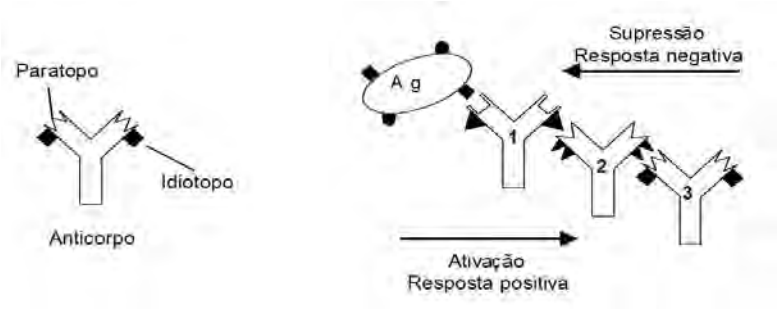


Figura 6.5: Teoria da rede imunológica. (a) Molécula de anticorpo destacando o paratopo e idiotopo. (b) Respostas positiva e negativa resultantes da interação de um paratopo com um idiotopo ou um epítopo.

O comportamento da rede imunológica, ou *rede idiotípica*, ilustrado na Figura 6.5, pode ser explicado como a seguir (Jerne, 1974). Quando um dado antígeno invade o nosso organismo, seu epítopo é reconhecido (com vários graus de especificidade) por um conjunto de diferentes paratopos, chamado p_1. Estes paratopos do conjunto p_1 estão associados a um conjunto de idiotopos i_1. O símbolo p_1i_1 denota o conjunto total de moléculas de anticorpo e linfócitos B capazes de reconhecer este antígeno. Dentro

da teoria da rede imunológica, cada paratopo do conjunto p_1 reconhece um conjunto de idiotopos, e todo o conjunto p_1 reconhece um conjunto ainda maior de idiotopos. Este conjunto i_2 de idiotopos é chamado de *imagem interna* do epítopo (ou antígeno), pois ele é reconhecido pelo mesmo conjunto p_1 que reconhece o antígeno. O conjunto i_2 está associado a um conjunto p_2 de paratopos expresso por moléculas e receptores celulares do conjunto p_2i_2. Além disso, cada idiotopo do conjunto p_1i_1 é reconhecido por um conjunto de paratopos, de forma que todo o conjunto i_1 é reconhecido por um conjunto p_3 ainda maior de paratopos que estão associados aos idiotopos i_3 de anticorpos e linfócitos pertencentes a um conjunto p_3i_3 chamado de conjunto anti-idiotípico. Seguindo este esquema, é possível chegar a conjuntos cada vez maiores de receptores que reconhecem e são reconhecidos por conjuntos previamente definidos na rede. Além do conjunto p_1i_1, existe um conjunto paralelo p_xi_1 de moléculas e receptores que apresentam idiotopos do conjunto i_1 associados a paratopos que não reconhecem o antígeno dado. As setas indicam um efeito estimulatório quando os idiotopos são reconhecidos por paratopos em receptores celulares e um efeito supressivo quando os paratopos reconhecem os idiotopos em receptores celulares.

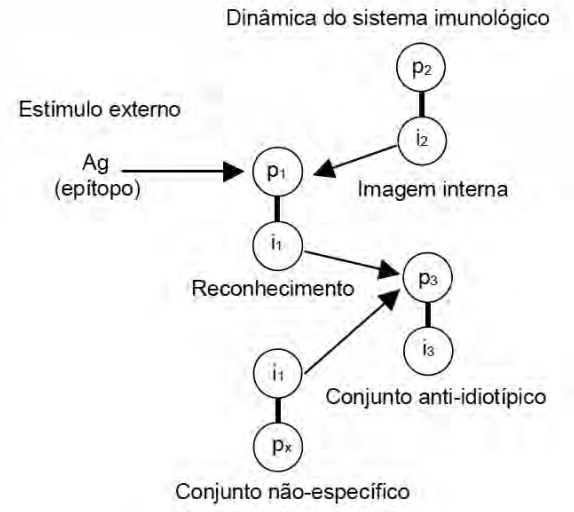


Figura 6.6: Visão detalhada da rede idiotípica (adaptado de (Jerne, 1974)).

É possível destacar três características das redes imunológicas (Varela et al., 1988; Bersini e Varela, 1990; Varela e Coutinho, 1991):

- *Estrutura*: descrição dos padrões de interconexão entre seus componentes celulares e moleculares, desconsiderando as consequências destas interações;
- *Dinâmica*: interações entre os diversos componentes do sistema;
- *Metadinâmica*: uma propriedade importante do sistema imunológico, que vai além da dinâmica de rede, é a contínua produção e morte de células e moléculas.

Em resumo, a característica central da teoria da rede imunológica é a definição da identidade molecular do indivíduo, pois a tolerância é uma propriedade global que não pode ser reduzida à existência ou à atividade de um clone específico. Ela surge a partir de uma estrutura em forma de rede que se expressa no início da evolução do sistema imunológico e é seguida pela aprendizagem ontogênica da composição molecular do ambiente no qual o sistema imunológico se desenvolve. A

organização em rede impõe um padrão de dinâmica para os anticorpos que é distinto das respostas imunológicas a antígenos externos. Estes padrões de dinâmica são perfeitamente compatíveis com a manutenção da memória que não está localizada em células de memória, mas distribuída pela rede.

Teoria do Perigo

Com um ponto de vista conceitualmente diferente, Matzinger (1994, 2002) introduziu a *teoria de perigo*. Em essência, o modelo de perigo acrescenta outra camada de células e sinais aos modelos do tipo próprio/não-próprio. Ela propõe que células apresentadoras de antígenos (APCs) são ativadas pelos alarmes causados pelos danos ao organismo no momento de uma infecção.

A teoria do perigo tenta responder a uma das perguntas fundamentais em imunologia: Como a auto-imunidade é induzida? A proposta do perigo sugere que a ação do sistema imunológico está mais relacionada a dano (prevenindo destruição) do que com o não-próprio. A teoria considera o que acontece quando os corpos mudam (por exemplo, na puberdade, gravidez, envelhecimento, etc.); por que células B e T específicas para auto-imunidade estão presentes; por que nós não fabricamos respostas imunes a vacinas; por que neonatos são facilmente tolerados; por que silicone, fragmentos de osso bem fervidos e outros elementos não promovem respostas imunes; por que não rejeitamos tumores; e assim sucessivamente (Matzinger, 1994, 2002). Uma pergunta enigmática é como distinguir entre perigoso e não-perigoso. Ao mesmo tempo há várias coisas estranhas ao organismo que são perigosas, como toxinas bacterianas, vírus, lombrigas e outros. Também há o próprio perigoso, como tumores, e o não-próprio inofensivo, como bactérias benéficas e vírus que não causam doenças.

A teoria do perigo propõe que células apresentadoras de antígeno são ativadas por sinais de perigo/alarme de células danificadas, como quando expostas a patógenos, toxinas ou sofrem danos mecânicos. Os mecanismos de ativação imune seriam uma consequência dos danos a células e tecidos. A morte da célula nem sempre é um resultado de ataque parasitário, sendo um evento normal durante o desenvolvimento embrionário, formação e morte de células hematopoiéticas, maturação e expulsão de ovócitos, etc. Em tal caso, a morte é controlada, normalmente apoptótica e células que morrem por este processo programado normal geralmente são purificadas antes de se desintegrarem. Em contraste, células que morrem por estresse ou necrose liberam os seus conteúdos nas redondezas e estes servem como sinais de perigo. O conceito-chave é que aqueles sinais de perigo não são enviados por células normais (saudáveis), mas apenas por tecidos feridos. Os sinais de perigo podem ser ativos ou passivos (Matzinger, 1994). Mudanças abruptas na condição de uma célula, como, por exemplo, variação de temperatura ou infecção, geram uma série de proteínas de choque térmico que auxiliam na sua recuperação e que servem como sinais de perigo. Moléculas internas, normalmente não secretadas, também podem servir como sinais de perigo; assim, qualquer dano celular causado por um corte, contusão e infecção pode ser percebido.

2. Engenharia Imunológica

Diferentemente dos algoritmos evolutivos, que surgiram a partir de uma ideia central principal e se desdobraram em vários ramos e variações, os sistemas imunológicos artificiais consideram múltiplas características, processos e modelos do sistema imunológico. O número de aplicações e algoritmos apresentados na literatura de SIA é vasto, mas algumas ideias essenciais foram exploradas amplamente, com destaque para a seleção clonal e maturação de afinidade, a seleção negativa e as redes imunológicas. Muitas propostas novas que envolvem conceitos sobre imunidade inata e teoria do perigo apareceram nas últimas conferências do ICARIS e na literatura científica (Timmis et al., 2003; Nicosia et al., 2004; Jacob et al., 2005; de Castro et al., 2007).

Sistemas imunológicos artificiais (SIA) podem ser definidos como "sistemas adaptativos, inspirados por imunologia teórica e funções, princípios e modelos imunológicos, que são aplicados na solução

de problemas." (de Castro e Timmis, 2002). Para projetar sistemas imunológicos artificiais, de Castro e Timmis (2002) propuseram uma abordagem em camadas baseada na *engenharia imunológica* introduzida por de Castro (2001). O termo *engenharia imunológica* se refere a "... um processo de meta-síntese, o qual vai definir a ferramenta de solução de um determinado problema baseado nas características do próprio problema e depois aplicá-la na obtenção da solução. Ao invés de buscar a reconstrução parcial ou total do sistema imunológico tão fielmente quanto possível, a engenharia imunológica deve procurar desenvolver e implementar modelos pragmáticos inspirados no sistema imunológico que preservem algumas de suas propriedades essenciais e que se mostrem passíveis de implementação computacional e eficazes no desenvolvimento de ferramentas de engenharia."(de Castro; 2001, pag. 44)

É importante salientar que não é o objetivo da engenharia imunológica estabelecer uma fronteira rígida entre os conceitos e nomenclaturas já existentes (SIA, imunocomputação, etc.) e a engenharia imunológica, mas apresentá-la como um novo paradigma computacional que não apenas utiliza os já existentes, mas que também propõe um modelo genérico de construção de ferramentas de solução de problemas. O processo de engenharia imunológica que conduz a um framework para projetar SIA está composto pelos seguintes elementos básicos (de Castro e Timmis, 2002):

- *Representação*: uma representação para os componentes do sistema.
- *Avaliação de interações*: um conjunto de mecanismos para avaliar a interação entre os indivíduos e com o ambiente.
- *Algoritmos imunológicos*: procedimentos de adaptação que governam a dinâmica e metadinâmica do sistema.

Representação

Há vários tipos de células e moléculas que compõem o sistema imunológico. O primeiro passo para projetar um sistema imunológico artificial é propor uma representação matemática ou computacional destas células e moléculas. Qualquer resposta imunológica requer o reconhecimento de um antígeno por um receptor de célula. Este reconhecimento acontece por complementaridade de forma entre duas moléculas: o receptor da célula e o antígeno. Note que alguns receptores de célula (por exemplo, anticorpos) podem ser liberados da superfície da célula e encontrados livres em solução.

Para descrever quantitativamente as interações entre as células e moléculas do sistema imunológico e os antígenos, Perelson e Oster (1979) introduziram o conceito de *espaço de formas*, S (*Shape-Space*). A idéia do espaço de formas é que o grau de *ligação* (*matching*, ou *reconhecimento*) entre moléculas depende de uma *afinidade* mútua mínima, que será representada na forma de um nível de acoplamento entre superfícies que possuam uma determinada quantidade de *regiões complementares*, como ilustrado na Figura 6.7. Suponhamos que seja possível descrever a forma de uma molécula de anticorpo, mais especificamente de sua região de ligação, por um conjunto de L parâmetros: comprimento, largura, profundidade, carga eletrostática, etc., de qualquer padrão da região de ligação. A quantidade exata de parâmetros (L) e seus respectivos valores podem ser tomados como arbitrários no desenvolvimento de ferramentas de sistemas imunológicos artificiais. Assim, um ponto em um espaço L-dimensional, denominado espaço de formas S, especifica a forma generalizada da região de ligação ao antígeno.

Se o repertório de moléculas de anticorpo é de tamanho N, então o espaço de formas para este repertório contém N pontos. É esperado que estes pontos estejam contidos em uma região finita V do espaço, uma vez que existe uma quantidade finita de larguras, comprimentos, cargas eletrostáticas, etc., que uma região de ligação pode assumir. Como as interações Ag-Ab são medidas via regiões de complementaridade, os determinantes antigênicos (*epítopos* ou *idiotopos*) também são caracterizados por formas generalizadas cujos complementos devem estar contidos dentro da mesma região V. Se as formas do paratopo e do epítopo (ou idiotopo) não forem exatamente complementares, estas moléculas ainda assim podem se ligar, porém com menor afinidade. Assume-se, então, que cada

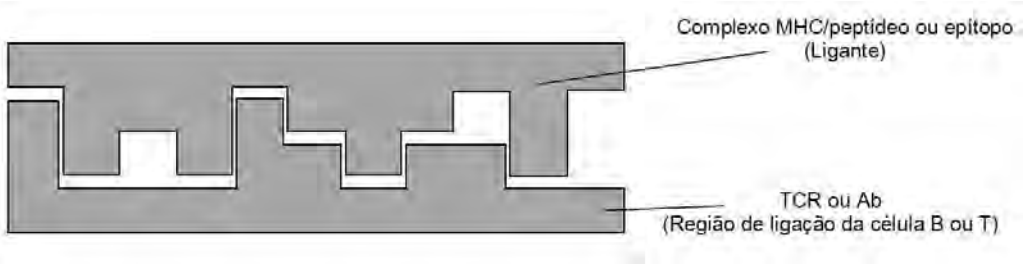


Figura 6.7: Reconhecimento via porções de regiões complementares.

paratopo interage especificamente com todos os epítopos (idiotopos) que, no espaço de formas, têm seus complementos contidos em uma região V_ε, chamada de *região de reconhecimento*, ao redor do paratopo, caracterizada pelo parâmetro ε denominado de *limiar de afinidade*. Como cada anticorpo pode reconhecer todos os epítopos dentro de uma vizinhança de reconhecimento, um número reduzido de anticorpos pode reconhecer uma quantidade bem maior de pontos na região V_ε. Este princípio está relacionado à *reatividade cruzada*, ou *resposta reativa cruzada*, discutida anteriormente, através da qual o complemento de padrões similares ocupa regiões vizinhas no espaço de formas e pode ser reconhecido pelo mesmo padrão de anticorpos, desde que um limiar de afinidade ϵ adequado seja escolhido. A Figura 6.8 ilustra o espaço de formas S, destacando os paratopos, epítopos (ou idiotopos) e o limiar de afinidade.

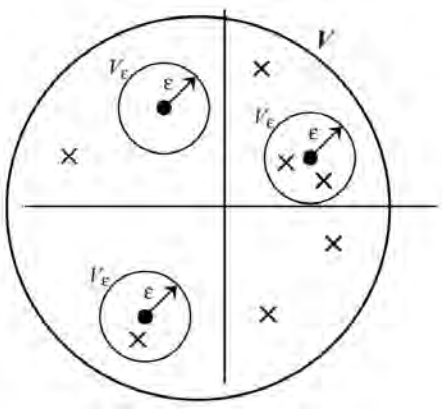


Figura 6.8: No espaço de formas S, existe uma região V na qual a forma do paratopo (•) e do complemento do epítopo (×) estão localizadas. Um anticorpo é capaz de reconhecer qualquer epítopo (ou idiotopo) cujo complemento esteja situado em uma região V_ϵ em torno do paratopo adaptado de (Perelson e Oster, 1979).

Embora o reconhecimento dentro do sistema imunológico aconteça por complementaridade de forma, a maioria dos SIA quantifica o grau de semelhança em vez de complementaridade entre estruturas de dados. O tipo de estrutura de dados mais comum é uma cadeia de atributos ou *sentença* que pode ser um vetor de valores reais, uma string de inteiros, uma string binária ou uma string simbólica. Estes resultam em espaços de forma diversos:

- *Espaço de formas real*: as sentenças são vetores de valores reais.
- *Espaço de formas inteiro*: as sentenças são inteiras.

- *Espaço de formas de Hamming*: as sentenças são construídas a partir de um alfabeto finito de comprimento k.
- *Espaço de formas simbólico*: normalmente composto por tipos diferentes de atributos, nos quais pelo menos um deles é simbólico, como 'idade', 'altura', etc.

Suponha o caso no qual qualquer molécula receptora de uma célula imune é chamada de anticorpo e é representada pelas sentenças $\mathbf{Ab} = \langle Ab_1, Ab_2, ..., Ab_L \rangle$; um antígeno é determinado por $\mathbf{Ag} = \langle Ag_1, Ag_2, ... , Ag_L \rangle$, onde as letras correspondem aos atributos da sentença e L é o tamanho das sentenças. A interação de anticorpos, ou de um anticorpo e um antígeno, pode ser avaliada por uma distância ou uma *medida de similaridade*, também chamada de medida de afinidade, entre as correspondentes sentenças, dependendo se complementaridade ou similaridade de formas são buscadas. Assim, a *medida de afinidade* traduz o nível de interação entre duas sentenças em um número real não-negativo que corresponde à *afinidade* ou *grau de reconhecimento* entre as sentenças, $S^L \times S^L \rightarrow \mathfrak{R}^+$. No caso em que o reconhecimento é proporcional à complementaridade de forma, quanto maior a distância, melhor o reconhecimento. Por outro lado, se o reconhecimento é proporcional à semelhança, quanto menor a distância, melhor o reconhecimento.

Avaliação de Interações

Assumindo os vários tipos de sentenças para representar as formas generalizadas de moléculas no sistema imunológico, cada um destes tipos exigirá uma classe particular de *medida de afinidade*. Espaços de forma de valores reais requerem medidas de afinidade que tratam de vetores de valores reais; espaços de forma de inteiros requerem medidas de afinidade que tratam de sentenças de inteiros; e assim sucessivamente. Para espaços de forma de valor real, as medidas de afinidade mais comuns são a distância Euclidiana e de Manhattan, dadas pela Equação (6.1) e Equação (6.2), respectivamente:

$$D = \sqrt{\sum_{i=1}^{L} (Ab_i - Ag_i)^2} \tag{6.1}$$

$$D = \sum_{i=1}^{L} |(Ab_i - Ag_i)| \tag{6.2}$$

Para espaços de forma de Hamming a distância de Hamming pode ser usada para avaliar a afinidade entre duas moléculas. Neste caso, as moléculas são representadas como sucessões de símbolos tirados de um alfabeto finito de comprimento k. A Equação (6.3) descreve a distância de Hamming, usada para avaliar a afinidade entre duas sentenças de comprimento L em um espaço de formas de Hamming.

$$D = \sum_{i=1}^{L} \delta_i, \text{onde } \delta_i = \begin{cases} 1 & \text{se } Ab_i \neq Ag_i \\ 0 & \text{caso contrário} \end{cases} \tag{6.3}$$

Se sentenças binárias, $k \in \{0,1\}$, forem usadas para representar as moléculas, então tem-se um *espaço de formas binário de Hamming* ou um *espaço de formas binário*. Se sentenças ternárias, i.e., $k = 3$, forem usadas para representar as moléculas, então tem-se um *espaço de formas ternário de Hamming* ou *espaço de formas ternário*; e assim por diante.

Para ilustrar o uso da distância de Hamming para avaliar afinidade, considere duas sentenças arbitrárias $\mathbf{Ag}$ = [1 0 0 0 1 1 1 0 1 0] e $\mathbf{Ab}$ = [1 0 1 0 1 0 1 0 1 0]. Executar uma combinação entre estas sentenças significa compará-las para avaliar a afinidade mútua, como ilustrado na Figura 6.9. Se a afinidade estiver sendo medida pela complementaridade entre as duas sentenças, então a afinidade delas é igual à distância de Hamming entre elas. Por outro lado, se similaridade corresponder

à afinidade, então a afinidade entre elas pode ser determinada por L - $D(\mathbf{Ag},\mathbf{Ab})$, onde $D(\mathbf{Ag},\mathbf{Ab})$ é a distância Hamming entre **Ag** e **Ab**.

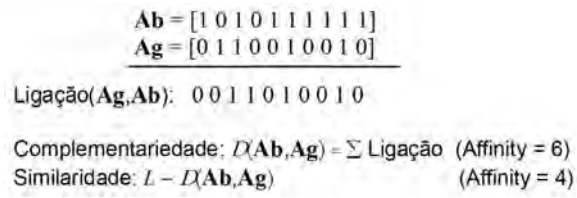


Figura 6.9: Match ou ligação entre um antígeno Ag e um anticorpo Ab, e sua afinidade.

Duas outras medidas de afinidade são amplamente usadas em espaços de formas de Hamming, a chamada regra dos *r-bits consecutivos* (*rcb*) e a regra dos *r-chunks*. No caso dos *rcb*, o número de bits consecutivos (assumindo que a similaridade entre sentenças é desejada) determina a afinidade entre duas sentenças. Para calcular a afinidade entre duas moléculas (sentenças), é feito o matching entre elas e é determinado o comprimento da maior sequência consecutiva de bits. No método dos r-chunks, somente r posições consecutivas (r-blocos ou r-chunks) são especificadas ao invés de considerar todos os L símbolos da sentença (Esponda et al., 2004). A Figura 6.10 ilustra as medidas de afinidade *r-bits consecutivos* e *r-chunks*.

Na Figura 6.10(a), se $r \leq 4$, então o antígeno é reconhecido pelo anticorpo; senão, nenhum reconhecimento acontece. O reconhecimento é determinado pelo número de bits consecutivos e, portanto, 4. Na Figura 6.10(b) um antígeno é representado por uma sentença de comprimento $L = 5$ e três blocos de comprimento $r = 3$ são utilizados. Assim, para $r = 3$ qualquer anticorpo que contém um dos blocos ilustrados reconheceria o antígeno.

(a) (b)

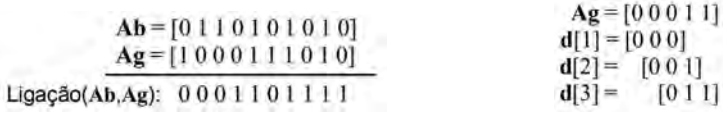


Figura 6.10: Grau de ligação entre um antígeno Ag e um anticorpo Ab.

Em espaços de forma simbólicos, as sentenças que representam os componentes do sistema imunológico artificial são compostas por pelo menos um atributo simbólico.

Um último conceito tem que ser discutido antes de proceder com os principais algoritmos imunológicos, pois a determinação da afinidade entre duas sentenças não é suficiente para indicar se duas moléculas se ligarão. Um *limiar de afinidade* é necessário para permitir que os sistemas imunológicos artificiais incorporem o conceito de reatividade cruzada. Para ilustrar o papel do limiar de reatividade cruzada quando a distância Hamming é usada para avaliar a afinidade entre as sentenças proporcionalmente ao seu grau de similaridade $L - D$ (L é o comprimento da sentença), considere o exemplo de Figura 6.11. Neste exemplo, a afinidade entre **Ab** e $\mathbf{Ag}_1$ é $aff(\mathbf{Ab},\mathbf{Ag}_1) = 6$, e a afinidade entre **Ab** e $\mathbf{Ag}_2$ é $aff(\mathbf{Ab},\mathbf{Ag}_2) = 2$. Se um limiar de afinidade $\epsilon = 6$ é usado, então o anticorpo **Ab** reconhece $\mathbf{Ag}_1$, mas não reconhece $\mathbf{Ag}_2$.

Algoritmos Imunológicos

A literatura é rica em trabalhos que usam aspectos particulares e princípios do sistema imunológico para projetar algoritmos novos ou melhorar técnicas existentes para solução de problemas. Porém,

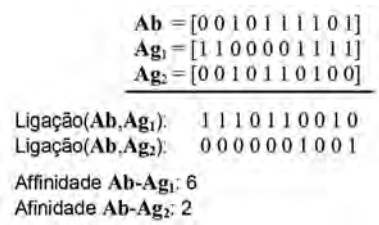


Figura 6.11: Se o limiar de reatividade cruzada é $\epsilon = 6$, então Ag_1 foi reconhecido por Ab, enquanto Ag_2 não foi.

dada uma representação satisfatória para as células e moléculas imunes, e como avaliar as interações entre elas e com o ambiente é possível identificar alguns algoritmos imunológicos de uso geral. Estes algoritmos podem ser separados em duas classes: baseados em população e baseados em rede. A primeira classe envolve todos os algoritmos que não levam em conta princípios de conectividade entre agentes e os algoritmos baseados em rede são todos aqueles inspirados pela teoria da rede imunológica. Serão revisadas cinco classes principais de algoritmos aqui:

- *Medula óssea*: gera populações de células ou moléculas imunes a serem usadas em SIA (e possivelmente em outras técnicas, como algoritmos genéticos).
- *Seleção negativa*: define um conjunto de detectores para executar, principalmente, detecção de anomalias.
- *Seleção clonal*: gera repertórios de células imunes com base na teoria da seleção clonal e maturação de afinidade.
- *Redes imunológicas contínuas*: redes imunológicas com dinâmica contínua.
- *Redes imunológicas discretas*: redes imunológicas com dinâmica discreta.

3. Algoritmos Imunológicos

Atualmente há muitos algoritmos imunológicos disponíveis na literatura, mas praticamente todos eles são baseados em um ou mais dos princípios e teorias imunológicas discutidos neste capítulo. Nesta seção será dada ênfase aos algoritmos pioneiros da área de sistemas imunológicos artificiais. Os leitores interessados no estado da arte da área podem encontrar as contribuições mais recentes na série da Springer-Verlag voltada para a publicação dos trabalhos da Conferência Internacional de Sistemas Imunológicos Artificiais (International Conference on Artificial Immune Systems – ICARIS), que ocorre anualmente (http://www.artificial-immune-systems.org).

Modelos de Medula Óssea

A medula óssea é responsável pela geração das células sanguíneas, incluindo os linfócitos. O material genético de uma molécula de anticorpo e, portanto, de uma célula B, está armazenado em cinco bibliotecas separadas e distintas. A produção de uma molécula de anticorpo se dá através da concatenação de componentes selecionados aleatoriamente a partir de cada uma das bibliotecas gênicas. O modelo computacional mais simples de medula óssea é aquele que gera sentenças, ou vetores, de comprimento L utilizando um gerador de números pseudo-aleatórios. Para o caso de espaços reais, basta determinar o intervalo de pertinência do vetor m como, por exemplo, $m \in [0, 1]^L$. No caso de espaços

de Hamming, o vetor que representa a molécula m deve ser composto por elementos pertencentes a um alfabeto finito pré-definido. Para espaços inteiros, um algoritmo de permutação de L pode ser empregado.

Os modelos mais complexos e biologicamente mais plausíveis de construção de repertórios linfocitários exigem a utilização de bibliotecas gênicas a partir das quais as células e/ou moléculas serão evoluídas ou simplesmente geradas. Hightower et al. (1995) e Perelson et al. (1996) utilizaram um algoritmo genético para estudar o efeito da evolução na codificação genética das moléculas de anticorpo. Uma característica desta codificação é que nem todos os genes existentes no *genótipo* (coleção total de genes) estão expressos no *fenótipo* (moléculas de anticorpo expressas). Neste modelo, as sentenças binárias representando o genótipo de um indivíduo foram divididas em quatro bibliotecas de segmentos gênicos para a geração de moléculas de anticorpo, como ilustrado na Figura 6.12. Cada biblioteca continha oito elementos, representados por sentenças binárias de comprimento 16, de forma que cada genoma individual possuía um total de 128 bits. Os anticorpos expressos tinham comprimento total igual a 64 ($4 \times 16 = 64$).

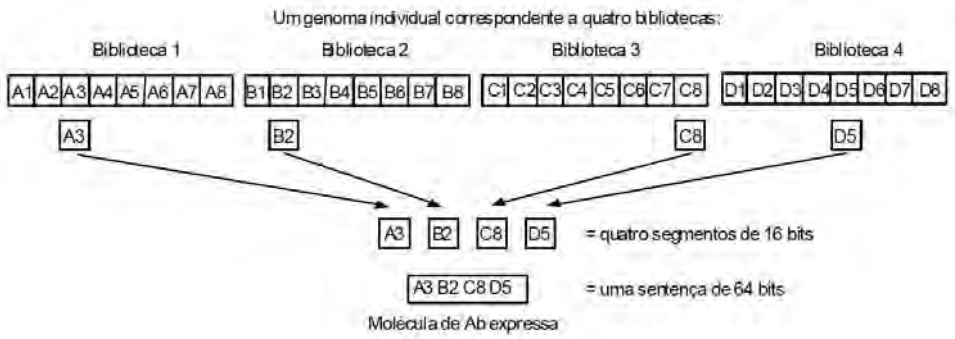


Figura 6.12: Processo de construção/expressão de um anticorpo a partir de bibliotecas genéticas.

Modelos semelhantes podem ser empregados para a simulação da medula óssea no processo de geração de receptores celulares. A quantidade de bibliotecas, o tamanho dos segmentos gênicos e o comprimento L das moléculas serão definidos pelo projetista de acordo com o problema a ser tratado. É importante ressaltar que a utilização de bibliotecas gênicas para a geração do repertório linfocitário impõe inerentemente uma determinada estrutura ao repertório, como o tipo de dado a ser utilizado (numérico, simbólico, etc.) e o intervalo de varredura (p.ex., 1-10, Segunda, Terça, ..., etc.). Um sistema imunológico contendo l bibliotecas, cada uma com c componentes, pode produzir c^l moléculas de anticorpo distintas, ou seja, o repertório potencial de anticorpos é composto por c^l moléculas.

Algoritmo de Seleção Negativa

Forrest et al. (1994) desenvolveram um algoritmo para detecção de anomalias, com aplicações em proteção computacional, baseado na seleção negativa de linfócitos T dentro do timo. O algoritmo foi denominado de *algoritmo de seleção negativa* (ASN) e é executado em duas fases, como a seguir:

1. *Censoriamento*, (a):
   - Defina o conjunto de sentenças próprias (S) que se deseja proteger;
   - Gere sentenças aleatoriamente e avalie a afinidade (*match*) entre cada uma delas e as sentenças próprias. Caso a afinidade seja superior a um determinado limiar, rejeite; caso contrário, armazene-a em um conjunto de detectores (R).

2. *Monitoramento*, (b):

   - Dado o conjunto de sentenças que se deseja proteger (sentenças protegidas), avalie a afinidade entre cada uma delas e o conjunto de detectores. Se a afinidade for superior a um limiar pré-definido, então um elemento não-próprio foi identificado.

Os autores utilizaram uma representação no espaço binário de Hamming e a regra dos r-bits consecutivos como medida de afinidade. Também foram apresentadas equações estimando a probabilidade de ocorrência de uma ligação em pelo menos r-bits consecutivos entre duas sentenças binárias aleatórias.

Note que este algoritmo é capaz de executar tarefas como reconhecimento de padrões armazenando informações sobre o conjunto complementar (não-próprio) ao conjunto dos padrões que se deseja proteger (próprio).

Algumas observações podem ser feitas em relação ao algoritmo de seleção negativa:

1. Ele é ajustável proporcionalmente à probabilidade de detecção de uma anomalia para a regra dos r-bits consecutivos;

2. O tamanho do conjunto de detectores não necessariamente cresce com o número de sentenças a serem protegidas;

3. A probabilidade de detecção de anomalias aumenta com a quantidade de detectores independentes;

4. A detecção é simétrica; e

5. Existe um custo exponencial de geração de detectores em relação ao tamanho do conjunto de sentenças a serem protegidas (próprio), pois a geração aleatória das sentenças do conjunto R_0 leva à geração repetida de diversas sentenças.

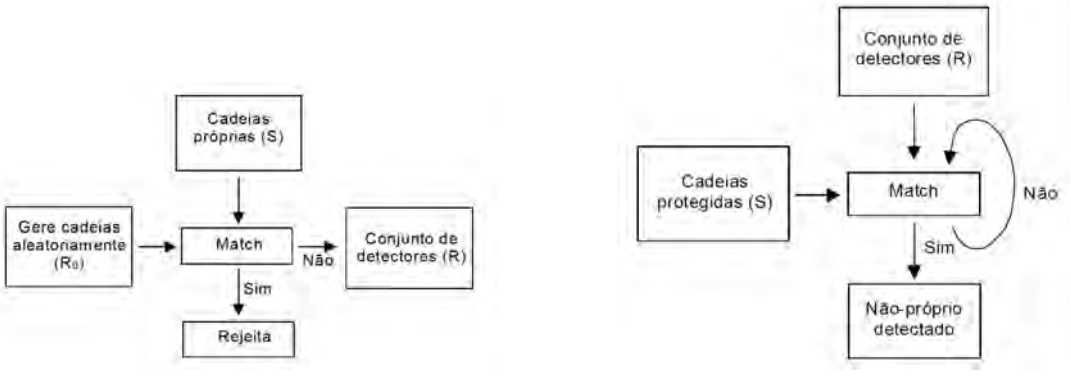


Figura 6.13: Fluxograma do algoritmo de seleção negativa. (a) Geração de um conjunto válido de detectores (censoriamento). (b) Monitoramento das sentenças protegidas, que podem ser as sentenças próprias adicionadas de outros dados que se deseja proteger.

Um Algoritmo de Seleção Clonal

O princípio de seleção clonal é usado para descrever as características básicas de uma resposta imune adaptativa. Ele propõe que só as células que reconhecem antígenos se proliferam, sendo selecionadas em detrimento daquelas que não reconhecem antígenos. As células selecionadas sofrem uma expansão clonal sujeita ao processo de maturação de afinidade. Algumas características são importantes para o desenvolvimento de um algoritmo de seleção clonal (Forrest et al., 1993; de Castro e Timmis, 2002):

- Os antígenos são tipicamente encontrados várias vezes e o sistema imunológico responde com o subconjunto de células imunes que entram em contato com o antígeno.
- Um antígeno seleciona várias células imunes para proliferar (clonagem). A taxa de proliferação de cada célula imune é proporcional à sua afinidade em relação ao antígeno selecionado: quanto maior a afinidade, maior o número de clones gerados; e vice-versa.
- A mutação sofrida por cada célula imune durante a reprodução é inversamente proporcional à afinidade do receptor da célula com o antígeno: quanto maior a afinidade, menor a taxa de mutação e vice-versa.

Forrest et al. (1993) focaram na primeira característica listada - a localidade da apresentação antigênica - e propuseram que um algoritmo genético sem crossover é um modelo razoável de seleção clonal, enquanto um algoritmo genético com crossover modela a evolução genética. Com uma visão independente, de Castro e Von Zuben (2002) focaram na afinidade proporcional à reprodução e mutação durante a seleção clonal e sugeriram que este processo é mais que um GA sem crossover, embora ele possa ser caracterizado como um processo evolutivo. Além destes foram propostos outros algoritmos de seleção clonal, como o CLIGA (Cutello, 2004) e o algoritmo de células B (Kelsey e Timmis, 2003), mas estes não serão discutidos aqui.

Os principais aspectos considerados para o desenvolvimento do algoritmo de seleção clonal (CLONALG – *clonal selection algorithm*) foram (de Castro e Von Zuben, 2002): manutenção das células de memória funcionalmente independentes do repertório, seleção e reprodução (clonagem) das células mais estimuladas, morte das células menos estimuladas, maturação de afinidade e re-seleção dos clones com maiores afinidades antigênicas, geração e manutenção de diversidade, e hipermutação proporcional à afinidade celular. Este algoritmo foi desenvolvido em duas versões: uma para o reconhecimento de padrões e aprendizagem de máquina e a outra para otimização. O funcionamento do algoritmo pode ser descrito como segue:

1. Gere um conjunto (P) de candidatos à solução, composto pelo subconjunto de células de memória (M) mais o restante ($P_{\{r\}}$) da população (P=$P_{\{r\}} \cup$ M);
2. Determine (processo de seleção) os n melhores indivíduos ($P_{\{n\}}$) da população (P), baseado em uma medida de afinidade;
3. Reproduza (processo de clonagem) estes n melhores indivíduos, gerando uma população temporária de clones (C). A quantidade de filhos de cada indivíduo é diretamente proporcional à sua afinidade;
4. Submeta a população de clones a um esquema de hipermutação em que a taxa de mutação é proporcional à afinidade do anticorpo. Uma população de anticorpos maduros é gerada (C^*);
5. Re-selecione os melhores indivíduos de C^* para compor o conjunto de memória M;
6. Substitua d anticorpos por novos indivíduos (diversidade). Os anticorpos com menores afinidades possuem maiores probabilidades de serem substituídos.

Neste algoritmo, se tomarmos n=N, ou seja, se selecionarmos todos os indivíduos da população para reprodução, cada candidato à solução será visto localmente, resultando em um algoritmo capaz de executar uma busca multimodal dentro do espaço de formas S.

As etapas 2 e 5 do CLONALG podem ser feitas de forma probabilística, ou seja, aqueles indivíduos com maiores afinidades terão maiores probabilidades de serem selecionados.

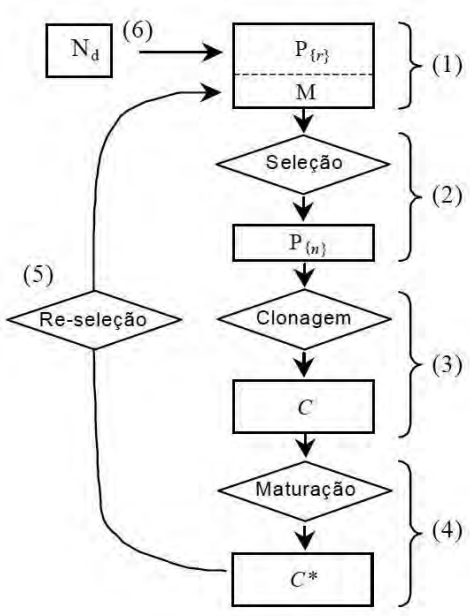


Figura 6.14: Fluxograma de blocos do algoritmo de seleção clonal, CLONALG. A numeração aponta as etapas de funcionamento do algoritmo.

Modelos de Rede Imunológica

Seguindo a proposta de rede de Jerne (1974), vários modelos de rede foram introduzidos por pesquisadores teóricos interessados em estudar e modelar o sistema imunológico. A rede imunológica é diferente da seleção clonal, pois assume que o sistema imunológico é um sistema inerentemente dinâmico. De acordo com a perspectiva de rede, células imunes e moléculas são capazes de reconhecer umas as outras e apresentar algum padrão dinâmico de comportamento, mesmo na ausência de estímulos externos.

Com esta visão dinâmica do sistema imunológico é muito natural que os modelos pioneiros de teoria da rede imunológica fossem baseados em Equações Diferenciais Ordinárias. Estes modelos *contínuos* foram importantes para o desenvolvimento dos sistemas imunológicos artificiais e também serviram como base para a proposta de algoritmos de redes imunológicas *discretas*. Esta seção revisa ambas as abordagens separadamente. Foco será dado ao modelo de rede contínua introduzida por Farmer e colaboradores (Farmer et al., 1986) e ao modelo genérico de rede discreta proposta por de Castro e Von Zuben (2001).

A abordagem de rede é particularmente interessante para o desenvolvimento de ferramentas computacionais porque provê, naturalmente, uma explicação de propriedades emergentes como aprendizagem, memória, tolerância ao próprio, tamanho e diversidade de populações de células, e interações de rede com o ambiente e os componentes do próprio (de Castro e Timmis, 2002). O que ambos os

modelos, contínuo e discreto, têm em comum, é uma dinâmica geral que pode ser resumida como segue:

$$RPV = NSt - NSu + INE - DUE \tag{6.4}$$

onde RPV é a taxa de variação da população, NSt é a estimulação da rede, NSu é a supressão da rede, INE é a influência de elementos novos e DUE é a morte de elementos não-estimulados. Note que os dois primeiros termos do lado direito da Equação 6.4 são relacionados à dinâmica da rede imunológica e os dois últimos termos correspondem a metadinâmica imunológica.

Um Modelo de Rede Contínua

Farmer et al. (1986) e Farmer et al. (1987) propuseram modelos binários para a teoria da rede imunológica. Como no modelo proposto por Jerne (1974b), eles consideraram um conjunto de equações diferenciais para representar a dinâmica das células e moléculas do sistema imunológico, juntamente com um limiar de afinidade para remover anticorpos pouco estimulados e operadores genéticos de recombinação (Holland, 1975) para simular as variações genéticas.

Neste modelo, as células e moléculas do sistema imunológico eram constituídas de cadeias binárias com comprimentos variáveis, como ilustrado na Figura 6.15. Uma molécula de anticorpo estava representada pelo seu epítopo (e, ou idiotopo i) e seu paratopo (p), concatenados em uma única sentença binária. As sentenças podiam se ligar por regiões de complementaridade em qualquer alinhamento possível, modelando o fato de que duas moléculas podem reagir de várias formas distintas. A Equação 6.5 corresponde a uma *matriz de especificidades de ligação*, ou simplesmente matriz de especificidades,$m_{i,j}$

$$m_{i,j} = \sum_k G\left(\sum_n e_i(n+k)^{p_j(n)-\varepsilon+1} \right) \tag{6.5}$$

na qual $G(x) = x$ para $x > 0$ e $G(x) = 0$ nos outros casos, $e_i(n)$ é o n-ésimo bit do i-ésimo epítopo , $p_j(n)$ é o n-ésimo bit do j-ésimo paratopo, $\wedge$ corresponde à medida de afinidade (ou regra de ligação) por complementaridade, Equação 6.7, e ϵ corresponde ao limiar de afinidade. Se a ligação ocorre em mais de uma alinhamento, seus pesos são somados.

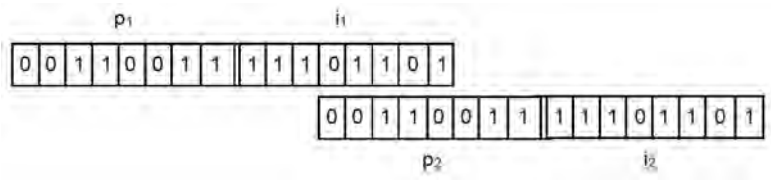


Figura 6.15: Cadeias binárias representando o epítopo (ou idiotopo) e paratopo dos anticorpos.

Para modelar a dinâmica da rede imunológica, foram assumidos N tipos de anticorpos com concentrações $\{x_1, ..., x_N\}$ e M antígenos com concentrações $\{y_1, ..., y_M\}$. A taxa de variação da concentração de anticorpos é dada por:

$$\dot{x_i} = c\left[\sum_{j=1}^{N} m_{j,i} x_i x_j - K_1 \sum_{j=1}^{N} m_{i,j} x_i x_j + \sum_{j=1}^{M} m_{j,i} x_i x_j \right] - k_2 x_i, \tag{6.6}$$

onde o primeiro termo representa a estimulação do paratopo de um anticorpo do tipo i por um idiotopo de um anticorpo j. O segundo termo representa a supressão de um anticorpo do tipo i quando seu

idiotopo é reconhecido pelo paratopo do tipo j. O parâmetro c é uma taxa constante que depende da quantidade de colisões por unidade de tempo e da taxa de produção de anticorpos estimulados por uma colisão. A constante k_1 representa uma possível desigualdade entre estimulação e supressão. O terceiro termo modela as concentrações antigênicas e o último termo modela a tendência das células morrerem (taxa de mortalidade natural, k_2). Uma equação para medir a variação na concentração antigênica também foi apresentada

$$\dot{y_i} = -k_3 \sum_{j=1}^{M} m_{j,i} x_i x_j, \tag{6.7}$$

onde k_3 é uma constante arbitrária.

As Equações de 6.5 a 6.7 representam sistemas adaptativos, no sentido de que anticorpos que reconhecem antígenos ou outros anticorpos possuem seus clones amplificados, ou seja, se reproduzem. Por outro lado, aqueles anticorpos pouco ou não estimulados são eliminados do repertório. A fonte de novos anticorpos fornece ao sistema a capacidade de reconhecer antígenos diferentes daqueles para os quais o sistema está preparado. Como pode ser visto nestas equações, não é feita distinção alguma entre as células B, suas moléculas de superfície e os anticorpos livres.

Um Modelo de Rede Discreta

Na teoria da rede imunológica introduzida por Jerne (1974) o sistema imuno- lógico é composto por uma rede dinâmica cujo comportamento é perturbado pelos antígenos externos. Assim, há basicamente dois níveis de interação na rede: 1) a interação com os antígenos estranhos ao organismo; e 2) a interação com outros elementos da própria rede. Embora haja várias propostas de redes imunológicas artificiais na literatura, esta seção descreve uma das pioneiras. A descrição que segue é baseada na rede imunológica discreta proposta por de Castro e Von Zuben (2001) e nomeada aiNet (Artificial Immune Network). Outras propostas da literatura incluem os trabalhos de Timmis et al. (2000), Neal (2003) e Galeano e González (2005).

Na aiNet há uma população **S** de antígenos a serem reconhecidos e um conjunto **P** de células é aleatoriamente inicializado na rede. Cada elemento da rede corresponde a uma molécula de anticorpo; não há distinção entre o anticorpo e a célula B. Um vetor de valores reais em um espaço de formas Euclidiano é usado para representar anticorpos e antígenos. Cada padrão antigênico é apresentado a cada célula de rede e a afinidade entre eles é determinada usando a distância Euclidiana (Equação (6.1)). Vários anticorpos com alta afinidade ao antígeno são selecionados e reproduzidos (*expansão de clonal*) proporcionalmente à afinidade ao antígeno: quanto maior a afinidade, maior o número de clones e vice-versa. Os clones gerados sofrem mutação somática inversamente proporcional à afinidade antigênica: quanto maior a afinidade, menor a taxa de mutação. Vários clones de afinidade alta são selecionados para serem mantidos na rede, constituindo o que é definido como uma memória clonal.

A afinidade entre os anticorpos restantes é determinada e aqueles cuja afinidade entre si é maior que um determinado limiar são eliminados da rede em um processo conhecido como supressão clonal. Todos os anticorpos cuja afinidade ao antígeno é menor que um determinado limiar também são eliminados da rede (*taxa de mortalidade natural*). Além disso, novos anticorpos gerados aleatóriamente são incorporados à rede (*metadinâmica*). Os anticorpos restantes são incorporados à rede e a afinidade deles aos anticorpos existentes é determinada.

A aiNet pode ser resumida como a seguir (de Castro e Timmis, 2002):

1. *Inicialização*: crie uma população aleatória **P** de anticorpos na rede.

2. *Apresentação de Antígenos*: para cada antígeno **s**, faça:

(a) *Seleção e expansão clonal*: para cada elemento da rede determine sua afinidade em relação ao antígeno apresentado. Selecione um numero n_1 de elementos com maior afinidade e reproduza-os (clonagem) proporcionalmente a sua afinidade.

(b) *Maturação de afinidade*: mute cada clone em proporção inversa a afinidade. Reselecione um número n_2 de clones com maior afinidade e insira-os no conjunto de memória.

(c) *Metadinâmica*: elimine todas as células de memória cuja afinidade ao antígeno seja menor que o limiar de reatividade cruzada ϵ.

(d) *Interações clonais*: calcule a afinidade entre todos os anticorpos de memória.

(e) *Supressão clonal*: elimine as células de memória cuja afinidade entre si seja maior que um limiar de supressão pré-especificado σ_s.

(f) *Construção da rede*: incorpore os clones restantes da memória à rede.

3. *Interações da rede*: determine a afinidade entre todos os anticorpos da rede.

4. *Supressão da rede*: elimine todos os anticorpos da rede cuja afinidade entre si é maior que um limiar de supressão de pré-especificado σ_s.

5. *Diversidade*: introduza um numero n_3 de novos anticorpos gerados aleatoriamente na rede.

6. *Ciclo*: repita os Passos 2 a 5 até que um número de passos pré-especificados (max_it) seja executado.

4. Exemplo de Aplicação: Uma Rede Imunológica para Agrupamento de Dados

O algoritmo imunológico a ser apresentado aqui é intitulado ocopt-aiNet (*Optimal Clustering opt-aiNet*), toma como base o algoritmo imunológico opt-aiNet (de Castro e Timmis, 2002) que, por sua vez, foi desenvolvido a partir da aiNet, brevemente descrita acima. A ocopt-aiNet tem como principal objetivo encontrar múltiplos *agrupamentos* ótimos para uma base de dados. Para isso, a ocopt-aiNet propõe múltiplas partições distintas para a base de dados e utiliza uma função de custo capaz de avaliar a qualidade de cada partição em termos da distância intra e intergrupo de cada objeto da base. A próxima subseção faz uma breve descrição da tarefa de agrupmento de dados e, na sequência, a ocopt-aiNet é apresentada.

O Problema de Agrupamento de Dados

Uma das habilidades mais básicas dos organismos vivos é a capacidade de agrupar objetos similares para produzir uma classificação. A idéia de organizar coisas similares em categorias é bastante antiga e reflete a capacidade de identificar características similares em alguns objetos, como forma, cor, cheiro, posição, altura, etc. *Análise de grupos* ou *clustering* é um termo genérico usado para designar um amplo espectro de métodos numéricos de análise de dados multivariados com o objetivo de descobrir grupos ou clusters homogêneos de objetos (Han e Kamber, 2000; Witten e Frank, 2005; Everitt et al., 2001; Jain et al., 1999). Clustering é o processo de *agrupamento* ou segmentação de um conjunto de objetos em classes ou *clusters* (*grupos*) de objetos similares.

Diferentemente dos processos de classificação, a análise de clusters considera dados de entrada não-rotulados, ou seja, a classe à qual cada objeto de entrada pertence não é conhecida *a priori*. O processo de agrupamento ou clusterização é normalmente usado para identificar tais classes (Everitt et al., 2001). Os objetos são agrupados com o objetivo de *maximizar a distância inter-classe* e *minimizar a distância intra-classe*, ou, dito de outra forma, *maximizar a similaridade intra-classe e*

minimizar a similaridade inter-classe. Portanto, um *cluster* é uma coleção de objetos similares uns aos outros e dissimilares aos objetos de outros clusters. Cada cluster formado pode ser visto como uma *classe de objetos*. Como os rótulos das classes dos dados não são conhecidos de antemão, este processo é denominado de *treinamento não-supervisionado* (ou *aprendizagem não-supervisionada*).

A ocopt-aiNet: Visão Geral

A ocopt-aiNet possui as seguintes características: 1) determinação automática do número de grupos; 2) busca ampla pelo espaço de soluções candidatas, aumentando a probabilidade de se encontrar soluções ótimas globais para o problema; e 3) manutenção de diversidade, que permite a proposição simultânea de diferentes partições de boa qualidade para a mesma base de dados.

Para descrever a ocopt-aiNet considere a seguinte terminologia:

- *Célula*: representa um indivíduo da população que codifica uma possível solução para o problema de agrupamento de dados, ou seja, cada célula da rede associa um objeto da base de dados a um grupo e, portanto, corresponde a uma *partição* dos dados;
- *Antígeno*: cada objeto da base de dados é um antígeno;
- *Clones*: conjunto de células copiadas a partir de uma célula já existente. Os clones são sujeitos ao operador de mutação;
- *Afinidade*: nível de similaridade (interação) entre as células da rede, avaliado através de uma medida de afinidade específica que considera a representação de cada célula da rede;
- *Supressão*: eliminação (extinção) de algumas células da população;
- *Fitness*: valor de adaptabilidade de cada célula em relação a uma função objetivo.

A descrição da ocopt-aiNet será feita com base na proposta de Engenharia Imunológica introduzida neste capítulo.

Representação e Inicialização

A representação das células imunológicas da ocopt-aiNet é feita de forma simples. Considerando uma base contendo N objetos, cada célula possui posições que representam os objetos da base. Dessa forma, uma célula possuirá N posições, sendo que cada posição corresponde a um rótulo que indica o grupo ao qual o objeto pertence. Se a solução possui k grupos, então a possibilidade de rótulos para um objeto será $\{1, ...k\}$. Como exemplo, considere a célula apresentada na Figura 6.16.

Célula 1: [111122221133333]

Figura 6.16: Exemplo de codificação da ocopt-aiNet.

A célula ilustrada na Figura 6.16 possui quinze posições, o que significa que a base de dados possui quinze objetos. Neste caso, seis objetos (1,2,3,4,9,10) formam o grupo 1, o grupo de rótulo 2 é formado por quatro objetos (5,6,7,8), e o grupo de rótulo 3 é formado por cinco objetos (11,12,13,14,15).

A população inicial de células é gerada aleatoriamente, assumindo uma quantidade máxima de grupos pré-definida. Um exemplo de uma população de células é apresentado na Figura 6.17. Este repertório de células possui cinco células (valor escolhido aleatoriamente ou definido preliminarmente) e cada célula possui uma representação da solução de agrupamento.

Célula 1:[111525453111123]
Célula 2:[324451235555322]
Célula 3:[111225424133314]
Célula 4:[352541452411132]
Célula 5:[152342251133455]

Figura 6.17: Exemplo de população inicial gerada aleatoriamente na ocopt-aiNet.

Avaliação de Interações

A ocopt-aiNet possui dois mecanismos de avaliação de interações: 1) uma *medida de fitness*, que avalia a qualidade de cada indivíduo em relação a uma função objetivo capaz de medir a qualidade dos agrupamentos propostos pela rede; e 2) uma *medida de afinidade*, que avalia o grau de similaridade das células da rede entre si.

A função objetivo que será utilizada para avaliar o fitness de cada indivíduo da população é baseada no *critério da Silhueta*, considerado uma estratégia robusta para a predição de agrupamentos ótimos de dados (Bolshakova, 2002). Para explicar a *Silhueta* considere uma base de dados qualquer com N objetos. Seja um objeto i pertencente a um grupo A e a dissimilaridade média entre os outros objetos de A em relação a i denotada por $a(i)$. Seja outro grupo B, a dissimilaridade do objeto i em relação aos objetos de B é denotada por $d(i,B)$. Após o cálculo de $d(i,B)$ para todos os grupos $B \neq A$, o de menor valor é escolhido, ou seja, $b(i) = min\ d(i,B)$, $B \neq A$. Este valor representa o valor do objeto i para o grupo mais próximo. A *Silhueta* é dada pela Equação 6.8:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{6.8}$$

O valor $s(i) \in [-1, +1]$, sendo que quanto maior o valor de $s(i)$ maior é a proximidade do objeto i com um certo grupo. Se o valor de $s(i)$ for zero, então não é possível definir claramente se este objeto i deveria estar no grupo atual ou em outro grupo próximo. Se um grupo A possui apenas um valor, então $s(i)$ não é definida e é considerada zero. O *critério da Silhueta* é dado pela média de $s(i)$, $i = 1$, ... , N e o agrupamento ótimo em relação a este critério é obtido maximizando esta função (Kaufman e Rousseeuw, 1990; Bolshakova, 2002; Hruschka et al., 2004; Argoud et al., 2008).

Para o algoritmo imunológico apresentado a *Silhueta* foi alterada baseando-se no mesmo critério utilizado pelo algoritmo *EAC* (*Evolutionary Algorithm for Clustering*) (Hruschka et al., 2004). Neste algoritmo o *critério da Silhueta* é alterado para que ao invés de ser calculada a distância do objeto aos outros objetos encontrados na base, a distância do objeto tenha seu cálculo feito a partir dos centróides de cada grupo encontrado na célula. Sendo assim, o termo $a(i)$ da Equação 6.8 é alterado para ser a distância do objeto i ao centróide do seu grupo A. Similarmente, o cálculo do termo $d(i,B)$, que trata da distância do objeto i para os objetos do grupo B, $B \neq A$, é alterado para ser a distância do objeto i ao centróide do grupo B, $B \neq A$. Esta alteração promove uma redução do tempo computacional e

mantém uma semelhança com os operadores de mutação (Hruschka et al., 2004) a serem apresentados a seguir.

Na ocopt-aiNet a afinidade tem como objetivo final suprimir células de memória que apresentem codificações equivalentes de agrupamentos. A afinidade das células mantidas para a próxima geração é calculada a partir da quantidade de objetos rotulados para um mesmo grupo em duas células de memória. Neste caso não é utilizado o *fitness* para avaliar a afinidade, pois os valores de *fitness* de duas células podem ser iguais, mesmo quando as codificações dos agrupamentos gerados pelas células são diferentes.

Para ilustrar a medida de afinidade, considere as células ilustradas na Figura 6.18, representando uma base com quinze objetos. Pode-se verificar que as partições representadas pelas células 1 e 2, apesar de possuírem rotulações diferentes para os grupos, representam um mesmo agrupamento. Neste caso a afinidade seria de 100% entre as células 1 e 2, e uma delas seria suprimida (eliminada), restando três células (agrupamentos) diferentes.

Célula 1:[112233333334444]
Célula 2:[221133333334444]
Célula 3:[121233333334444]
Célula 4:[121133333334444]

Figura 6.18: Células de memória após a primeira iteração do algoritmo.

No caso das soluções representadas pelas células 3 e 4 existe uma alta afinidade, mas que não é máxima. Neste caso, o objeto que difere as duas células é o objeto 4. Sendo assim, a afinidade entre estas células é de *93,34%*. Portanto, o algoritmo de cálculo da afinidade entre células considera os agrupamentos formados por cada célula e calcula a similaridade entre eles, independentemente da rotulagem dos dados. A supressão ocorre a partir de um limiar de afinidade escolhido a priori. Caso este limiar seja igual a 1 apenas células com afinidade 100% são suprimidas.

Operadores de Mutação

São utilizados três operadores de mutação na ocopt-aiNet. O primeiro operador é aplicado apenas nas células que possuem mais de dois grupos em sua codificação, promovendo a eliminação, através de uma escolha aleatória, de um dos grupos da solução. Como exemplo pode-se utilizar a célula com a codificação ilustrada na Figura 6.19.

Célula 1: [113413422134434]

Figura 6.19: Célula representando possível solução

Supondo que o grupo 2 seja escolhido para ser excluído desta célula, os objetos nas posições 8 e 9 terão que ser movidos para outro grupo. O(s) novo(s) grupo(s) ao(s) qual(is) estes objetos pertencerão será(ão) escolhido(s) através da proximidade de cada um destes objetos com o centróide de cada grupo remanescente. A Figura 6.20 ilustra um caso hipotético de objetos com dois atributos cada codificados pela célula 1 ilustrada na Figura 6.19 e apresenta os novos grupos dos objetos 8 e 9.

O segundo operador pode ser aplicado em qualquer grupo de uma célula que possua mais que dois objetos. Este operador divide um grupo, escolhido aleatoriamente, em dois novos grupos, sendo que os objetos mais próximos do centróide do grupo original formarão um novo grupo e os objetos mais próximos do objeto mais distante do centróide formarão outro novo grupo. Como exemplo considere a célula ilustrada na Figura 6.21.

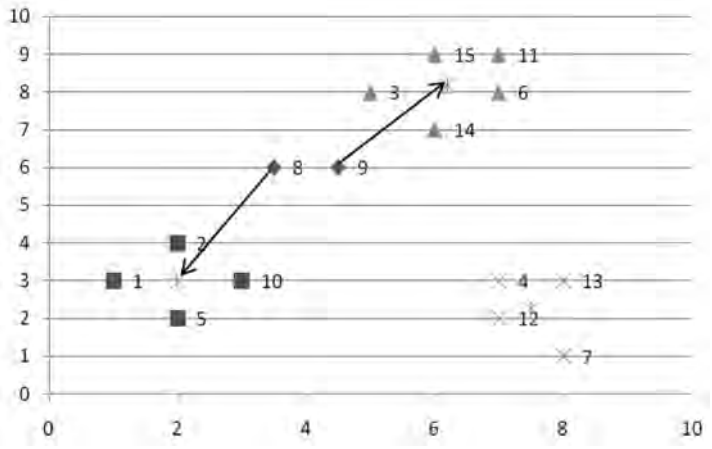


Figura 6.20: Agrupamento codificado pela célula 1, no qual os objetos 8 e 9 pertencentes originalmente ao grupo 2 são realocados para os grupos remanescentes mais próximos.

Célula 2 [113413413134434]

Figura 6.21: Célula utilizada para sofrer mutação através do operador 2.

Foi escolhido aleatoriamente o grupo 4 ao qual será aplicado o operador de mutação 2. Na célula 2 ilustrada na Figura 6.21 o grupo 4 vai ser dividido em dois grupos, um grupo formado pelos objetos 4, 12 e 13 e outro grupo formado apenas pelo objeto 7. Estes grupos com suas novas formações são ilustrados na Figura 6.22

O terceiro operador poderá ser aplicado para todas as posições da célula com probabilidade P= 10% de ocorrência, calculado para cada posição. Este operador verifica se o objeto correspondente à posição encontra-se no grupo correto. Para isso, é verificada a similaridade do objeto com os centróides dos grupos formados pela célula, caso esteja no grupo correto (similaridade maior com o centróide de seu próprio grupo) é mantido o rótulo; caso exista um centróide de outro grupo que este objeto tenha maior similaridade do que o centróide do grupo ao qual ele está alocado, então o rótulo é alterado para o desse centróide com maior similaridade.

Para cada clone a probabilidade de aplicação dos operadores segue a Equação 6.9.

$$P(i) = (1 - f)/2, \tag{6.9}$$

onde f é o valor de fitness da célula pai e $P(i)$ é o valor da probabilidade de ser aplicado um operador de mutação para a célula $i, i = 1, ..., M$, sendo M o tamanho da população nesta iteração. Desta forma, os clones cuja célula pai possui maior fitness terão menor probabilidade de terem aplicados os operadores de mutação.

Apenas um operador pode ser aplicado para cada clone. A probabilidade de ocorrência dos operadores 1 e 2 de mutação é de 25%, enquanto a probabilidade de ocorrência do operador 3 é de 50%.

Algoritmo Imunológico

De forma resumida, a ocopt-aiNet possui os passos descritos no Algoritmo 1.

A ocopt-aiNet possui dois critérios de parada: 1) O primeiro critério (Passo 2), chamado de *critério local* (δ), utiliza a estabilidade do fitness médio da população para determinar se a diferença entre as

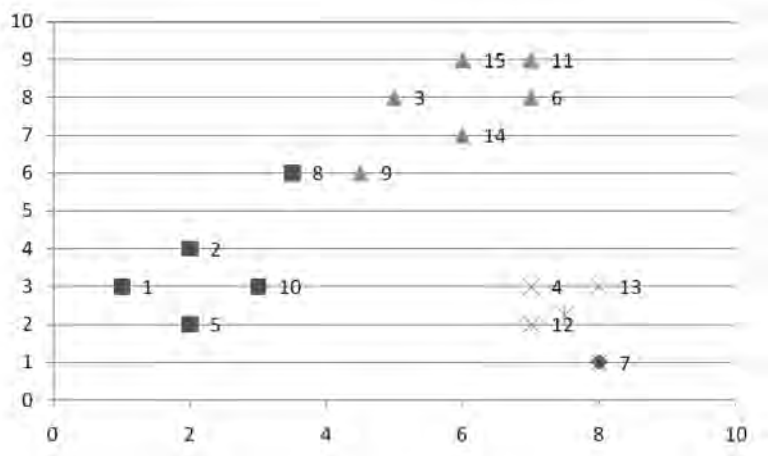


Figura 6.22: Agrupamento codificado pela célula 2 através do qual o grupo 4 forma dois novos grupos (4, 12 e 13; 7).

Algoritmo 1 Pseudocódigo da ocopt-aiNet.

1. *Inicialização*: crie uma população aleatória de células na rede.
2. *Critério de parada clonal*: enquanto o fitness médio da população não variar significativamente em relação ao passo anterior, faça:
   (a) *Seleção e expansão clonal*: para cada célula da rede determine seu fitness normalizado. Isso será feito com base na função objetivo Silhoueta. Gere um numero n_1 de clones para cada célula da rede.
   (b) *Maturação de afinidade*: mute cada clone em proporção inversa a afinidade e selecione a célula de maior fitness para cada clone gerado. Calcule o fitness médio da população de células.
3. *Interações da rede*: calcule a afinidade entre todas as céluas da rede.
4. *Supressão da rede*: elimine as células cuja afinidade entre si seja menor que um limiar de supressão pré-especificado σ_s.
5. *Diversidade*: introduza um numero n_2 de novas células geradas aleatoriamente na rede e volte ao Passo 2.
6. *Ciclo*: repita os Passos 2 a 5 até que o número de grupos em uma janela de tamanho w varie menos que um limiar de estabilidade global Δ.

células na iteração atual e as células da iteração anterior é menor que um limiar δ determinado pelo usuário, o algoritmo passa para o próximo passo caso a diferença seja menor que esse limiar; e 2) O segundo critério (Passo 6), chamado *critério global* (Δ), tem por objetivo determinar a estabilidade dos grupos gerados em um determinado número de iterações, chamado de *janela de estabilidade* (w). Para esse critério caso a diferença do número médio de grupos não alcançar o limiar Δ, então o algoritmo irá finalizar a execução, indicando que o número de grupos proposto esteve estável durante

a janela de estabilidade.

Avaliação de Desempenho

Para comparar o desempenho da ocopt-aiNet foram utilizados dois algoritmos conhecidos na literatura, o *k-means* e o *EAC* (*Evolutionary Algorithm for Clustering*) (Hruschka et al., 2004). O algoritmo *EAC* foi desenvolvido com o objetivo de encontrar automaticamente partições ótimas de uma base de dados, enquanto o *k-means* requer a definição à priori do valor de k, ou seja, do número de grupos a serem encontrados na base.

Inicialmente foram realizados testes com duas bases de dados da literatura:

1. A primeira base de dados, chamada *Carros*, está disponível no Repositório de Bases de Dados de Aprendizagem de Máquina da Universidade da Califórnia em Irvine (Newman et al., 1998). Nos experimentos a serem apresentados aqui foram escolhidos 50 objetos da base, cada um com 8 atributos, sendo 7 deles numéricos e 1 nominal; não há valores ausentes na base.

2. A segunda base, chamada *Animais*, possui 16 objetos com 13 atributos binários cada (Haykin, 1999); não há valores ausentes.

Os algoritmos foram implementados utilizando a linguagem C# e os testes foram realizados em um microcomputador AMD Turion X2 1.8 Ghz com 2Gb de memória RAM. O algoritmo ocopt-aiNet foi parametrizado com número inicial de células $n = 20$, clones $n_1 = 10$, limiar de estabilidade local $\delta = 0.001$, limiar de estabilidade global $\Delta = 0.001$, janela de estabilidade global $w = 20$, geração de $n_2 = 10\%$ novos indivíduos e taxa de supressão $\sigma_s = 0\%$. Para o algoritmo *EAC* foram utilizados os seguintes parâmetros: $n = 20$ indivíduos; e número de iterações $it = 5000$. Esse algoritmo utiliza o *k-means* para busca local como parte de seu processamento. Desta forma, o *k-means* dentro do algoritmo *EAC* foi parametrizado com dois possíveis critérios de parada: 1) número máximo de iterações igual a 5; ou 2) diferença máxima absoluta entre centróides em duas iterações consecutivas menor ou igual a 0.001.

Foram realizadas 10 simulações para cada base de dados com o número k de grupos iniciais, tanto para o *EAC*, quanto para a ocopt-aiNet, igual a k =10 e $k = N/2$, onde N representa o número de objetos da base de dados. Como o algoritmo *k-means* requer a definição à priori do número k de partições e este número não varia ao longo das iterações, os testes foram realizados variando o valor de k no intervalo [2, 10]. Assim, foram feitas 10 simulações com cada base com valores de k = 2,..,10 para a base *Animais* e para a base *Carros* k = 2,..,11. O critério de parada para o *k-means* foi a diferença máxima absoluta entre centróides em duas iterações consecutivas menor ou igual a 0.001. Os valores iniciais de k para esse algoritmo foram obtidos com base na análise dos resultados dos algoritmos *EAC* e ocopt-aiNet.

Para avaliar as partições obtidas foi utilizado o valor da função objetivo implementada tanto no algoritmo *EAC*, quanto na ocopt-aiNet, a função *Silhueta*, dada pela Equação 6.8. Esta função apresenta a qualidade das partições obtidas pelos algoritmos considerando a similaridade inter e intra-grupo de cada objeto. Além disso, como um dos objetivos da ocopt-aiNet é obter múltiplas partições de boa qualidade em um mesmo experimento, foi utilizado um índice para avaliação da diversidade das partições obtidas, o *Adjusted Rand Index* (ARI). O ARI é adotado em diversos trabalhos para analisar a qualidade de partições geradas por algoritmos de agrupamento quando os grupos são conhecidos a priori, ou seja, esse índice apresenta um dado sobre a partição obtida e avalia se é uma partição de boa qualidade.

Para explicar o *Adjusted Rand Index* considere a *matriz de confusão* de duas possíveis partições A e B para uma base de dados qualquer. As linhas da matriz correspondem aos grupos de A e as colunas correspondem aos grupos de B. O valor m_{ij} na posição i, j da matriz de confusão corresponde

ao número de objetos no grupo i da partição A e grupo j na partição B. Seja s_i o valor da soma de todas as colunas da linha i; desta forma tem-se s_i correspondendo ao número de objetos no grupo i da partição A. Seja s_j o valor da soma de todas as linhas da coluna j; desta forma, tem-se s_j correspondendo ao número de objetos no grupo j da partição B. Seja k_a o número de grupos de A e k_b o número de grupos de B. O valor do *Adjusted Rand Index*, ARI, é calculado usando os valores m_{ij} da matriz de confusão das duas partições (Kuncheva e Hadjitodorov, 2004), como apresentado nas Equações (6.10) e (6.11)

$$t_1 = \sum_{i=1}^{k_a} \binom{s_i}{2}, t_2 = \sum_{j=1}^{k_b} \binom{s_j}{2}, t_3 = \frac{2t_1 t_2}{N(N-1)} \tag{6.10}$$

$$ARI(A,B) = \frac{\sum_{i=1}^{k_a}\sum_{j=1}^{k_b}\binom{m_{ij}}{2} - t_3}{\frac{1}{2}(t_1+t_2) - t_3} \tag{6.11}$$

Aqui o *Adjusted Rand Index* é utilizado para avaliar a diversidade das partições. Como ele varia no intervalo [0,1], valores altos para a função objetivo e valores baixos para o *Adjusted Rand Index* são interpretados como partições de boa qualidade e alta diversidade, e vice-versa.

A Tabela 1 apresenta a média e o desvio padrão, a partir de 10 experimentos, dos valores da função objetivo para as partições encontradas por cada algoritmo. Como o algoritmo *EAC* tende a encontrar uma melhor partição como solução para o agrupamento seu valor médio encontrado foi maior que para os algoritmos *ocopt-aiNet* e *k-means*.

	k-means	EAC $k=10$	EAC $k=N/2$	ocopt-aiNet $k=10$	ocopt-aiNet $k=N/2$
Carros	0.7976±0.016	0.8668±0.001	0.8664±0.001	0.8298±0.008	0.8399±0.004
Animais	0.7504±0.024	0.8063±0.002	0.8063±0.002	0.7669±0.004	0.7658±0.004

Tabela 6.1: Valores da função objetivo para as bases Carros e Animais, e algoritmos ocopt-aiNet, k-means e EAC.

A Tabela 2 e a Tabela 3 ilustram o cálculo da diversidade das partições e o número de partições diferentes obtidas pelos algoritmos, respectivamente. Pela Tabela 2 pode-se verificar que o algoritmo EAC não apresentou uma diversidade alta nas soluções obtidas. Porém, como explicado anteriormente, o EAC tem o objetivo de encontrar uma única solução ótima, o que pode ser observado na Tabela 3, que apresenta o número baixo de diferentes partições obtidas nos experimentos.

O algoritmo *k-means*, na base *Carros*, apesar de obter uma diversidade maior que as soluções encontradas pela *ocopt-aiNet*, como descrito na Tabela 2, obteve uma média da função objetivo menor que a *ocopt-aiNet*. Dessa forma, as partições obtidas pela *ocopt-aiNet* apresentaram uma melhor qualidade do que as obtidas pelo *k-means*. Esta avaliação também é válida para a base *Animais*, mas nesta base o algoritmo *ocopt-aiNet* obteve uma diversidade maior para as partições encontradas do que os outros dois algoritmos.

Pela Tabela 3 também pode ser observado que a *ocopt-aiNet* apresenta diversas partições como possíveis soluções para o agrupamento. Apesar deste número elevado de partições o valor da função objetivo (Tabela 1) é melhor que o obtido pelo algoritmo *k-means*, que também obteve mais de uma partição, mostrando que a *ocopt-aiNet* pode encontrar múltiplas partições de boa qualidade.

A Tabela 4 apresenta o número médio de grupos gerados por cada algoritmo. Pode-se observar que o algoritmo *EAC* tende a manter o mesmo número de grupos em suas simulações, enquanto o número de grupos para a *ocopt-aiNet* tem uma maior variação em relação ao número de grupos inicial.

Base	k-means	EAC $k = 10$	EAC $k=N/2$	ocopt-aiNet $k = 10$	ocopt-aiNet $k=N/2$
Carros	0.6853±0.035	0.8730±0.156	0.8700±0.157	0.7027±0.023	0.6882±0.021
Animais	0.6034±0.187	0.9087±0.181	0.9087±0.181	0.4565±0.010	0.4602±0.023

Tabela 6.2: Valores do cálculo da diversidade pelo *Adjusted Rand Index*.

Base	k-means	EAC $k = 10$	EAC $k=N/2$	ocopt-aiNet $k = 10$	ocopt-aiNet $k=N/2$
Carros	8.5±2.41	4.0	3.0	40.2±4.04	63.0±4.87
Animais	8.5±3.00	2.0	2.0	62.5±12.42	66.7±20.13

Tabela 6.3: Número de partições geradas.

O algoritmo *k-means* varia conforme o valor inicial k, mas ao elevar o número k alguns protótipos gerados inicialmente podem ficar estagnados.

Como um dos objetivos da *ocopt-aiNet* é encontrar múltiplas partições que sejam de boa qualidade para a solução do agrupamento e não apenas uma, pôde-se observar, pela Tabela 5, que esse algoritmo exige um tempo computacional maior que os outros algoritmos para os problemas abordados. A *ocopt-aiNet* possui um número variável de células por iteração e todas estas células são consideradas possíveis soluções, só sendo suprimidas caso possuam uma alta afinidade com outra célula já existente na rede. Dessa forma, o número de células pode ser elevado, como observado na Tabela 6.3, enquanto o algoritmo *k-means* é executado apenas sobre uma partição e o algoritmo *EAC* trabalha com um número limitado de indivíduos onde apenas o de maior fitness é apresentado como resultado.

Para ilustrar as soluções obtidas pelo EAC e pela ocopt-aiNet, a Figura 6.23 apresenta resultados típicos do EAC e as Figuras 6.24 e 6.25 apresenta resultados típicos da ocopt-aiNet para a base de dados dos Animais. Nesta figura, cada coluna corresponde a um indivíduo da população e os círculos indicam objetos mapeados no mesmo grupo.

Base	k-means	EAC $k = 10$	EAC $k=N/2$	ocopt-aiNet $k = 10$	ocopt-aiNet $k=N/2$
Carros	4.83±1.60	10.50±0.85	10.10±0.74	7.42±0.21	10.74±0.28
Animais	4.31±1.50	5.60±1.26	5.60±1.26	6.41±0.13	6.07±0.21

Tabela 6.4: Número de grupos das partições.

Base	k-means	EAC k=$N/2$	EAC $k = 10$	ocopt-aiNet $k = 10$	ocopt-aiNet k=$N/2$
Carros	< 00:01±00:00	02:50±00:04	02:55±00:06	07:11±05:37	16:59±21:24
Animais	< 00:01±00:00	00:51±00:03	00:51±00:03	00:56±00:48	02:06±02:06

Tabela 6.5: Tempo computacional de simulação.

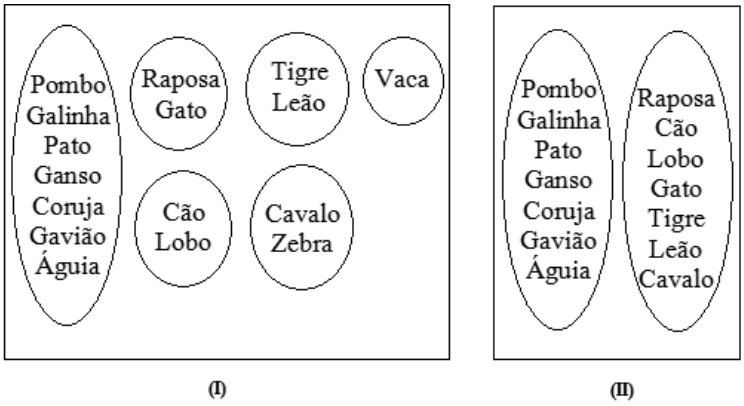


Figura 6.23: Representação de agrupamentos típicos gerados pelo EAC para a base de dados dos Animais.

5. Sistemas Imunológicos Artificiais e Computação Evolutiva

Conforme discutimos anteriormente, há processos que ocorrem no sistema imunológico que são inerentemente evolutivos, porém que representam uma microevolução quando comparada a evolução das espécies. Esta última originou os algoritmos evolutivos e praticamente todos eles são regidos pelos mesmos três princípios básicos: reprodução com herança, variação e seleção. No sistema imunológico a seleção clonal e maturação de afinidade é seguramente o representante mais marcante deste tipo de processo. Fazendo um mapeamento dos sistemas imunológicos artificiais descritos neste capítulo percebemos que há diferentes métodos sendo usados: há algoritmos de seleção negativa, algoritmos de seleção clonal e algoritmos de redes imunes, dentre outros. Destes, os algoritmos de seleção clonal são representantes naturais de algoritmos evolutivos, embora sua fonte de inspiração seja diferente, em essência, daquela que deu origem aos algoritmos evolutivos. As propostas de seleção negativa e rede imunológica são, por sua vez, diferentes daquelas dos algoritmos evolutivos, embora também possam primariamente conter a essência dos algoritmos evolutivos. Por estas razões podemos dizer que os sistemas imunológicos artificiais fazem parte do amplo espectro de técnicas evolutivas, mas não em sua totalidade. Para sermos mais precisos devemos dizer que os sistemas imunológicos artificiais constituem um novo paradigma da inteligência computacional (de Castro e Timmis, 2002) ou que eles formam um novo paradigma de computação flexível (de Castro e Timmis, 2003).

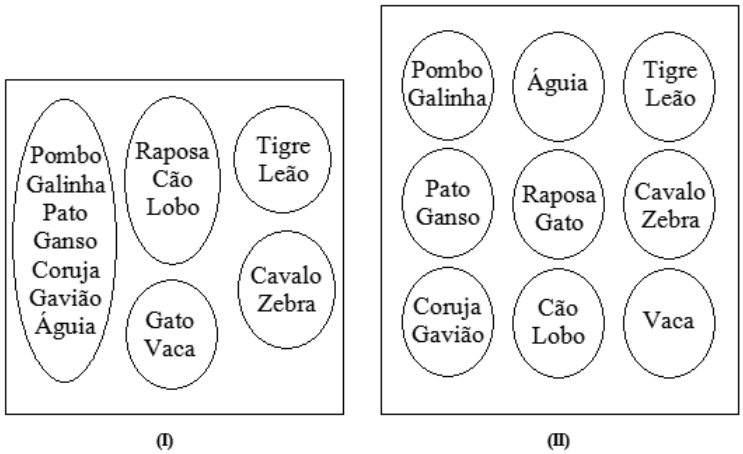


Figura 6.24: Representação de agrupamentos típicos gerados pela ocopt-aiNet para a base de dados dos Animais.

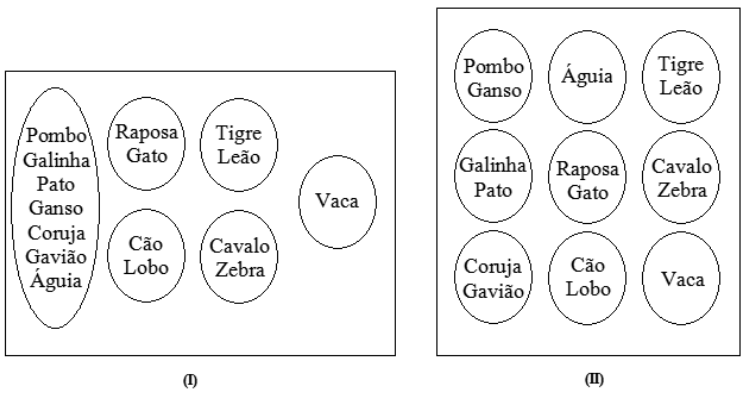


Figura 6.25: Representação de agrupamentos típicos gerados pela ocopt-aiNet para a base de dados dos Animais.

PARTE II

Métodos Não Bio-Inspirados

(Página deixada propositadamente em branco)

CAPÍTULO 7

Evolução Diferencial

Frederico G. Guimarães

Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais

Este capítulo apresenta os algoritmos de *evolução diferencial* para a solução de problemas de otimização. Inicialmente, faz-se uma apresentação do algoritmo em sua versão básica e uma análise de seu comportamento em algumas funções-objetivo usadas como exemplo. Na sequência, são discutidos aspectos avançados e variações do algoritmo de evolução diferencial. Posteriormente, é desenvolvido um esquema geral para os algoritmos de evolução diferencial.

1. Introdução

O algoritmo de Evolução Diferencial é um algoritmo de otimização simples e eficiente que tem recebido cada vez mais destaque no âmbito da otimização não linear com variáveis contínuas. A primeira publicação sobre esse algoritmo ocorreu em 1995, em um relatório técnico de Rainer Storn e Kenneth Price (Storn e Price, 1995). O algoritmo ganhou destaque na comunidade internacional de Computação Evolutiva nos anos seguintes, após apresentar excelente desempenho nas edições de 1996 e 1997 da *International Contest on Evolutionary Optimization* da *IEEE International Conference on Evolutionary Computation* (IEEE ICEC), ver (Storn e Price, 1996; Price, 1997). Na edição de 1996, o algoritmo de evolução diferencial ficou em terceiro lugar. O algoritmo classificado em primeiro

explorava a característica de separabilidade presente nas funções de teste usadas na competição e o algoritmo classificado em segundo lugar se sustentava em quadrados latinos e, por essa razão, não era muito escalável para problemas com muitas variáveis. O algoritmo de evolução diferencial foi o melhor classificado entre os algoritmos de otimização de propósito geral, perdendo para dois métodos mais especializados e menos gerais. Já na edição de 1997, com um novo conjunto de funções de teste, o algoritmo de evolução diferencial apresentou o melhor desempenho entre os algoritmos classificados. Em dezembro do mesmo ano, Storn e Price (1997) publicaram um artigo no *Journal of Global Optimization* apresentando testes experimentais mais amplos e resultados empíricos que ilustravam a robustez do algoritmo.

Em 1999, foi publicado o livro *New Ideas in Optimization*, editado pelos pesquisadores David Corne, Marco Dorigo e Fred Glover, incluindo uma seção de capítulos sobre o algoritmo de evolução diferencial (Lampinen e Zelinka, 1999; Price, 1999; Storn, 1999). Atualmente, já podem ser encontrados alguns livros dedicados exclusivamente aos algoritmos de evolução diferencial, ver por exemplo (Price et al., 2005; Feoktistov, 2006; Chakraborty, 2008), incluindo um livro recente com foco e diversas aplicações em engenharia elétrica e eletrônica (Qing, 2009).

Nos últimos anos, esse algoritmo tem se mostrado versátil e eficaz em muitas aplicações práticas, desde o projeto de filtros digitais (Storn, 1999) até a otimização de sistema reservatório de água (Reddy e Kumar, 2007) e localização do foco sísmico de terremotos (Ruzek e Kvasnicka, 2001). O algoritmo de evolução diferencial também tem se mostrado eficiente para o treinamento de redes neurais (Masters e Land, 1997; Magoulas et al., 2001; Abbass, 2002; Ilonen et al., 2003), para o projeto de dispositivos de engenharia elétrica (Pahner e Hameyer, 2000; Kim et al., 2007) e solução de problemas inversos (Michalski, 2001).

Pesquisas recentes têm se concentrado no estudo de variações do algoritmo (Mezura-Montes et al., 2006), em técnicas para o tratamento de restrições (Kim et al., 2007; Gong e Cai, 2008) e em versões para a solução de problemas de otimização multiobjetivo (Xue et al., 2005; Reddy e Kumar, 2007; Batista et al., 2009).

Este capítulo tem como objetivo apresentar uma visão geral sobre esse algoritmo e seu peculiar mecanismo de busca, sustentado no operador de *mutação diferencial*, que dá nome ao algoritmo. Embora o método de evolução diferencial seja classificado como um algoritmo evolutivo, e se enquadre em um esquema geral de um algoritmo evolutivo, a mutação diferencial não tem base ou inspiração em nenhum processo natural. A forma como esse operador gera perturbações (mutações) nos indivíduos da população se sustenta em argumentos matemáticos e heurísticos que indicam sua adequabilidade para a otimização de funções e não exatamente em argumentos derivados de metáforas da natureza. Contudo, o algoritmo de evolução diferencial segue uma linha histórica de algoritmos e métodos que evoluem uma população de soluções candidatas segundo operadores heurísticos inspirados em mecanismos bastante gerais de adaptação natural. Por essa razão, o algoritmo de evolução diferencial é classificado como uma instância dos algoritmos evolutivos.

A mutação diferencial emprega a diferença entre pares de indivíduos na população corrente para gerar os vetores de perturbação, denominados *vetores-diferença*. Porém, à medida que o algoritmo progride no processo de busca, a distribuição espacial da população se modifica de acordo com o panorama da função-objetivo. Essa mudança, por sua vez, altera as orientações e tamanhos dos vetores-diferença que podem ser criados a partir da população. Por essa razão, observa-se que a distribuição dos vetores-diferença, e portanto a distribuição das direções e tamanhos de passo das perturbações, se ajusta ao panorama da função. Essa característica de autoadaptação da mutação diferencial confere ao algoritmo de evolução diferencial qualidades interessantes do ponto de vista da otimização, tais como robustez, versatilidade e eficiência em diversos problemas. Neste capítulo, mostraremos por meio de exemplos essa autoadaptação dos vetores-diferença à medida que o algoritmo avança no processo de otimização.

2. Evolução Diferencial

Nesta seção é apresentada uma visão geral do algoritmo de evolução diferencial. Considere um problema genérico de otimização não linear com variáveis contínuas, formulado como:

$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$

$$\text{sujeito a:} \begin{cases} \mathbf{x} \in \mathbb{R}^n \\ g_1(\mathbf{x}) \leq 0 \\ \vdots \\ g_m(\mathbf{x}) \leq 0 \end{cases} \tag{7.1}$$

Neste capítulo, vamos considerar o problema irrestrito, isto é, sem as funções de restrição $g_1(\mathbf{x})$ a $g_m(\mathbf{x})$. Lampinen (2002) apresenta uma forma simples de tratar restrições no algoritmo de evolução diferencial. O tratamento de restrições em algoritmos evolutivos será discutido de forma mais geral no Capítulo 14.

Ao longo deste capítulo, usaremos a notação $\mathcal{U}_{[a,b]}$ para indicar a amostragem de uma variável aleatória com distribuição uniforme entre a e b e a notação $\mathcal{N}_{[\mu,\sigma]}$ para indicar a amostragem de uma variável aleatória com distribuição normal com média μ e desvio padrão σ.

Seja uma população de soluções candidatas para o problema, representada por $X_t = \{\mathbf{x}_{t,i};\ i = 1, \ldots, N\}$, em que t é o índice da geração corrente e i é o índice do indivíduo na população. Cada indivíduo na população corrente é representado por um vetor coluna:

$$\mathbf{x}_{t,i} = \begin{bmatrix} x_{t,i,1} \\ x_{t,i,2} \\ \vdots \\ x_{t,i,n} \end{bmatrix} \tag{7.2}$$

dessa forma, o terceiro índice indica uma entre as n variáveis do problema de otimização.

O mecanismo de busca do algoritmo de evolução diferencial utiliza vetores-diferença criados a partir de pares de vetores da própria população. Dois indivíduos são selecionados aleatoriamente da população corrente, criando-se um vetor-diferença que nada mais é do que a diferença entre estes dois indivíduos. Este vetor-diferença, por sua vez, é somado a um terceiro indivíduo, também selecionado aleatoriamente, produzindo uma nova solução mutante. A nova solução mutante é portanto o resultado de uma perturbação em algum indivíduo da população, sendo esta perturbação um vetor-diferença construído aleatoriamente. A equação a seguir ilustra esse procedimento:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right), \quad r_1, r_2, r_3 \in \{1, \ldots, N\} \tag{7.3}$$

em que $\mathbf{v}_{t,i}$ representa a i-ésima solução mutante e F é um fator de escala aplicado ao vetor-diferença e parâmetro do algoritmo de evolução diferencial. O vetor $\mathbf{x}_{t,r_1}$, ao qual é aplicada a mutação diferencial, é denominado *vetor de base*.

Usando este procedimento, obtém-se uma população mutante $V_t = \{\mathbf{v}_{t,i};\ i = 1, \ldots, N\}$. Os próximos passos no algoritmo são bem simples. Os indivíduos da população corrente X_t são recombinados com os indivíduos da população mutante, produzindo a descendência ou população de soluções teste U_t. Na versão clássica do algoritmo de evolução diferencial, emprega-se a recombinação discreta com probabilidade $C \in [0, 1]$:

$$u_{t,i,j} = \begin{cases} v_{t,i,j}, & \text{se } \mathcal{U}_{[0,1]} \leq C \ \vee \ j = \delta_i \\ x_{t,i,j}, & \text{caso contrário} \end{cases} \tag{7.4}$$

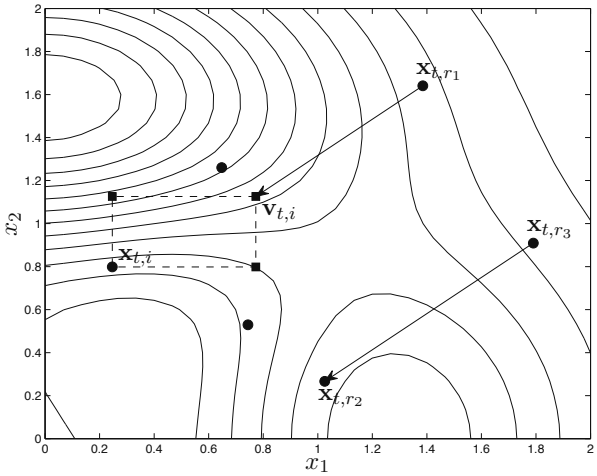


Figura 7.1: Ilustração do procedimento de geração de uma solução mutante.

em que $\delta_i \in \{1, \ldots, n\}$ é um índice aleatório sorteado para o vetor teste i. Como em algum momento a igualdade $j = \delta_i$ será verificada, essa condição garante que pelo menos um dos parâmetros da solução teste será herdado do indivíduo mutante. O parâmetro C controla a fração de valores em $\mathbf{u}_{t,i}$ que são copiados do vetor mutante $\mathbf{v}_{t,i}$. Quanto mais próximo de 1 for o valor de C, maior a chance de que a solução teste contenha muitos valores herdados do vetor mutante. No limite, quando $C = 1$, o vetor teste é igual ao vetor mutante.

A Figura 7.1 ilustra a geração de um vetor mutante e as possíveis soluções teste obtidas após a recombinação, indicadas por ■ na Figura. Note que pelo menos a coordenada $j = \delta_i$ será herdada do vetor mutante, dessa forma, garante-se $\mathbf{u}_{t,i} \neq \mathbf{x}_{t,i}$. Pode-se observar que uma solução teste é o resultado da recombinação de cada solução $\mathbf{x}_{t,i}$ com uma solução mutante gerada a partir de uma perturbação em algum indivíduo da população. A direção e o tamanho dessa perturbação são definidos pela diferença entre as soluções $\mathbf{x}_{t,r_2}$ e $\mathbf{x}_{t,r_3}$, portanto dependem das posições relativas destes indivíduos no domínio de busca.

Finalmente, o valor da função-objetivo é avaliado em $\mathbf{u}_{t,i}$. Cada solução teste $\mathbf{u}_{t,i}$ é comparada com seu correspondente na população corrente, no caso $\mathbf{x}_{t,i}$. Se a solução teste é melhor do que a solução corrente $\mathbf{x}_{t,i}$, a solução corrente é eliminada e seu lugar passa a ser ocupado por $\mathbf{u}_{t,i}$. Caso contrário, a solução teste é descartada e a solução corrente sobrevive, permanecendo na população da próxima geração, representada por X_{t+1}. O processo se repete até que algum critério de parada definido seja satisfeito.

Como pode-se observar, o algoritmo de evolução diferencial em sua versão original é bastante simples. Essa simplicidade torna-se evidente quando escrevemos as equações (7.3) e (7.4) na forma compacta a seguir:

$$u_{t,i,j} = \begin{cases} x_{t,r_1,j} + F\left(x_{t,r_2,j} - x_{t,r_3,j}\right), & \text{se } \mathcal{U}_{[0,1]} \leq C \ \vee \ j = \delta_i \\ x_{t,i,j}, & \text{caso contrário} \end{cases} \tag{7.5}$$

com $r_1, r_2, r_3 \in \{1, \ldots, N\}$, $t = 1, \ldots, t_{\max}$, $i = 1, \ldots, N$, e $j = 1, \ldots, n$.

A seleção para sobrevivência pode ser descrita por:

$$\mathbf{x}_{t+1,i} = \begin{cases} \mathbf{u}_{t,i}, & \text{se } f(\mathbf{u}_{t,i}) \leq f(\mathbf{x}_{t,i}) \\ \mathbf{x}_{t,i}, & \text{caso contrário} \end{cases} \tag{7.6}$$

As operações do algoritmo de evolução diferencial básico são apresentadas na forma de pseudocódigo no Algoritmo 1 a seguir.

Algoritmo 1 Pseudocódigo do algoritmo de evolução diferencial básico

```
1: t ← 1
2: Inicializar população X_t = {x_{t,i}; i = 1, ..., N}
3: enquanto algum critério de parada não for satisfeito faça
4:    para i = 1 até N faça
5:       Selecione aleatoriamente r_1, r_2, r_3 ∈ {1, ..., N}
6:       Selecione aleatoriamente δ_i ∈ {1, ..., n}
7:       para j = 1 até n faça
8:          se U_[0,1] ≤ C ∨ j = δ_i então
9:             u_{t,i,j} = x_{t,r_1,j} + F (x_{t,r_2,j} − x_{t,r_3,j})
10:         senão
11:            u_{t,i,j} = x_{t,i,j}
12:         fim se
13:      fim para
14:   fim para
15:   para i = 1 até N faça
16:      se f(u_{t,i}) ≤ f(x_{t,i}) então
17:         x_{t+1,i} ← u_{t,i}
18:      senão
19:         x_{t+1,i} ← x_{t,i}
20:      fim se
21:   fim para
22:   t ← t + 1
23: fim enquanto
```

A equações (7.5)-(7.6) descrevem todas as operações necessárias no algoritmo de evolução diferencial em sua versão básica. É realmente impressionante que um algoritmo tão simples apresente tantas características computacionais desejáveis, tais como robustez, versatilidade, eficiência e adaptabilidade. Nas próximas seções, tentaremos compreender o comportamento do algoritmo em alguns exemplos de funções-objetivo de forma a elucidar seu mecanismo de funcionamento. Além disso, discutiremos variantes do algoritmo básico e aspectos avançados da evolução diferencial.

3. Comportamento da Mutação Diferencial

Nesta seção discutiremos o comportamento do algoritmo de evolução diferencial, buscando compreender os princípios de funcionamento de seu mecanismo de busca.

O princípio básico de funcionamento do algoritmo de evolução diferencial é perturbar soluções da população corrente gerando vetores mutantes. Essas perturbações são proporcionais à diferença entre pares de soluções escolhidas aleatoriamente na população. Portanto, para entender melhor o comportamento do algoritmo convém verificar a distribuição dos possíveis vetores-diferença em diversos instantes do processo de otimização.

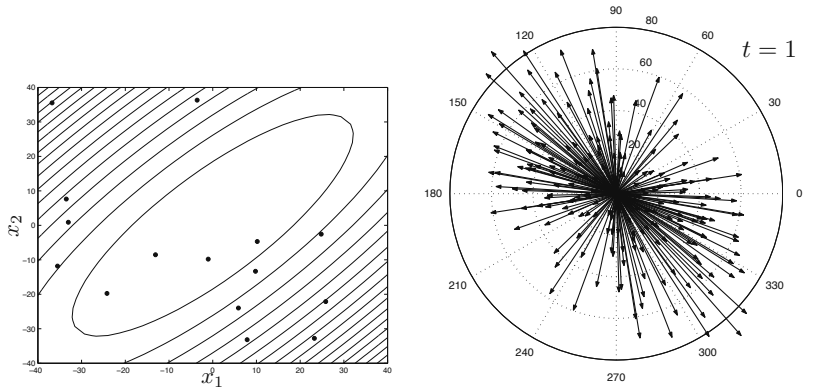
Vamos considerar uma população de tamanho N, supondo inicialmente que os N vetores sejam distintos entre si. Existem ao todo N^2 combinações de diferenças possíveis, das quais N são nulas, pois correspondem a diferenças de um vetor com ele mesmo. As $N(N-1)$ diferenças restantes são não nulas. Além disso, essas $N(N-1)$ diferenças apresentam simetria, porque cada diferença $(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3})$ possui seu simétrico correspondente, bastando para isso trocar os índices r_2 e r_3. Como os índices são escolhidos aleatoriamente com distribuição uniforme, a probabilidade de selecionar uma dada diferença e seu simétrico é a mesma. Fica claro que a distribuição de todos os vetores-diferença possíveis de ser construídos com N vetores distintos quaisquer apresenta média nula, uma vez que cada vetor possui seu correspondente negativo e uma mesma probabilidade de ser selecionado.

Matematicamente, tem-se:

$$\langle \Delta \mathbf{x}_t \rangle = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\mathbf{x}_{t,i} - \mathbf{x}_{t,j}) = \mathbf{0} \tag{7.7}$$

Nesta seção, adotaremos como recurso de visualização o desenho da distribuição dos vetores-diferença em um gráfico polar, considerando que todos tenham a mesma origem em $(\rho = 0, \theta = 0)$. O gráfico polar nos ajuda a visualizar a distribuição dos vetores-diferença de acordo com suas orientações e tamanhos.

Função convexa



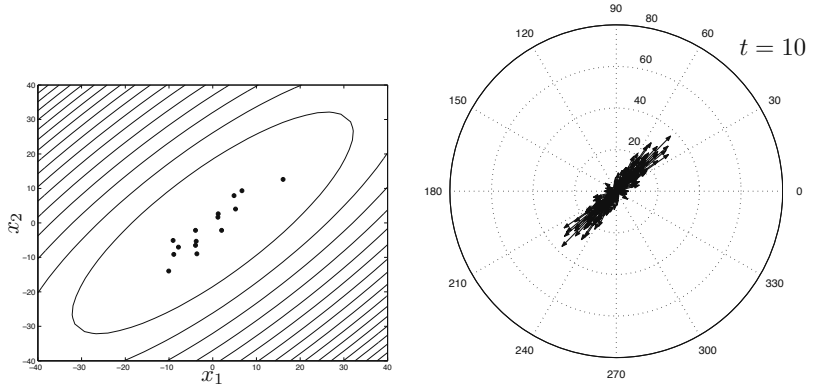
(a) (b)

Figura 7.2: Função-objetivo quadrática. (a) Distribuição espacial da população na geração $t = 1$. (b) Distribuição dos vetores-diferença na geração $t = 1$.

O ponto fundamental para entender o funcionamento do algoritmo de evolução diferencial é perceber que *a distribuição dos vetores-diferença depende da distribuição espacial dos indivíduos da população no problema em questão.* À medida que a população se distribui de acordo com o "contorno" da função, a distribuição dos vetores-diferença também se ajusta a esse contorno.

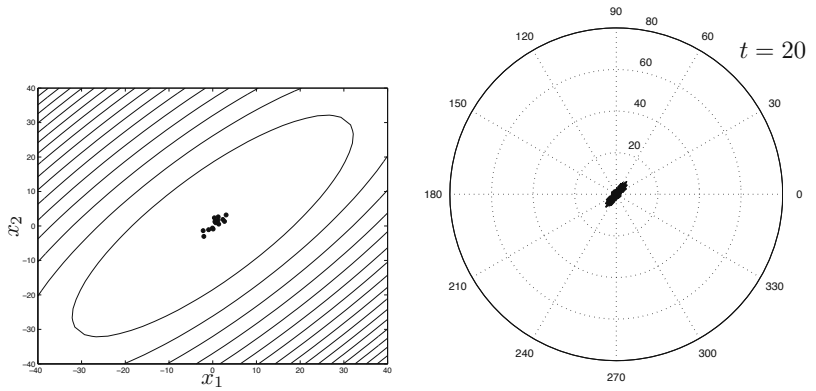
As Figuras 7.2 a 7.4 ilustram essa propriedade do algoritmo em uma função quadrática, cujas curvas de nível correspondem a elipsóides rotacionados de $\pi/4$ no sentido anti-horário em relação aos eixos coordenados. Além disso, um dos eixos desse elipsóide é maior do que o outro, tornando o elipsóide "alongado" numa dada direção.

Na primeira geração, a população é distribuída aleatoriamente com distribuição uniforme em uma região retangular que corresponde aos limites inferiores e superiores de cada variável. Neste primeiro



(a) (b)

Figura 7.3: Função-objetivo quadrática. (a) Distribuição espacial da população na geração $t = 10$. (b) Distribuição dos vetores-diferença na geração $t = 10$.



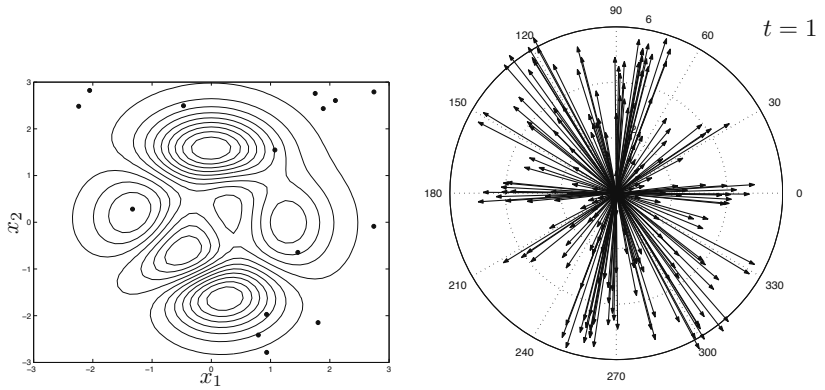
(a) (b)

Figura 7.4: Função-objetivo quadrática. (a) Distribuição espacial da população na geração $t = 20$. (b) Distribuição dos vetores-diferença na geração $t = 20$.

momento, a população não possui nenhuma informação sobre o contorno da função. A Figura 7.2-(a) ilustra a distribuição espacial inicial da população e o gráfico na Figura 7.2-(b) mostra a distribuição dos vetores-diferença correspondente. A distribuição inicial dos vetores-diferença não é polarizada em nenhuma direção, havendo vetores com vários tamanhos distintos e apontando para diversas direções.

Esses vetores-diferença são usados na operação de mutação diferencial para perturbar indivíduos da população, produzindo a população mutante V_t. Cada indivíduo $\mathbf{x}_{t,i}$ sofre recombinação com seu mutante correspondente, produzindo uma solução teste $\mathbf{u}_{t,i}$. Algumas soluções teste serão piores do que as soluções originais e serão descartadas, porém, algumas soluções teste serão melhores e substituirão as soluções originais. Nesse momento, a distribuição espacial da população corrente X_t se altera. A tendência é que, ao longo das gerações, essa distribuição espacial se alinhe com o contorno da função.

A Figura 7.3 ilustra a distribuição espacial da população e a distribuição dos vetores-diferença 10 gerações após a distribuição inicial. Observe que a distribuição dos vetores-diferença agora está mais



(a) (b)

Figura 7.5: Função-objetivo multimodal. (a) Distribuição espacial da população na geração $t = 1$. (b) Distribuição dos vetores-diferença na geração $t = 1$.

alinhada com o eixo maior do elipsóide, ou seja, mais alinhada com a direção mais favorável para a minimização desta função-objetivo quadrática. Outra característica interessante é que os tamanhos de passo estão menores do que na distribuição da Figura 7.2-(b), devido à aglomeração dos indivíduos em torno do ponto de mínimo.

A Figura 7.4 ilustra a distribuição espacial da população e a distribuição dos vetores diferenciais na geração $t = 20$. A população está agora mais próxima do ponto de mínimo e ocupa um volume reduzido em relação à distribuição espacial em $t = 1$. A distribuição dos vetores-diferença continua alinhada com os elipsóides que formam as curvas de nível da função, porém os tamanhos desses vetores estão reduzidos, favorecendo uma busca bem mais local. Nesse momento, o algoritmo converge para o mínimo local.

A sequência de gráficos nas Figuras 7.2 a 7.4 mostra que as direções e os tamanhos de passo dos vetores usados na perturbação das soluções se adaptam ao longo do processo de otimização. As orientações dos vetores-diferença se alinham com a direção mais favorável para a minimização e os tamanhos dos vetores-diferença diminuem à medida que a população se aglomera em torno de algum ponto, favorecendo uma busca cada vez mais local.

Esse exemplo ilustra o comportamento geral do algoritmo de evolução diferencial em funções convexas, mostrando claramente a adaptação dos tamanhos de passo e das direções das mutações. As seções seguintes ilustram o comportamento do algoritmo em uma função multimodal e em uma função unimodal não convexa.

Função multimodal

Na seção anterior vimos como o algoritmo de evolução diferencial se comporta em uma função convexa. Contudo, cabe explorar o comportamento do algoritmo em uma função multimodal, com diversos mínimos locais. Em uma função multimodal, a população tende a se distribuir em torno dos mínimos locais. Nesse caso, convém verificar como ficará a distribuição dos vetores-diferença. Para explorar essas questões, usaremos a mesma metodologia usada anteriormente, mostrando a distribuição espacial da população e o gráfico em coordenadas polares da distribuição de vetores-diferença correspondente em instantes distintos da otimização.

As Figuras 7.5 a 7.7 mostram o resultado obtido para o caso da otimização de uma função-objetivo multimodal.

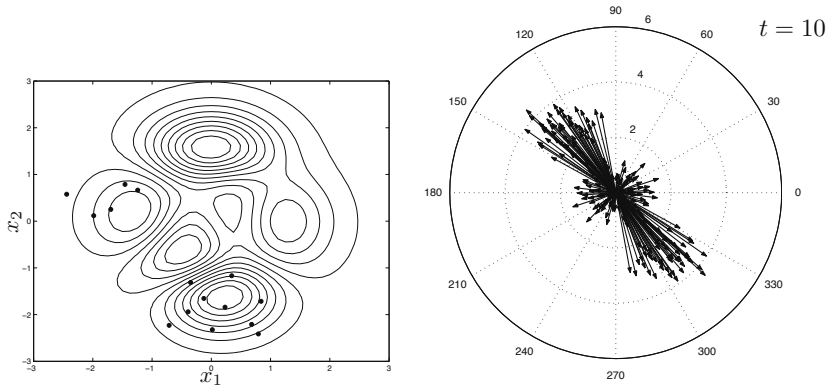


Figura 7.6: Função-objetivo multimodal. (a) Distribuição espacial da população na geração $t = 10$. (b) Distribuição dos vetores-diferença na geração $t = 10$.

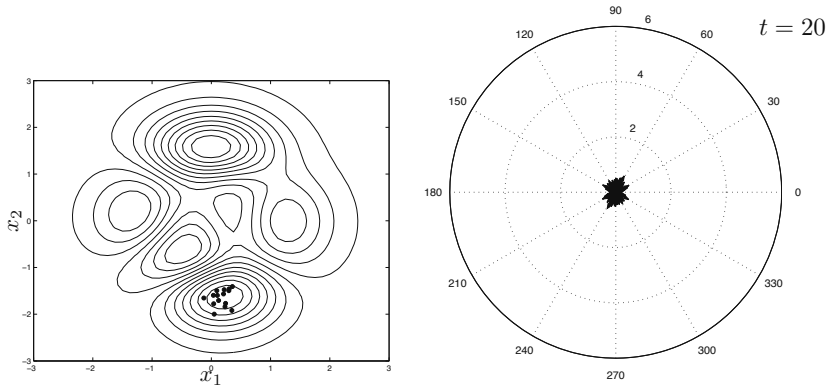


Figura 7.7: Função-objetivo multimodal. (a) Distribuição espacial da população na geração $t = 20$. (b) Distribuição dos vetores-diferença na geração $t = 20$.

Na primeira geração, os vetores-diferença apontam para quase todas as direções e possuem diversos tamanhos devido à geração aleatória da população inicial. Após algumas gerações, ver Figura 7.6, a população se concentra nas duas bacias de atração existentes[1]. Observe a distribuição de vetores-diferença correspondente na Figura 7.6-(b). Pode-se notar dois grupos bem definidos de vetores. Essa distribuição apresenta um conjunto de vetores de pequena amplitude, formados por pares de soluções que se localizam na mesma bacia de atração. Os vetores desse conjunto favorecem uma busca local em cada bacia de atração. A distribuição de vetores-diferença apresenta um segundo conjunto de vetores de maior amplitude, estes por sua vez formados por pares de soluções que se localizam cada uma em bacias distintas. Além disso, as direções desses vetores-diferença de maior amplitude se alinham com a direção que une as duas bacias de atração. Esses vetores-diferença favorecem uma busca global, levando a perturbações que permitem "saltar" de uma bacia à outra, além de gerar soluções na região

[1] Esta função apresenta três bacias de máximo local e duas bacias de mínimo local.

intermediária entre as duas bacias.

Finalmente, na geração $t = 20$, ver Figura 7.7, a população se concentra em uma única bacia, aquela que apresenta melhores valores de função-objetivo. Os vetores-diferença automaticamente diminuem de tamanho, favorecendo uma busca local e mais refinada. A convergência para o mínimo global da função na região considerada é agora iminente.

Esse exemplo ilustra que, no caso de funções multimodais, a distribuição espacial da população se concentra em bacias de atração distintas, causando a geração de grupos de vetores-diferença bem definidos. Alguns conjuntos de vetores causam perturbações que levam a saltos na direção de uma bacia à outra, enquanto outros grupos de vetores-diferença causam perturbações pequenas, favorecendo uma busca local em cada bacia de atração. Após algumas gerações, a população se concentra em uma única bacia e a distribuição de vetores-diferença se reduz ao caso de uma função-objetivo convexa.

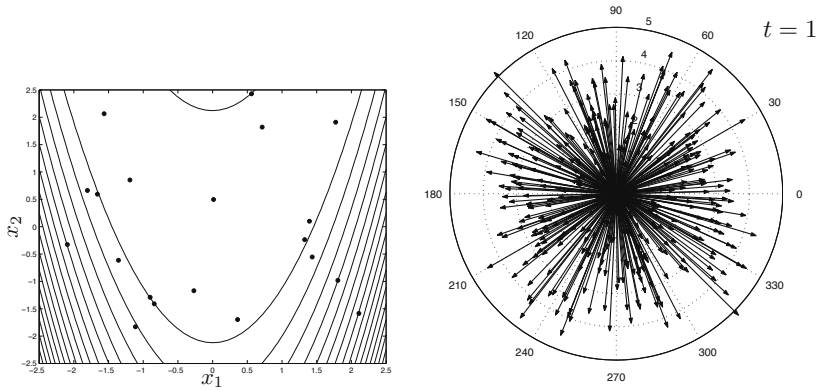
Função unimodal não convexa

Vamos analisar agora o comportamento do algoritmo em uma função unimodal não convexa. Usaremos a função a seguir, uma função-objetivo de teste conhecida como função de Rosenbrock:

$$f(\mathbf{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{7.8}$$

Essa função-objetivo é unimodal e suas curvas de nível formam conjuntos não convexos. Observando as curvas de nível desta função, vemos que seu ponto de mínimo se localiza em um vale estreito e longo com curvatura na forma de uma parábola. Essa região apresenta uma inclinação pequena que dificulta a convergência de métodos de otimização baseados em derivadas.

As Figuras 7.8 a 7.10 mostram o comportamento do algoritmo para essa função não convexa. Observe que após algumas gerações ($t = 10$) a população se concentra em torno do vale em curva da função. O gráfico da distribuição de vetores-diferença apresenta características interessantes. Observa-se dois grupos de vetores em direções quase ortogonais, quase formando a figura da letra 'x'. Os vetores-diferença em cada grupo são formados por pares de vetores localizados em uma das metades da curvatura em formato de parábola. Além disso, observa-se um grupo de vetores com amplitude maior que representam perturbações as quais levam de uma metade à outra da curvatura. O comportamento



(a) (b)

Figura 7.8: Função-objetivo unimodal não convexa. (a) Distribuição espacial da população na geração $t = 1$. (b) Distribuição dos vetores-diferença na geração $t = 1$.

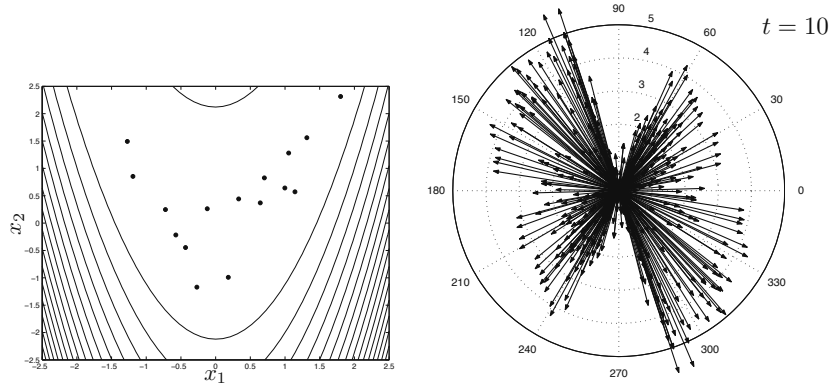


Figura 7.9: Função-objetivo unimodal não convexa. (a) Distribuição espacial da população na geração $t = 10$. (b) Distribuição dos vetores-diferença na geração $t = 10$.

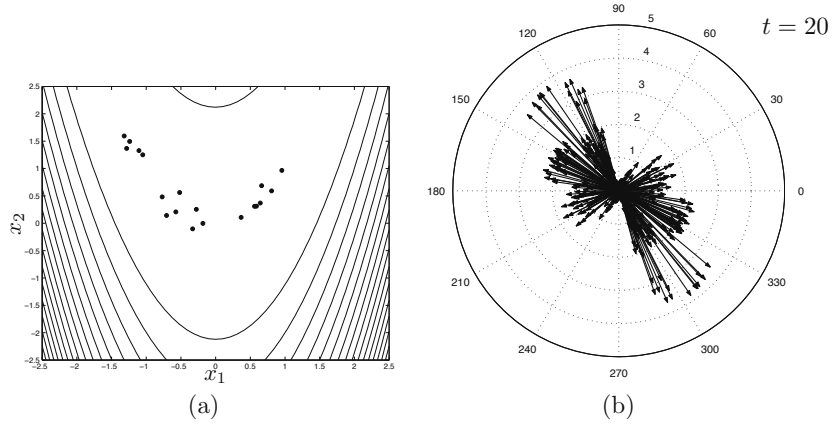


Figura 7.10: Função-objetivo unimodal não convexa. (a) Distribuição espacial da população na geração $t = 20$. (b) Distribuição dos vetores-diferença na geração $t = 20$.

do algoritmo de evolução diferencial nesta função é similar ao comportamento do algoritmo em uma função multimodal.

Quando a distribuição espacial da população apresenta "formato" linear, como no caso da função convexa, dizemos que há uma forte correlação linear entre as variáveis nessa distribuição espacial. O algoritmo de evolução diferencial é capaz de descobrir correlações lineares na distribuição espacial da população e utilizar essa informação para produzir perturbações favoráveis. Entretanto, no caso de distribuições mais complexas do que uma reta, como a distribuição espacial curva na Figura 7.10, o algoritmo trata essa distribuição como uma combinação de distribuições com correlações lineares. Dessa forma, o algoritmo trata funções não convexas, que causariam distribuições espaciais com formatos curvos mais complicados, como se fossem funções multimodais, mesmo que a função não convexa em questão seja unimodal. Cada trecho aproximadamente convexo é enxergado pelo algoritmo como uma bacia de atração e gera um grupo de vetores-diferença.

Na Figura 7.10-(a), pode-se perceber três agrupamentos de indivíduos bem destacados na distribuição espacial da população. Na Figura 7.10-(b), que ilustra a distribuição de vetores-diferença correspondente, pode-se notar também três grupos de vetores-diferença bem distintos. Cada grupo de vetores-diferença está alinhado com a direção que conecta dois agrupamentos distintos e gera perturbações que permitem saltar de um grupo de pontos ao outro. O algoritmo se comporta como se existissem três bacias de atração na função.

Rotação e translação

Uma das características interessantes do operador de mutação diferencial é sua invariância à rotação e translação do sistema de coordenadas. É fácil verificar que a mutação diferencial é invariante à rotação e translação, já que se trata de uma operação vetorial. O que importa na definição dos vetores-diferença são as posições relativas dos indivíduos da população, não suas posições absolutas.

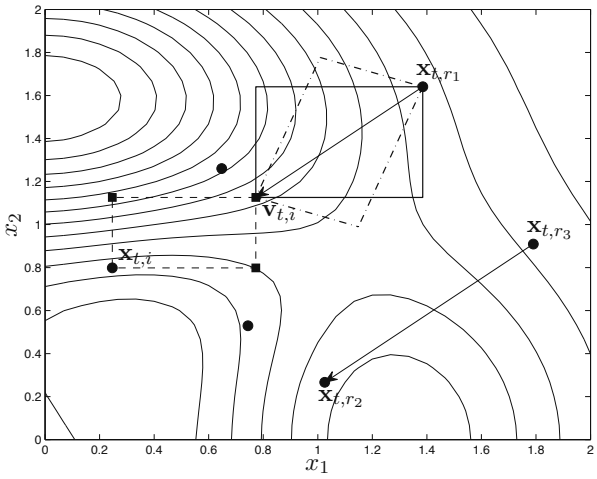


Figura 7.11: Invariância da mutação diferencial em relação à rotação do sistema de coordenadas.

A propriedade de invariância à rotação é uma característica desejável em algoritmos de otimização. Um algoritmo cujo desempenho não depende da orientação do sistema de coordenadas é mais geral, uma vez que a orientação ideal não é conhecida na prática.

Embora a mutação diferencial apresente essas características desejáveis, a recombinação discreta usada no algoritmo de evolução diferencial clássico não possui invariância à rotação. Observando a Figura 7.11, percebe-se claramente que os vetores teste produzidos pela recombinação discreta variam com a rotação do sistema de coordenadas, exceto quando $C = 1$, pois nesse caso não há recombinação, somente mutação, e $\mathbf{u}_{t,i} = \mathbf{v}_{t,i}$.

4. Aspectos Avançados

Combinações degeneradas

Ao apresentarmos a equação do operador de mutação diferencial, ver (7.3), apenas comentamos que os índices r_1, r_2, r_3 são sorteados aleatoriamente para cada i. Entretanto, há uma probabilidade, mesmo que pequena, de que algum par dos índices i, r_1, r_2, r_3 em (7.3) seja coincidente. Nesta seção discutimos esses casos degenerados e procedimentos que podem ser adotados para evitá-los.

As seguintes situações degeneradas podem ocorrer:

1. $r_2 = r_3$. Nesse caso, (7.3) se reduz a:
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1}$$
e não há perturbação aplicada ao vetor de base. O vetor de base não sofre mutação, causando a recombinação de $\mathbf{x}_{t,i}$ com algum vetor escolhido aleatoriamente da população.

2. $r_1 = r_2$ ou $r_1 = r_3$. Nesse caso a mutação diferencial se reduz a um operador de recombinação aritmética[2]. Para $r_1 = r_2$, temos:
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_3}\right) = (1+F)\mathbf{x}_{t,r_1} - F\mathbf{x}_{t,r_3}$$
Ocorre uma recombinação degenerada, em que a nova solução é gerada externamente ao segmento que une $\mathbf{x}_{t,r_1}$ e $\mathbf{x}_{t,r_3}$, ver Figura 7.12. Para $r_1 = r_3$, ocorre uma recombinação aritmética tradicional entre $\mathbf{x}_{t,r_1}$ e $\mathbf{x}_{t,r_2}$:
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_1}\right) = (1-F)\mathbf{x}_{t,r_1} + F\mathbf{x}_{t,r_2}$$
se $F \in [0,1]$, ocorrendo cruzamentos degenerados para valores de $F > 1$.

3. $i = r_1$. Uma perturbação é aplicada a $\mathbf{x}_{t,i}$ de acordo com:
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,i} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)$$
A solução teste é o resultado da recombinação da solução $\mathbf{x}_{t,i}$ com sua versão perturbada. Na prática, o parâmetro C passa a ter o significado de um parâmetro de mutação, controlando quantas variáveis de $\mathbf{x}_{t,i}$ serão perturbadas.

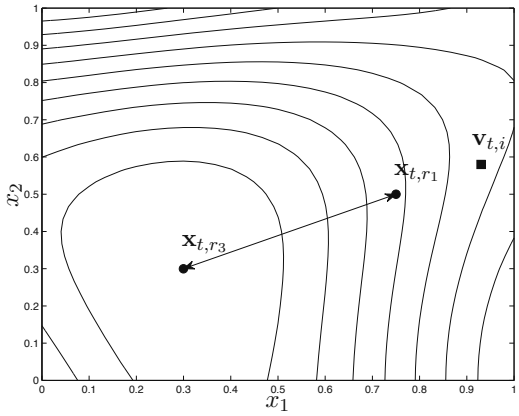
4. $i = r_2$ ou $i = r_3$. Nesse caso o vetor-diferença está na direção que liga $\mathbf{x}_{t,i}$ a algum outro vetor da população, $\mathbf{x}_{t,r_2}$ ou $\mathbf{x}_{t,r_3}$. Essa combinação não é necessariamente indesejável.

Estas combinações degeneradas, mesmo que tenham baixa probabilidade de ocorrência, são em geral indesejáveis e podem prejudicar o desempenho do algoritmo. Por essa razão, recomenda-se adotar índices mutuamente distintos, isto é, $i \neq r_1 \neq r_2 \neq r_3$, em implementações mais práticas do algoritmo de evolução diferencial. Uma maneira simples de garantir essa condição é substituir a linha 5 no Algoritmo 1 pelo código no Algoritmo 2 a seguir.

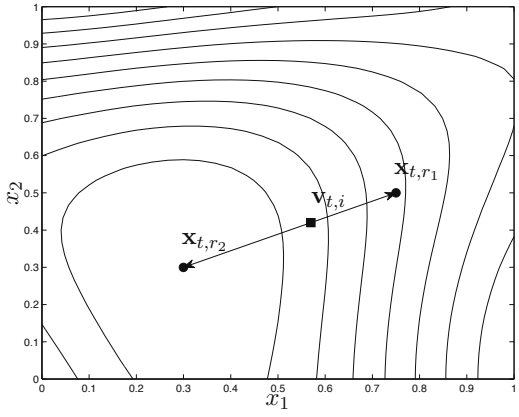
Algoritmo 2 Pseudocódigo para a seleção de índices mutuamente distintos

```
1: repita
2:     Selecione aleatoriamente r1 ∈ {1, ..., N}
3: até r1 ≠ i
4: repita
5:     Selecione aleatoriamente r2 ∈ {1, ..., N}
6: até r2 ≠ i ∧ r2 ≠ r1
7: repita
8:     Selecione aleatoriamente r3 ∈ {1, ..., N}
9: até r3 ≠ i ∧ r3 ≠ r1 ∧ r3 ≠ r2
```

[2] A recombinação aritmética de dois vetores $\mathbf{x}_a$ e $\mathbf{x}_b$ é definida pela combinação convexa $\lambda\mathbf{x}_a + (1-\lambda)\mathbf{x}_b$, para $\lambda \in [0,1]$. Por essa razão, a recombinação aritmética é também conhecida na literatura como recombinação convexa ou recombinação intermediária. Para o caso particular $\lambda = 0.5$, a recombinação é denominada recombinação média. No contexto de algoritmos genéticos, a operação de recombinação é tradicionalmente denominada cruzamento.



(a)



(b)

Figura 7.12: (a) Quando $r_1 = r_2$, ocorre uma recombinação degenerada, em que a nova solução é gerada fora do segmento que une as duas soluções recombinantes. (b) Quando $r_1 = r_3$, ocorre uma recombinação aritmética tradicional. Nos exemplos acima, $F = 0.4$.

Outra maneira de implementar a escolha de índices distintos é utilizar uma rotina de permutação aleatória de um vetor de inteiros, se disponível. Nesse caso, pode-se utilizar essa rotina para gerar uma permutação aleatória do vetor de índices $(1, \ldots, N)$. Em seguida, utilize os três primeiros elementos do vetor resultante como valores para os índices r_1, r_2 e r_3. Esse procedimento assegura $r_1 \neq r_2 \neq r_3$. Para garantir a condição $i \neq r_1 \neq r_2 \neq r_3$, procure o índice i no vetor resultante da permutação aleatória e use os três elementos seguintes como valores para os índices r_1, r_2 e r_3. Nesse caso, deve-se considerar uma contagem circular para as posições do vetor, isto é, ao se chegar à última posição do vetor, a posição seguinte representa um retorno à primeira posição.

Variações do algoritmo

O algoritmo de evolução diferencial em sua versão básica utiliza seleção aleatória com probabilidade uniforme do vetor de base, um vetor-diferença para a mutação, e recombinação discreta entre a solução corrente e seu vetor mutante correspondente. Entretanto, muitas variações desse esquema básico são

possíveis. Nesta seção, comentaremos sobre algumas dessas variações.

Com relação ao vetor de base, este pode ser escolhido aleatoriamente entre os indivíduos da população corrente com probabilidade uniforme ou com probabilidade proporcional à qualidade de cada solução[3]. Ao usar uma seleção com probabilidade uniforme, de fato o algoritmo de evolução diferencial está eliminando a pressão seletiva para a reprodução, uma vez que cada indivíduo possui a mesma probabilidade de ser selecionado como vetor de base, portanto, cada indivíduo produz em média um descendente. Essa forma de seleção corresponde a uma seleção por roleta estocástica em que cada indivíduo ocupa uma área igual na roleta.

A seleção uniforme pode entretanto causar repetição do vetor de base e fazer com que algumas soluções na população não sejam usadas como vetor de base. Para garantir a seleção de um vetor de base único para cada indivíduo, pode-se usar permutação aleatória de um vetor de índices $(1, \ldots, N)$. Dessa forma garante-se que cada indivíduo da população será selecionado uma única vez como vetor de base na mutação diferencial, portanto, cada indivíduo produz um único descendente, um único vetor mutante $\mathbf{v}_{t,i}$. Usando a analogia com a seleção por roleta, essa estratégia de seleção dos vetores de base equivale a um giro de uma roleta com N ponteiros igualmente espaçados, ao invés de N giros de uma roleta com um ponteiro, como na situação anterior. Como cada indivíduo ocupa uma área igual na roleta, cada indivíduo é selecionado uma única vez.

Pode-se ainda utilizar um mesmo vetor como vetor de base em todas as operações de mutação diferencial. A escolha mais comum é usar a melhor solução na população como vetor de base. Assim, temos:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,best} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right) \tag{7.9}$$

em que $\mathbf{x}_{t,best}$ representa o melhor indivíduo na população na geração t.

Pode-se ainda adotar o vetor correspondente à média da distribuição espacial da população como vetor base. As perturbações geradas pela mutação diferencial são aplicadas à média da população:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,mean} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right) \tag{7.10}$$

com:

$$\mathbf{x}_{t,mean} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_{t,i} \tag{7.11}$$

Utilizar um único vetor de base significa adotar uma pressão seletiva forte na reprodução, em contraste à ausência de pressão seletiva da seleção uniforme. Em geral, essa estratégia apresenta uma maior velocidade de convergência e uma rápida redução de diversidade, que podem levar à convergência prematura em alguns problemas, em particular em problemas em que a função-objetivo é multimodal. O algoritmo de evolução diferencial já possui uma pressão seletiva forte na sobrevivência dos indivíduos, em que ocorre uma competição determinística entre o indivíduo $\mathbf{x}_{t,i}$ e a solução teste $\mathbf{u}_{t,i}$. Portanto, se a pressão seletiva para reprodução também for forte, pode ocorrer uma rápida perda de diversidade na população. Na versão clássica do algoritmo, a ausência de pressão seletiva para a reprodução compensa a pressão seletiva forte na etapa de sobrevivência de maneira similar ao que ocorre com as *estratégias evolutivas*, ver Capítulo XX.

Outra possibilidade é gerar soluções na direção do melhor indivíduo para servirem como vetores de base. A equação a seguir ilustra essa abordagem:

$$\mathbf{v}_{t,i} = \underbrace{\mathbf{x}_{t,i} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,i}\right)}_{\mathbf{x}_{t,base}} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right) \tag{7.12}$$

[3] Esquema de seleção proporcional à aptidão muito utilizado em algoritmos genéticos.

com $\lambda \in [0,1]$. Nesse caso, ocorre uma recombinação aritmética entre $\mathbf{x}_{t,i}$ e $\mathbf{x}_{t,best}$. O vetor de base corresponde a um ponto gerado aleatoriamente sobre o segmento que liga $\mathbf{x}_{t,i}$ e $\mathbf{x}_{t,best}$. Essa forma de seleção do vetor de base apresenta menor pressão seletiva do que em (7.9), embora haja uma polarização na direção da melhor solução.

A mutação diferencial pode ser generalizada para empregar mais vetores-diferença na criação do vetor mutante:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + \sum_{k=1}^{d} F_k \Delta \mathbf{x}_{t,k} \tag{7.13}$$

em que a perturbação aplicada ao vetor de base é composta pela soma de d vetores-diferença da forma:

$$\Delta \mathbf{x}_{t,k} = \left(\mathbf{x}_{t,r_{k+1}} - \mathbf{x}_{t,r_{k+1+d}}\right) \tag{7.14}$$

Por exemplo, usando o mesmo valor de F_k para todo k e $d = 3$, a equação (7.13) fica:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_5} + \mathbf{x}_{t,r_3} - \mathbf{x}_{t,r_6} + \mathbf{x}_{t,r_4} - \mathbf{x}_{t,r_7}\right) \tag{7.15}$$

O uso de mais de um vetor-diferença na mutação diferencial aumenta a capacidade de geração de diversidade no algoritmo. Contudo, essa estratégia reduz a capacidade de alinhamento dos vetores-diferença com o contorno da função, embora o alinhamento não seja totalmente perdido. Além disso, fica cada vez mais difícil garantir índices mutuamente distintos. O uso de mais vetores-diferença aumenta a busca global no algoritmo e prejudica a busca local, porque os tamanhos de passo adaptados se somam produzindo perturbações de maior magnitude. Por essa razão, o valor de F deve ser menor para compensar o aumento no tamanho das perturbações.

Existe uma notação sintética para representar as variações do algoritmo de evolução diferencial. Essa notação padrão segue o formato `DE/base/d/rec`. O termo usado no lugar de `base` indica a forma como o vetor de base é selecionado, `d` indica o número de vetores-diferença usados, e o termo usado no lugar de `rec` faz referência ao operador de recombinação utilizado.

Por exemplo, a versão clássica do algoritmo de evolução diferencial, ver Algoritmo 1, pode ser representada pela notação `DE/rand/1/bin`, em que `rand` indica que o vetor base na mutação diferencial é escolhido aleatoriamente com probabilidade uniforme, $d = 1$ indica que apenas um vetor-diferença é usado, e `bin` indica o método de recombinação. O termo `bin` faz referência à distribuição binomial, pois a recombinação discreta com probabilidade C faz com que o número de valores herdados de $\mathbf{v}_{t,i}$ siga uma distribuição binomial. A probabilidade de que p valores sejam herdados do vetor mutante é dada por:

$$P\{X = p\} = \binom{n}{p} C^p (1-C)^{n-p} \tag{7.16}$$

que corresponde a todas as combinações de p sucessos e $n-p$ falhas, isto é, p ocorrências de $\mathcal{U}_{[0,1]}$ que foram menores do que C e $n-p$ ocorrências de $\mathcal{U}_{[0,1]}$ que foram maiores do que C.

Existe uma forma alternativa de implementação da recombinação discreta, que produz uma distribuição exponencial do número de valores herdados da solução mutante. Na recombinação discreta binomial uma realização de $\mathcal{U}_{[0,1]}$ é obtida para cada coordenada e o valor de $u_{t,i,j}$ é copiado de $v_{t,i,j}$ se $\mathcal{U}_{[0,1]} \leq C$ para a coordenada j, isto é, cada coordenada é testada de maneira independente. Na recombinação discreta exponencial, indicada por `exp` na notação padrão, uma posição no vetor é sorteada aleatoriamente com probabilidade uniforme. A partir dessa posição, os valores da solução teste são herdados da solução mutante *enquanto* $\mathcal{U}_{[0,1]} \leq C$. Na primeira ocorrência de $\mathcal{U}_{[0,1]} > C$, os valores para as coordenadas restantes são obtidos de $\mathbf{x}_{t,i}$. Observe que nesse caso a probabilidade de que p valores sejam herdados do vetor mutante é dada por:

$$P\{X = p\} = C^p(1-C) = C^p - C^{p+1} \tag{7.17}$$

que corresponde a p ocorrências *sucessivas* de $\mathcal{U}_{[0,1]}$ que foram menores do que C e uma ocorrência de $\mathcal{U}_{[0,1]}$ que foi maior do que C.

Na recombinação discreta do tipo exponencial, as primeiras coordenadas do vetor teste a partir da posição sorteada possuem maior probabilidade de serem herdadas do vetor mutante do que as últimas coordenadas. Em outras palavras, os valores das coordenadas em posições adjacentes têm maior probabilidade de permanecerem juntas no vetor teste do que os valores em posições não adjacentes. Na terminologia usada em algoritmos evolutivos, dizemos que esse operador apresenta *polarização posicional*, ou seja, uma polarização na probabilidade de que algumas posições do genótipo sejam escolhidas em detrimento de outras. Para contornar a polarização posicional desse tipo de recombinação, os índices das coordenadas podem ser embaralhados aleatoriamente antes de aplicar a recombinação.

Para concluir esta seção, a Tabela 7.1 apresenta alguns exemplos de instâncias do algoritmo de evolução diferencial, cada uma com sua notação padrão correspondente. Se o melhor indivíduo for usado como vetor de base, e utilizarmos três vetores-diferença na mutação diferencial, podemos referenciar o algoritmo de evolução diferencial correspondente como `DE/best/3/bin`. O algoritmo de evolução diferencial que utiliza a abordagem de seleção do vetor de base em (7.12) e recombinação discreta exponencial pode ser representado pela notação `DE/current-to-best/1/exp`.

Notação	Mutação diferencial
`DE/rand/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)$
`DE/best/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,best} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)$
`DE/mean/1/bin`	$\mathbf{v}_{t,i} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_{t,k} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)$
`DE/rand-to-best/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,r_1}\right) + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)$
`DE/current-to-best/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,i} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,i}\right) + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)$
`DE/rand/2/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + F_1\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_4}\right) + F_2\left(\mathbf{x}_{t,r_3} - \mathbf{x}_{t,r_5}\right)$

Tabela 7.1: Algumas instâncias do algoritmo de evolução diferencial.

Variação do parâmetro de escala

O parâmetro de escala F do vetor-diferença não necessariamente precisa variar, pois os tamanhos de passo são autoadaptados na distribuição de vetores-diferença, conforme vimos anteriormente. À medida que o algoritmo progride, e a população se agrupa em torno de um ponto de mínimo local, os tamanhos de passo automaticamente diminuem devido à proximidade das soluções no domínio de busca. Dessa forma, o parâmetro de escala pode ser mantido fixo no algoritmo de evolução diferencial sem comprometer seu desempenho significativamente.

Entretanto, variar o valor de F dentro de alguma faixa contínua de valores aumenta a diversidade de vetores-diferença e de soluções possíveis que podem ser geradas com o operador. Além disso, variar o parâmetro de escala reduz as chances de estagnação do algoritmo. A estagnação no algoritmo de evolução diferencial pode ocorrer com a escolha de um valor fixo para F em situações degeneradas, quando a população está presa numa dada distribuição espacial que não oferece vetores-diferença que permitam a geração de soluções melhores. Variando o valor de F, a possibilidade de estagnação diminui.

Uma maneira simples de variar o valor de F é fazer:

$$F = \mathcal{U}_{[a,b]} \tag{7.18}$$

Tipicamente, adota-se $\mathcal{U}_{[0.5,1]}$. Valores muito pequenos de F prejudicam a convergência pois reduzem muito o efeito da mutação diferencial, causando perturbações pequenas nos vetores de base. Por outro lado, valores de F maiores do que 1 desaceleram a redução natural dos tamanhos de passo, retardando a convergência do algoritmo. Observe que F pode variar em uma faixa relativamente pequena de valores, uma vez que os tamanhos de passo são adaptados no algoritmo de evolução diferencial de acordo com a distribuição espacial da população. Por exemplo, Zaharie (2002) indica o valor 0.3 como um limite inferior confiável para F. Outros estudos empíricos indicam um limite inferior para F de 0.4 (Ali e Törn, 2000; Gamperle et al., 2002). Até o momento, estudos empíricos na literatura indicaram que não há benefícios em se utilizar valores superiores a 1 para F.

Alguns autores estudaram o emprego de parâmetros de escala independentes para cada variável, fazendo a mutação diferencial da forma:

$$v_{t,i,j} = x_{t,r_1,j} + F_j \left(x_{t,r_2,j} - x_{t,r_3,j}\right) \tag{7.19}$$

ou, escrito na forma matricial:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_1} + \mathbf{F} \left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right) \tag{7.20}$$

com:

$$\mathbf{F} = \begin{bmatrix} F_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & F_n \end{bmatrix} \tag{7.21}$$

O parâmetro de escala F_j é retirado de uma distribuição aleatória independente para cada variável. Contudo, essa escolha se mostrou desvantajosa, porque utilizar valores independentes de F_j descorrelaciona as componentes do vetor-diferença, destruindo a correlação linear entre as variáveis aprendida pelo operador de mutação diferencial e o alinhamento das orientações dos vetores-diferença.

Entretanto, pequenas variações nos valores de F_j em torno de um valor central representam pequenas variações de ângulo nos vetores-diferença e podem ser interessantes para aumentar a diversidade de soluções geradas sem prejudicar gravemente o alinhamento dos vetores-diferença. Para implementar essa abordagem, pode-se utilizar:

$$F_j = F_0 + \mathcal{U}_{[-\alpha,\alpha]}, \quad j \in \{1, \ldots, n\} \tag{7.22}$$

ou ainda

$$F_j = F_0 + \mathcal{N}_{[0,\alpha]}, \quad j \in \{1, \ldots, n\} \tag{7.23}$$

com $\alpha \ll 1$ em ambos os casos. O valor de F_0 é fixo para todo j, mas pode variar para cada i, usando por exemplo (7.18).

Outros operadores de recombinação

Como discutido anteriormente, a recombinação discreta, seja ela do tipo binomial ou exponencial, não é invariante à rotação do sistema de coordenadas. Além disso, como somente algumas coordenadas são herdadas do vetor mutante, o vetor teste produzido é algum ponto obtido com passos ortogonais a partir da solução mutante, veja Figura 7.1 novamente.

A Figura 7.13 ilustra uma função multimodal com variáveis correlacionadas de tal forma que a bacia de atração $\mathcal{B}_2$ é alcançável a partir da bacia de atração $\mathcal{B}_1$ na Figura por meio de um movimento correlacionado em ambas as coordenadas. A recombinação discreta pode causar movimentos ortogonais em que apenas uma variável é modificada. Um movimento em uma única coordenada pode não alcançar

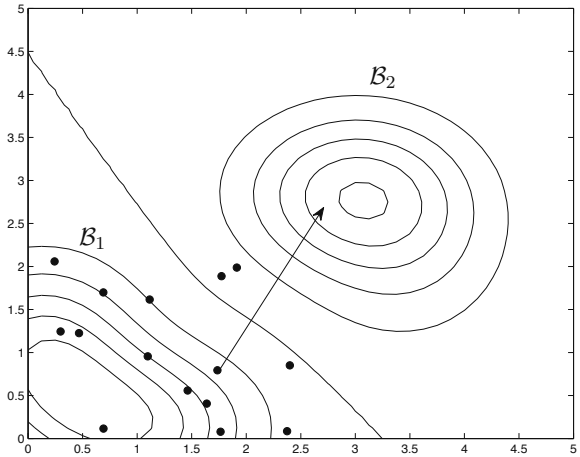


Figura 7.13: Função multimodal com variáveis correlacionadas.

a bacia de atração $\mathcal{B}_2$ e não ser aceito. Dessa forma, a bacia $\mathcal{B}_2$ não seria alcançável, a menos que um movimento seja executado em ambas as coordenadas simultaneamente e na direção adequada.

Em uma função convexa com variáveis correlacionadas, como aquela ilustrada na Figura 7.2, passos ortogonais diminuem a velocidade de convergência que poderia ser obtida com passos correlacionados. Em problemas multimodais, é interessante que os movimentos sejam executados simultaneamente em muitas ou em todas as variáveis. Por essa razão, em problemas convexos com variáveis correlacionadas e em problemas multimodais, o algoritmo de evolução diferencial usando recombinação discreta apresenta melhor desempenho quando C é próximo de 1. Para valores próximos de 1, vários valores da solução mutante são aceitos simultaneamente, permitindo movimentos correlacionados.

Outra forma de contornar o problema, é substituir a recombinação discreta por outros operadores de recombinação, incluindo operadores que executem passos em várias ou em todas as coordenadas simultaneamente e que sejam invariantes à rotação do sistema de coordenadas.

Um operador de recombinação simples que satisfaz esses requisitos é a recombinação aritmética, muito utilizada em algoritmos genéticos e estratégias evolutivas. A solução teste é obtida a partir de:

$$\mathbf{u}_{t,i} = \lambda \mathbf{x}_{t,i} + (1-\lambda)\mathbf{v}_{t,i}, \quad \lambda \in [0,1] \tag{7.24}$$

sempre que $\mathcal{U}_{[0,1]} \leq C$. Dessa forma, o parâmetro C passa a ter o significado de um parâmetro de taxa de recombinação da população atual X_t com a população mutante V_t.

A equação (7.24) pode ser combinada com (7.3) obtendo:

$$\mathbf{u}_{t,i} = \lambda \mathbf{x}_{t,i} + (1-\lambda)\left[\mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)\right], \quad \lambda \in [0,1] \tag{7.25}$$

que combina a mutação diferencial e a recombinação numa única equação.

Novamente, podemos escrever a equação para a geração do vetor teste na forma compacta a seguir:

$$\mathbf{u}_{t,i} = \begin{cases} \lambda \mathbf{x}_{t,i} + (1-\lambda)\left[\mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right)\right], & \text{se } \mathcal{U}_{[0,1]} \leq C \\ \mathbf{x}_{t,r_1} + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_3}\right), & \text{caso contrário} \end{cases} \tag{7.26}$$

em que a seleção para sobrevivência é feita como antes, ver (7.6).

Além da recombinação aritmética, outros operadores de recombinação propostos na literatura podem ser utilizados, veja (Gwiazda, 2006) para uma referência bastante ampla de operadores de recombinação para problemas de otimização numérica.

Esquema geral dos algoritmos de evolução diferencial

Vamos concluir esta seção com a apresentação de um esquema geral para as variações do algoritmo de evolução diferencial. Esse esquema geral é apresentado no Algoritmo 3.

Algoritmo 3 Pseudocódigo geral para a família de algoritmos de evolução diferencial

1: $t \leftarrow 1$
2: Inicializar população $X_t = \{\mathbf{x}_{t,i};\ i = 1, \ldots, N\}$
3: **enquanto** algum critério de parada não for satisfeito **faça**
4: **para** $i = 1$ até N **faça**
5: Selecionar vetor de base $\mathbf{x}_{t,base}$ [Seleção para reprodução]
6: Selecionar conjunto de vetores-diferença $\{\Delta\mathbf{x}_{t,k}\}$, $k = 1, \ldots, d$
7: Selecionar fatores de escala F_k, $k = 1, \ldots, d$
8: Gerar solução mutante usando $\mathbf{v}_{t,i} = \mathbf{x}_{t,base} + \sum_{k=1}^{d} F_k \Delta\mathbf{x}_{t,k}$
9: Adicionar $\mathbf{v}_{t,i}$ à população mutante V_t
10: **fim para**
11: Recombinar X_t e V_t, com parâmetro C, gerando U_t
12: Aplicar seleção para sobrevivência entre X_t e U_t
13: $t \leftarrow t + 1$
14: **fim enquanto**

Na linha 5 do Algoritmo 3, ocorre a seleção do vetor de base, que representa a seleção para reprodução no algoritmo de evolução diferencial. Nas duas linhas seguintes, são selecionados d vetores-diferença e os valores para os parâmetros de escala de cada vetor-diferença, que serão usados na geração do vetor mutante na linha 8.

O laço nas linhas 4 a 10 está relacionado somente à operação de mutação diferencial para produção da população mutante V_t. Após a operação de mutação diferencial, a recombinação é executada na linha 11. Finalmente, deve-se definir um mecanismo de seleção para sobrevivência entre X_t e U_t, formando a população X_t da próxima geração.

Apresentado dessa maneira, podemos observar que a mutação diferencial é um operador de busca que tem um papel primário no algoritmo, enquanto a recombinação tem um papel secundário.

O algoritmo de evolução diferencial básico apresentado no Algoritmo 1 e as variações discutidas neste capítulo podem ser vistos como instâncias particulares do Algoritmo 3, mais geral.

5. Conclusões

Este capítulo apresentou os algoritmos baseados em evolução diferencial para otimização de funções contínuas. Ultimamente, a evolução diferencial tem recebido bastante destaque no contexto da otimização não linear com variáveis contínuas, devido às suas características de versatilidade, robustez e autoadaptação, colocando-o entre os algoritmos evolutivos mais eficientes nesse contexto.

O algoritmo de evolução diferencial possui muitas qualidades desejáveis em algoritmos de otimização de propósito geral: (i) capacidade de adaptar-se à estrutura da função, aprendendo correlações lineares entre as variáveis do problema a partir da distribuição espacial da população; (ii) invariância à rotação e translação do sistema de coordenadas; (iii) poucos parâmetros de controle a serem ajustados, sendo que o parâmetro de escala não precisa ser necessariamente ajustado, uma vez que os tamanhos de passo da mutação são autoadaptados; e (iv) simplicidade de implementação.

Embora o algoritmo de evolução diferencial tenha sido desenvolvido para tratar problemas de otimização não linear contínua, pode-se encontrar na literatura, em especial nos últimos anos, alguns trabalhos que visam adaptar o algoritmo para problemas de otimização combinatória. Por exemplo,

Qian et al. (2008) apresenta uma versão do algoritmo de evolução diferencial para problemas de planejamento e escalonamento da produção. O leitor interessado também pode consultar o recente livro editado por Onwubolu e Davendra (2009), que discute várias abordagens de utilização da evolução diferencial em problemas de otimização baseados em vetores de permutações.

Agradecimentos

O autor gostaria de expressar sua gratidão ao aluno de doutorado Lucas de Souza Batista e aos colegas Felipe Campelo e Jaime Arturo Ramírez pelos comentários e sugestões que contribuiram para o formato final deste texto.

(Página deixada propositadamente em branco)

CAPÍTULO 8

Recozimento Simulado

Marconi A. Pereira *João A. Vasconcelos*

Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais

Recozimento Simulado[1] é uma metaheurística inspirada no processo físico de recozimento de um sólido para obtenção de estados de baixa energia na área da física da matéria condensada. Esse processo consiste em aquecer o sólido até atingir sua temperatura de fusão, para que a matéria passe do estado sólido para o líquido; posteriormente, a temperatura deve ser lentamente diminuída para evitar estados meta-estáveis. O objetivo deste processo é obter a matéria no estado cristalino, ou seja, com energia mínima.

No estado líquido a matéria possui grande quantidade de energia e suas partículas são dispostas aleatoriamente. Em contrapartida, no estado sólido cristalizado as partículas são dispostas de forma extremamente estruturada, com mínima energia. É muito importante que o resfriamento para a passagem do estado líquido para o sólido ocorra de maneira lenta e cuidadosa, pois do contrário, a matéria pode parar em um estado sólido intermediário e não mais se cristalizar. A Figura 8.1 mostra um esquema do recozimento.

O conceito de recozimento (annealing) foi introduzido em otimização combinatória na década de 80 por Kirkpatrick et al. (1982) e Kirkpatrick (1984). Assim como o recozimento procura levar a matéria

[1] Do inglês *simulated annealing*. No Brasil essa expressão vem sendo traduzida como *recozimento simulado*, guardando o significado original da expressão em língua inglesa, que faz referência ao processo metalúrgico de *recozimento* de metais. Em Portugal, a expressão vem sendo traduzida como *arrefecimento simulado*.

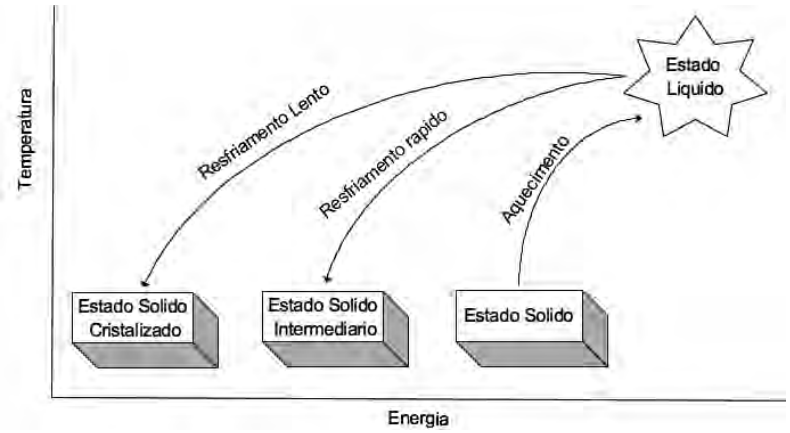


Figura 8.1: Recozimento. A matéria é aquecida (bloco à direita) e depois resfriada lentamente (bloco à esquerda). Um resfriamento rápido leva a matéria para um estado sólido de energia intermediária (bloco central).

para um estado de energia mínima, a otimização consiste em encontrar uma solução que minimize uma determinada função objetivo. O problema de otimização pode ser escrito de forma genérica como:

$$\begin{aligned} &\text{minimize } \overline{f}(\overline{x}) \\ &\text{sujeito a: } \begin{cases} \overline{g}(\overline{x}) \leq \overline{0} \\ \overline{h}(\overline{x}) = \overline{0} \end{cases} \end{aligned} \tag{8.1}$$

onde $\overline{x} \in \mathbb{R}^n$, $\overline{f}\colon \mathbb{R}^n \to \mathbb{R}^m$, $\overline{g}\colon \mathbb{R}^n \to \mathbb{R}^k$, $\overline{h}\colon \mathbb{R}^n \to \mathbb{R}^l$, com n, m, k e $l \in \mathbb{N}$. Se $n = 1$, o problema é mono-objetivo. Caso contrário, ele é multiobjetivo.

Na década de 50, Metropolis publicou um trabalho (Metropolis et al. (1953)) no qual o processo físico de recozimento foi modelado em um algoritmo que simula o resfriamento de um sólido em direção ao equilíbrio térmico. Esse algoritmo é baseado em técnicas de Monte Carlo (Gilks et al. (1995)), onde é gerada uma sequência de estados do sólido obtidos da seguinte maneira:

- Aplica-se uma perturbação em um estado corrente i do sólido cuja energia atual é E_i, para gerar um novo estado j. A energia desse novo estado é E_j.
- Se a diferença de energia $E_j - E_i$ for menor ou igual a zero, o estado j é aceito.
- Se a diferença é maior que zero, o estado j é aceito de acordo com a seguinte probabilidade:

$$p = \exp\left(\frac{E_i - E_j}{k_b T}\right), \tag{8.2}$$

onde T denota a temperatura e k_b é uma constante física chamada Constante de Boltzmann. Esse critério de aceitação do estado j é conhecido como critério de Metropolis.

A temperatura deve diminuir lentamente para que o equilíbrio térmico seja alcançado. O algoritmo de Metropolis deve gerar, portanto, um grande número de transições a fim de atingir esse equilíbrio. As transições geradas caracterizam uma distribuição de Boltzmann (Aarts e Korst (1989)). Analisando a equação 8.2 percebe-se que a probabilidade de aceite do estado é alta quando o valor da temperatura é alto. À medida em que a temperatura diminui, a probabilidade de aceite do novo estado também

Processo de Recozimento	Recozimento Simulado
Aplicado na busca pelo Equilíbrio Térmico da Matéria	Visa a Otimização de uma função
Variação energética	Variação da função objetivo
Procura determinar o estado de mínima energia	Procura determinar o valor mínimo da função objetivo
Baseado na diminuição da temperatura (T)	Baseado na diminuição do parâmetro de controle (c)
Pode parar em estados intermediários de energia da matéria, não atingindo o estado de mínima energia	Pode parar em mínimos locais da função objetivo, não atingindo o mínimo global

Tabela 8.1: Analogia entre o processo de recozimento e o algoritmo de recozimento simulado

diminui. De acordo com Aarts e Korst (1989), a probabilidade do sólido estar no estado i com energia E_i e temperatura T é dada por:

$$P_T\{X = i\} = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{k_b T}\right), \tag{8.3}$$

onde X é uma variável estocástica que indica o estado corrente do sólido e $Z(T)$ é a função de partição definida por:

$$Z(T) = \sum_j \exp\left(\frac{-E_i}{k_b T}\right), \tag{8.4}$$

onde o somatório cobre todos os estados possíveis.

Conforme informado anteriormente, o algoritmo de Metropolis é a base do recozimento simulado. A implementação do recozimento simulado será detalhada na seção seguinte.

1. Implementação do Recozimento Simulado

O recozimento simulado utiliza o algoritmo de Metropolis para simular o equilíbrio térmico. Assim, pode-se assumir uma analogia entre o processo físico e o processo de otimização combinatória baseada nas seguintes equivalências (Aarts e Korst (1989)):

- Soluções intermediárias de um problema de otimização combinatória equivalem às etapas de resfriamento da matéria;
- O valor da função objetivo do problema de otimização equivale à energia do sólido no processo físico de resfriamento em direção ao equilíbrio térmico.

A Tabela 8.1 mostra uma analogia entre o processo de recozimento e a aplicação do recozimento simulado em problemas de otimização.

O recozimento simulado assume a existência de uma vizinhança, que pode ser encontrada pelo mecanismo de busca. Assim, são utilizadas as seguintes definições (Aarts e Korst (1989)):

Definição 8.1 - (S, f) *denota uma amostra de um problema de otimização e* i *e* j *duas soluções com custo* $f(i)$ *e* $f(j)$*, respectivamente. Assim, o critério de aceite determina quando* j *é aceito em substituição a* i *pela aplicação da probabilidade de aceite descrita na equação 8.5.*

$$P_c\{\text{aceite}j\} = \begin{cases} 1 & \text{se } f(j) \leq f(i) \\ \exp\left(\frac{f(i)-f(j)}{c}\right) & \text{se } f(j) > f(i) \end{cases} \tag{8.5}$$

onde $c \in \mathbb{R}^+$ denota o parâmetro de controle.

Definição 8.2 - *Uma transição é uma ação que resulta na transformação de uma solução corrente em uma solução subsequente. Essa ação consiste nas seguintes etapas: (a) aplicação de uma perturbação, ou seja, mecanismo de geração de um novo estado; (b) aplicação do critério de aceitação.*

Seja c_k o valor do parâmetro de controle e L_k o número de transições geradas na k-ésima iteração do Algoritmo de Metropolis. Com estas definições, o pseudocódigo do SA pode ser apresentado (Algoritmo 1).

Algoritmo 1 Pseudocodigo para o algoritmo de recozimento simulado

```
1: INICIALIZA(i_inicial, c_0, L_0);
2: k ← 0;
3: i ← i_inicial;
4: enquanto não se atinge o critério de parada faça
5:    para l ← 1 to L_K faça
6:       GERE (j à partir de S_i);
7:       se f(j) ≤ f(i) então
8:          i ← j;
9:       senão
10:         se exp((f(i)−f(j))/c_k) > random[0,1) então
11:            i ← j;
12:         fim se
13:      fim se
14:   fim para
15:   k ← k + 1;
16:   CALCULA_TAMANHO(L_k);
17:   CALCULA_CONTROLE(c_k);
18: fim enquanto
```

Além de aceitar redução (minimização) na função de custo (linhas 7 e 8), o recozimento simulado pode aceitar uma deterioração (aumento) na função, com uma certa probabilidade (definição 1 implementada nas linhas 10 e 11 do algoritmo 1). Nas primeiras iterações do algoritmo, essa aceitação será maior, pois os valores de c_k serão maiores. À medida em que o algoritmo prossegue, os valores de c_k irão diminuindo (linha 17), fazendo com que a probabilidade de aceitação de deteriorações na função de custo também diminua. O fato do algoritmo aceitar algumas deteriorações na função de custo faz com que ele escape de mínimos locais.

O recozimento simulado utiliza alguns parâmetros que podem influir diretamente na eficiência do algoritmo: Valor inicial do parâmetro de controle (c_k), taxa de diminuição deste valor e o número de ciclos de temperatura (L_k) sobre cada valor de temperatura. Estes parâmetros precisam ser escolhidos corretamente para cada tipo de problema, pois podem ser determinantes para se obter um ótimo global. Esses valores devem ser escolhidos de acordo com o problema o qual se deseja otimizar. Versões mais recentes do algoritmo proporcionam formas de se escolher bons valores para esses parâmetros.

O valor de c_k, geralmente, é decrementado da seguinte maneira:

$$c_{k+1} = \alpha.c_k, k = 1, 2, \ldots \tag{8.6}$$

onde α é uma constante menor que 1, geralmente variando entre 0.8 e 0.99.

A quantidade de perturbações (L_k), em cada valor do parâmetro de controle (c_k), pode estar relacionada a uma combinação de diversos fatores, tais como o número máximo de iterações, taxa mínima de queda de energia, entre outros. O importante é que se atinja o equilíbrio térmico para cada valor de c_k.

2. Aplicações

O algoritmo de recozimento simulado é aplicado com grande sucesso em problemas de otimização combinatória: Problemas discretos, como por exemplo, o problema do caixeiro viajante e projeto de equipamentos eletrônicos, além de problemas contínuos como problemas de projeto das formas geométricas de equipamentos, são resolvidos com recozimento simulado. A seguir, será descrito como estes problemas podem ser modelados.

Problemas Discretos

O problema do caixeiro viajante consiste em encontrar o menor caminho que percorre todas as cidades de um dado conjunto. Formalmente pode-se definir esse problema da seguinte maneira:

Definição 8.3 - *Problema do Caixeiro Viajante: Seja $C = c_1, \ldots, c_n$ um conjunto de n cidades e $D = [d_{ij}]$ a matriz de adjacência cujo elemento d_{ij} denota a distância entre a cidade i e a cidade j. O problema consiste em encontrar o menor caminho no qual todas as cidades são visitadas.*

O problema do caixeiro viajante pode ser resolvido da seguinte maneira, utilizando o algoritmo recozimento simulado (Aarts e Korst (1989)):

- O espaço de busca das soluções S corresponde ao conjunto que contem todas as possíveis permutações entre caminhos que ligam todas as cidades. Cada caminho é dado por $\pi = (\pi(1), \ldots, \pi(n))$, onde $\pi(i), i = 1, \ldots, n$ denota a cidade sucessora da cidade i. A cidade número 1 é a sucessora da n-ésima cidade.
- Gera-se um estado inicial ($i_{inicial}$) que corresponde a um caminho entre as cidades, gerado aleatoriamente; considere $c_0 = 0.10 * n$, onde n é o número total de cidade e $L_k = 10 * n$;
- A função de custo a ser minimizada é dada por :

$$f(\pi) = \sum_{i=1}^{n} d_{i,\pi(i)} \tag{8.7}$$

- A perturbação consiste em gerar novas soluções (caminhos) escolhendo aleatoriamente duas cidades e invertendo suas posições na sequência de cidades a serem visitadas;
- A diferença em custo entre duas cidades p e q, que permutaram suas posições, pode ser calculada de maneira incremental a partir da seguinte expressão:

$$\Delta f = -a - b - c - d + a' + b' + c' + d' \tag{8.8}$$

onde a e b são as distâncias entre a cidade p e sua antecessora e sucessora respectivamente, c e d são as distâncias entre a cidade q e sua antecessora e sucessora; a' e b' são as distâncias entre a cidade q e sua nova antecessora e nova sucessora respectivamente e c' e d' são as distâncias entre a cidade p e sua nova antecessora e nova sucessora. A Figura 8.2 mostra um exemplo de permutação de cidades.

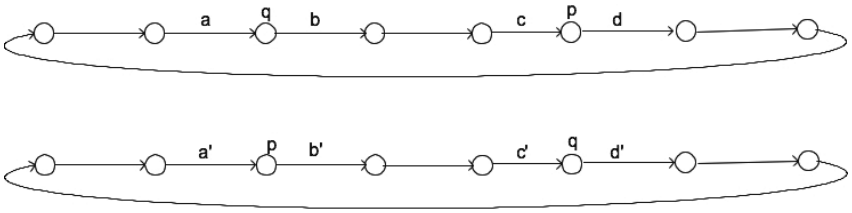


Figura 8.2: Problema do Caxeiro Viajante. O SA permuta o caminho entre duas cidades a fim de procurar a menor rota.

Um outro exemplo de aplicação citado é o problema de projeto de equipamentos. Trata-se de um problema bastante complexo, que na prática pode ter de 10^3 a 10^5 variáveis, além de vários objetivos conflitantes a serem otimizados. Um exemplo simples deste tipo de problema é mostrado em (Rutenbar (1989)): Deseja-se projetar a disposição dos componentes de um circuito integrado com o objetivo de otimizar o cabeamento (*wirability*) entre estes. O circuito é modelado como uma malha retangular, onde cada módulo deve estar em um dos nós da malha retangular e deve ter no máximo quatro terminais elétricos. Um conjunto de módulos é interligado, compondo uma rede, e posteriormente essas redes são interligadas de modo a compor o circuito como um todo. Devem-se posicionar os módulos das redes para otimizar o cabeamento.

A modelagem do problema é feita da seguinte forma:

- Identifica-se o modelo que define o espaço de busca das soluções, ou seja, as soluções viáveis. Em suma, deve-se identificar as restrições do problema;
- A perturbação em uma dada configuração i consiste em realizar uma alteração possível (que atenda às restrições do problema) da posição de um módulo da rede. O mais comum seria a troca de posição entre dois módulos. Para cada temperatura são executadas $100M$ perturbações, onde M é o número de módulos do circuito;
- A função de custo consiste no cálculo do comprimento total dos fios necessários para interligar o circuito. Em situações reais, outras variáveis poderiam ser consideradas, como por exemplo, o congestionamento do cabeamento (*wiring congestion*);
- O critério de parada do processo consiste na identificação de três quedas sucessivas de temperatura sem que a função de custo decaia em mais de 1%.

A Figura 8.3 mostra a conexão lógica dos módulos que compõem o circuito integrado. A Figura 8.4 mostra uma possível configuração (design) que implementa o circuito em questão.

Considerando a execução do problema de design do circuito integrado conforme descrito anteriormente, utilizando aproximadamente 800 módulos, obtém-se um gráfico de custo em função da temperatura (parâmetro de controle) conforme disposto na Figura 8.5.

Ao longo do processo, a temperatura é diminuída (da direita para a esquerda) ao mesmo tempo que o custo da função objetivo também diminui. Em alguns pontos há um pequeno acréscimo na função custo, mas posteriormente a função volta a cair. Esse comportamento contribui para que o algoritmo saia de mínimos locais da função objetivo.

Como o recozimento simulado é um processo estocástico, o resultado final de uma execução pode ser diferente do resultado obtido em outra. A Figura 8.6 mostra os resultados finais obtidos em 20 execuções do recozimento simulado. A figura mostra ainda uma linha informando a média e a variância (σ) dos valores finais.

Um exemplo real de concepção da planta baixa (*floorplannig*) de um chip é detalhado em (Rutenbar (1989)).

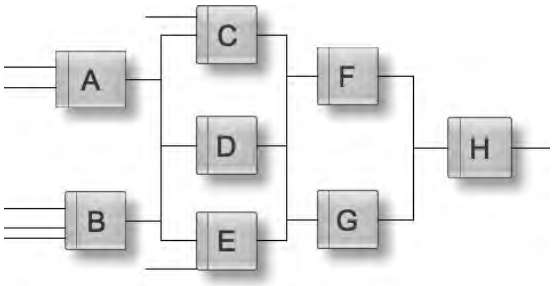


Figura 8.3: Conexão Lógica dos Módulos (Rutenbar (1989)).

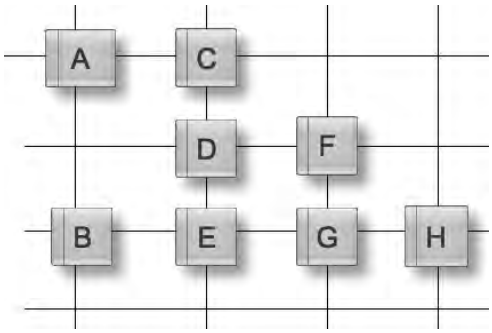


Figura 8.4: Configuração física do circuito na forma de grid (Rutenbar (1989)).

Problemas Contínuos

O recozimento simulado também é aplicado com sucesso em problemas cujas variáveis são contínuas. Uma das primeiras formas de se abordar este tipo de problema foi apresentada em Corana et al. (1987). Posteriormente, Vasconcelos et al. (1996) apresentaram uma melhora no primeiro algoritmo, acrescentando estratégias de busca tabu. Este capítulo irá detalhar essa última versão do algortimo, por se tratar de uma versão mais robusta e eficiente.

Seja $x \in \mathbb{R}^n$ um vetor com os seguintes componentes $x_1, \ldots, x_n$. Seja $f(x)$ a função a ser minimizada, onde $a_1 < x_1 < b_1, \ldots, a_n < x_n < b_n$ são suas n variáveis com os respectivos intervalos contínuos e finitos. A função $f(x)$ não precisa necessariamente ser continua, porém possui limites. O algoritmo apresentado por Vasconcelos et al. (1996) é descrito no Algoritmo 2.

Os parâmetros e variáveis utilizados no algoritmo são detalhados na tabela 8.2.

A implementação do recozimento simulado para problemas com variáveis contínuas é muito semelhante à versão para problemas com variáveis discretas. A diferença está basicamente em duas etapas:

- Atualização dos pontos. A versão para problemas de variáveis contínuas utiliza, além do ponto anterior(x_i) e de um valor randômico (dentro do intervalo $[-1, 1]$), mais um elemento: a i-ésima dimensão do vetor H.

- Atualização do vetor de direções. O parâmetro C_i controla a variação na direção u. Resultados práticos mostram que um bom valor para esse parâmetro é $C_i = 0.2, i = 1, 2, \ldots, n$.

No trabalho apresentado por Vasconcelos et al. (1996), o recozimento simulado foi implementado a fim de minimizar as funções Exponencial (8.9) e Rastrigin (8.10). Essas funções, dentre outras,

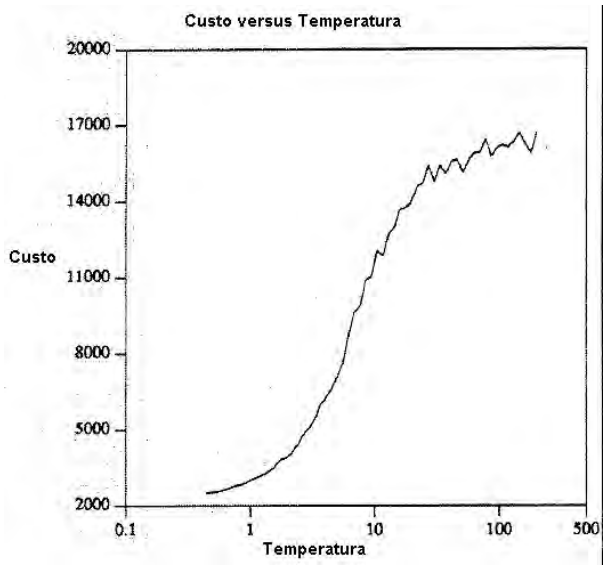


Figura 8.5: Gráfico do custo versus temperatura (Rutenbar (1989)).

são muito utilizadas em testes de algoritmos de otimização com variáveis discretas, uma vez que apresentam diversos mínimos locais. Um bom algoritmo de otimização deve ser capaz de escapar desses mínimos locais de forma a se aproximar do valor mínimo global. A função Exponencial foi definida como:

$$f(x) = 20 - e^{-\sum(x_i-1.5)^2+1} + e^{-\sum(x_i-2.5)^2+1.05} - e^{-\sum(x_i-3.5)^2+1.1} \tag{8.9}$$

onde $0 \leq x_i \leq 10 \quad i = 1, 2.$

Essa função possui dois mínimos locais. O mínimo global é o ponto $x^* = 3.59585; 3.59585)^T$. O valor da função nesso ponto é $f(x^*) = 17.30889$.

A função Rastrigin é dada por:

$$f(x) = 10n + \sum_{i=1}^{n}[(x_i)^2 - 10\cos(2\pi)(x_i)] \tag{8.10}$$

onde $n \in \mathbb{N}$ é o número de variáveis, $\overline{x} \in \mathbb{R}^n$ e $x_i \in [-5.12; 5.12] \quad \forall i = 1, 2, \ldots, n.$

A função Rastrigin é continua e multimodal. Para $x_i \in [-5.12; 5.12]$ e $n = 10$, existem 10^n minimos locais e um mínimo global em $x^* = (0; \ldots; 0)^T$ onde $f(x^*) = 0$.

A figura 8.7 mostra um gráfico das funções Exponencial e Rastrigin.

O experimento apresentado em Vasconcelos et al. (1996) utilizou como critério de parada o alcance de um valor mínimo da temperatura. Os resultados práticos mostraram que o algoritmo apresentado obteve melhores resultados que a implementação clássica do recozimento simulado.

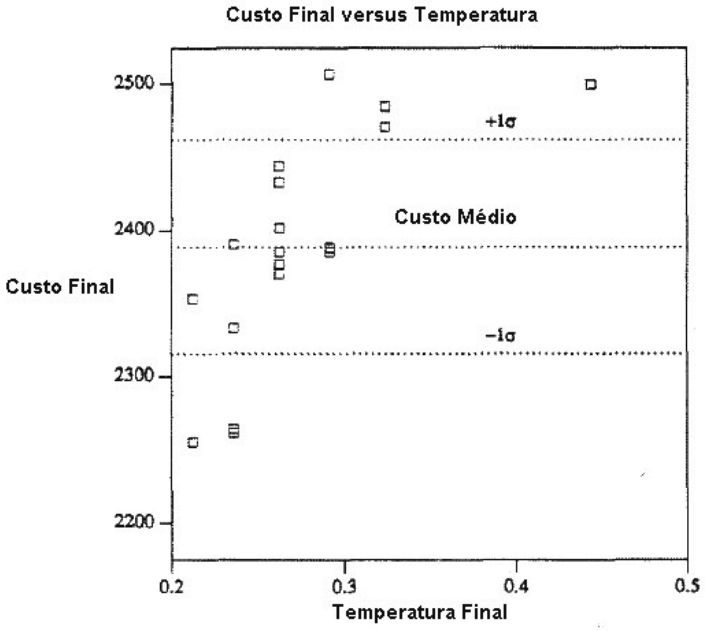


Figura 8.6: Resultados obtidos em 20 execuções do recozimento simulado para o projeto do circuito integrado (Rutenbar (1989)).

3. Recozimento Simulado Multiobjetivo

A grande maioria dos problemas reais, possui muitos objetivos que são geralmente conflitantes entre si. Desta forma, algumas versões do recozimento simulado multiobjetivo têm sido propostas na literatura. Nessa seção será mostrada uma das versões mais recentes e eficientes. Estudos mais detalhados sobre Simulted Annealing multiobjetivo podem ser encontrados em (Ehrgott e Gandibleux (2000)), (Smith et al. (2004)), (Suman (2004)) e (Bandyopadhyay et al. (2008)).

A grande dificuldade em se obter uma versão eficiente multiobjetivo do recozimento simulado está no fato de que esse algoritmo gera somente um único ponto por execução, enquanto que algoritmos multiobjetivo devem idealmente gerar uma população de pontos não-dominados, que estejam espalhados pelo conjunto e que sejam próximos à fronteira Pareto Ótima. Para maiores detalhes sobre otimização multiobjetivo, vide o capítulo 17.

A solução encontrada para o problema de se gerar um conjunto de pontos no recozimento simulado consiste em armazenar os valores intermediários obtidos durante a execução da ferramenta em um conjunto de pontos não-dominados. Czyzak e Jaszkiewicz (1998) apresentaram uma primeira versão deste algoritmo e posteriormente Bandyopadhyay et al. (2008) apresentaram uma versão mais eficiente, cujos resultados são comparados com algoritmos consolidados na literatura, como o NSGA-II (Deb et al. (2002)) e PAES (Knowles e Corne (2000a)). Uma versão do recozimento simulado multiobjetivo é apresentada no Algoritmo 3, extraído de (Bandyopadhyay et al. (2008)).

Os parâmetros utilizados no Algoritmo 3 além das principais variáveis são detalhados na tabela 8.3.

O algoritmo apresentado em Bandyopadhyay et al. (2008) é chamado AMOSA (*Archived Multi-objective Simulated Annealing*), pois procura, a cada iteração, identificar os pontos não-dominados e armazená-los. É feita uma clusterização nesse conjunto de pontos não-dominados a fim de forçar a

Tabela 8.2: Parâmetros/variáveis e respectivos significados do recozimento simulado para problemas com variáveis contínuas.

Parâmetro / Variável	**Significado**
NT	Número de passos
ND	Número de ciclos auxiliares
LIM	Limite de movimentos com sucesso
j	j-ésimo ciclo auxiliar
i	i-ésima variável contínua
u_i	u-ésimo sucesso da i-ésima variável
TEMP	Temperatura
p_0	Melhor solução
C_i	$C_i \approx 0.2$
u	$u = \frac{\sum_i u_i}{\sum_i NT_i}$

Tabela 8.3: Parâmetros e respectivos significados do recozimento simulado para problemas multiobjetivo

Parâmetro / Variável	**Significado**
T_{max}	Temperatura máxima. Corresponde ao valor inicial da temperatura.
T_{min}	Temperatura mínima. Valor de referência para teste de convergência.
Archive	Conjunto de tamanho variável que irá armazenar os pontos durante a execução do SA. Esse conjunto deve ser inicializado com pontos no espaço de busca, gerados aleatoriamente. Ao final, esse conjunto deverá conter os pontos não-dominados obtidos.
HL	Tamanho máximo do conjunto *Archive* ao final da execução do algoritmo. Esse valor corresponde ao número de soluções não dominadas a serem encontradas.
SL	Tamanho máximo do conjunto *Archive* antes de ser aplicada a operação de clusterização de forma que o conjunto *Archive* tenha tamanho HL.
iter	Número máximo de iterações para cada temperatura.
α	Taxa de redução da temperatura.
Δdom_{min}	Variável referente ao menor valor de intensidade de dominância encontrado. Dado um subconjunto de k pontos, é calculado a intensidade de dominância entre *new_pt* e cada um desses k pontos, utilizando a equação 8.11. O menor desses valores é atribuído à variável Δdom_{min}.

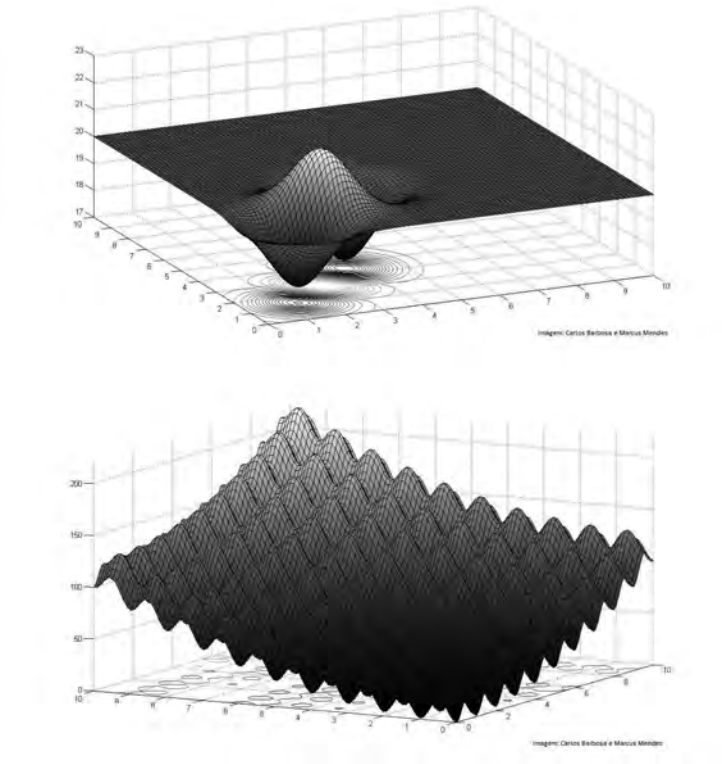


Figura 8.7: Função Exponencial e Rastrigin respectivamente.

diversidade das soluções. Durante a execução da ferramenta, o número de pontos não-dominados aumenta. Quando a quantidade de pontos atinge um valor ($|SL|$), é realizada a clusterização, utilizando-se $|HL|$ clusteres. O algoritmo de clusterização utilizado é Single Linkage (Jain e Dubes (1988)).

O algoritmo AMOSA utiliza um conceito a fim de mensurar um valor referente ao quanto uma solução é dominante sobre a outra. Esse conceito é chamado de *intensidade de dominância* e é definido da seguinte forma:

Definição 8.4 - *Sejam a e b duas soluções viáveis para um problema P. A intensidade de dominância é dada pela equação:*

$$\Delta dom_{a,b} = \prod_{i=1, f_i(a) \neq f_i(b)}^{M} \left(\frac{|f_i(a) - f_i(b)|}{R_i} \right), \tag{8.11}$$

onde M é o número de objetivos e R_i é o intervalo do i-ésimo objetivo.

O AMOSA procura por pontos que possuam uma grande intensidade de dominância. Entretanto, assim como o recozimento simulado mono-objetivo, nas primeiras iterações do algoritmo (enquanto a temperatura ainda está alta), são aceitos pontos com baixa intensidade de dominância e até mesmo pontos não-dominados. Contudo, à medida em que a temperatura vai caindo, a probabilidade de se aceitar um estado pior diminui.

Além de possuir uma complexidade computacional similar à complexidade dos algoritmos mais eficientes encontrados na literatura (vide tabela 8.4), resultados práticos mostraram que o AMOSA é

capaz de gerar resultados competitivos, considerando como medidas de comparação a:

- Convergência - Medida da distância entre os valores obtidos e a Fronteira Pareto Ótima obtida;
- Pureza - Fração calculada com a comparação entre o resultado obtido pelo algoritmo e a combinação dos resultados obtidos por diferentes algoritmos. Quanto mais próximo de 1, melhor é a performance; quanto mais próximo de 0, mais fraca é a performance do algoritmo em questão;
- Espaçamento - Medida da distribuição dos pontos não-dominados obtidos dentro da fronteira. Quanto maior for o espalhamento entre os pontos, melhor o resultado.

Tabela 8.4: Complexidade dos Algoritmos Multiobjetivos

Algoritmo	**Complexidade**
NSGA-II	$O(iter \times M \times N^2)$
PAES	$O(iter \times M \times N)$
AMOSA	$O(iter \times N \times (M + \log(N)))$

Na tabela 8.4 *iter* equivale ao número máximo de iterações (parâmetro que deve ser fornecido para o algoritmo); M é o número de objetivos e N é o tamanho da população. No caso do AMOSA, $N = |HL|$.

4. Conclusões

Esse capítulo apresentou o recozimento simulado, desde sua primeira versão mono-objetivo até a versão mais recente, multiobjetivo, destacando a sua aplicação em problemas discretos e contínuos. A simplicidade dessa ferramenta fez com que se tornasse bastante popular desde seu surgimento na comunidade científica, sendo utilizada em diversas aplicações.

Durante alguns anos, quando a comunidade de otimização se voltava mais para o estudo sobre problemas com múltiplos objetivos, o recozimento simulado foi pouco utilizado, uma vez que as primeiras versões multiobjetivo desse algoritmo não produziam um resultado com boa qualidade. A dificuldade inicial do recozimento simulado estava no fato de que só se trabalhava com um ponto por iteração, enquanto nas demais ferramentas Evolucionárias se trabalhava com uma população de pontos. A inserção da idéia de se usar um arquivo para armazenar os pontos não-dominados intermediários, bem como critérios para aumentar a diversidade (Pureza e Espaçamento) fez com que o recozimento simulado voltasse à pauta dos temas pesquisados como uma ferramenta robusta e eficiente para se resolver problemas de otimização.

Algoritmo 2 Pseudocódigo para o algoritmo recozimento simulado contínuo

1: INICIALIZA($NUCICL$, LIM, $TEMP$, x_0);
2: $Ind \leftarrow 0$; $H = \{h_1, h_2, \ldots, h_n\}$; $f_0 \leftarrow f(x_0)$;
3: $x'_i \leftarrow x'_i + rh_i$; $f' \leftarrow f(x')$; $NT_i \leftarrow NT_i + 1$;
4: **se** $f' \leq f$ **então**
5: $x \leftarrow x'$; $f \leftarrow f'$; $u_i \leftarrow u_i + 1$; $i \leftarrow i + 1$;
6: **se** $i > nd$ **então**
7: $i \leftarrow 1$;
8: **fim se**
9: **se** $f' < f_0$ **então**
10: $p_0 \leftarrow x'$; $f_0 \leftarrow f'$;
11: **fim se**
12: **senão**
13: {Execução do algoritmo de Metropolis}
14: Gere uma perturbação;
15: **se** Novo estado for aceito **então**
16: goto linha 3;
17: **fim se**
18: **fim se**
19: **se** $j < 10nd$ **então**
20: **se** $j > 2nd \wedge u \in [0.4; 0.6]$ **então**
21: goto linha 29;
22: **senão**
23: $j \leftarrow j + 1$;
24: **se** $Ind = 0 \wedge j \in [k, 2k, 3k]$ **então**
25: **se** $\frac{\sum_i u_i}{\sum_i NT_i} < 0.3$ **então**
26: Aumentar TEMP;
27: **fim se**
28: **se** $\frac{\sum_i u_i}{\sum_i NT_i} > 0.3$ **então**
29: Reduzir TEMP;
30: **fim se**
31: **senão**
32: goto linha 3;
33: **fim se**
34: **fim se**
35: **fim se**
36: **se** $\frac{u_i}{Nt_i} > 0.6$ **então**
37: $h_i \leftarrow h_i[1 + \frac{C_i(p_i - 0.6)}{0.4}]$;
38: **fim se**
39: **se** $\frac{u_i}{Nt_i} < 0.4$ **então**
40: $h_i \leftarrow h_i[1 + \frac{C_i(0.4 - p_i)}{0.4}]^{-1}$;
41: **fim se**
42: Reduzir TEMP; $Ind \leftarrow Ind + 1$;
43: **se** Atingiu Critério de Parada **então**
44: Fim do Algoritmo;
45: **senão**
46: $x \leftarrow p_0$; $f \leftarrow f_0$;
47: goto linha 3;
48: **fim se**

Algoritmo 3 Recozimento Simulado Multiobjetivo

INICIALIZA (T_{max}, T_{min}, HL, SL, $iter$, α, $temp$, $Archive$);
$current\_pt \leftarrow random(Archive)$;
enquanto $temp > T_{min}$ **faça**
  para $i \leftarrow 0$; $i < iter$; $i++$ **faça**
    $new\_pt \leftarrow$ perturb($current\_pt$);
    se $current\_pt$ domina new_pt **então** {Caso 1}
      $\Delta dom_{avg} = \frac{(\sum_{i=1}^{k} \Delta dom_{i,new\_pt}) + \Delta dom_{current\_pt,new\_pt}}{k+1}$; {$k$ = numero total de pontos em $Archive$ que dominam new_pt, $k \geq 0$}
      $prob = \frac{1}{1+\exp(\Delta dom_{avg} * temp)}$;
      Aceite new_pt como $current\_pt$ de acordo com a probabilidade $prob$;
    fim se
    se $current\_pt$ e new_pt não dominam um ao outro **então** {Caso 2}
      Verifique a dominância de new_pt em relação aos pontos em $Archive$
      se new_pt é dominado por $k, (k \geq 1)$ pontos de $Archive$ **então** {Caso 2a}
        $prob = \frac{1}{1+\exp(\Delta dom_{avg} * temp)}$; $\Delta dom_{avg} = \frac{\sum_{i=1}^{k} \Delta dom_{i,new\_pt}}{k}$;
        Aceite new_pt como $current\_pt$ de acordo com a probabilidade $prob$;
      fim se
      se new_pt é dominado por todos os pontos em $Archive$ **então** {Caso 2b}
        $new\_pt \leftarrow current\_pt$; Adicione new_pt a $Archive$;
        Se $Archive\_size > SL$, clusterize $Archive$ com HL clusteres;
      fim se
      se new_pt domina $k, (k \geq 1)$ pontos de $Archive$ **então** {Caso 2c}
        $new\_pt \leftarrow current\_pt$; Adicione new_pt a $Archive$; Remova os k pontos dominados de $Archive$;
      fim se
    fim se
    se new_pt domina $current\_pt$ **então** {Caso 3}
      Verifique a dominância de new_pt em relação aos pontos em $Archive$
      se new_pt é dominado por $k, (k \geq 1)$ pontos de $Archive$ **então** {Caso 3a}
        Δdom_{min} = MIN(diferença de dominância entre new_pt e os k pontos);
        $prob = \frac{1}{1+\exp(-\Delta dom_{min})}$;
        $current\_pt \leftarrow$ ponto que corresponde a Δdom_{min} com probabilidade = $prob$. Caso contrário, $new\_pt \leftarrow current\_pt$;
      fim se
      se new_pt é dominado por todos os pontos em $Archive$ **então** {Caso 3b}
        $new\_pt \leftarrow current\_pt$; Adicione new_pt a $Archive$;
        caso $current\_pt$ está em $Archive$, remova-o ponto, Senão, caso $Archive\_size > SL$, clusterize $Archive$ com HL clusteres;
      fim se
      se new_pt domina k pontos de $Archive$ **então** {Caso 3c}
        $new\_pt \leftarrow current\_pt$; Adicione new_pt a $Archive$; Remova os k pontos de $Archive$;
      fim se
    fim se
  fim para
  $temp = \alpha * temp$;
fim enquanto
Se $Archive\_size > SL$, clusterize $Archive$ com HL clusteres;

CAPÍTULO 9

Busca Tabu

Marcone J. F. Souza

Departamento de Computação
Universidade Federal de Ouro Preto

A Busca Tabu (BT), conhecida na literatura inglesa como *Tabu Search*, é uma metaheurística de busca local originada nos trabalhos independentes de Fred Glover (Glover, 1986) e Pierre Hansen (Hansen, 1986). Gendreau (2003) e Glover e Laguna (1997) apontam, no entanto, que muitos dos componentes presentes nesta primeira proposta da Busca Tabu, bem como outros que foram incorporados posteriormente, já haviam sido introduzidas em Glover (1977).

A BT surgiu como uma técnica para guiar uma heurística de busca local tradicional na exploração do espaço de soluções além da otimalidade local, usando para isso basicamente estruturas de memória. Ela é uma das metaheurísticas mais usadas e seu sucesso decorre de sua eficiência em produzir soluções de alta qualidade para vários problemas combinatórios, entre os quais: programação de horários (Santos et al., 2005; Souza et al., 2004), roteirização (Archetti et al., 2006; Cordeau et al., 2002; Gendreau et al., 1999, 1996) e sequenciamento (Allahverdi et al., 2008).

Sendo uma técnica de busca local, a BT parte de uma solução inicial e se move no espaço de soluções de uma solução para outra que esteja em sua vizinhança. Além da necessidade de especificar como gerar uma solução inicial, definir os movimentos para explorar o espaço de soluções e estabelecer o critério de parada, que são componentes típicos de técnicas de busca local, para projetar um algoritmo BT é necessário, também, especificar os seguintes componentes básicos: (1) Critério de escolha da próxima solução vizinha; (2) Seleção dos atributos do movimento; (3) Memória de curto prazo para armazenar

as regras de proibição (lista tabu); (4) Número de iterações que um atributo selecionado permanecerá proibido (tamanho da lista) e (5) Critério de aspiração.

Um algoritmo BT com esses componentes básicos é o mais difundido, sendo suficiente para resolver satisfatoriamente muitos problemas combinatórios (Glover e Laguna, 1997). Entretanto, a Busca Tabu não se resume apenas a isso. Algoritmos BT mais sofisticados incluem, também, o uso de memórias de longo prazo para a aplicação de estratégias de intensificação e/ou diversificação, bem como de Reconexão por Caminhos e Oscilação Estratégica, o que proporciona à técnica mais recursos para explorar melhor o espaço de soluções.

Um grande número de publicações descrevendo a Busca Tabu é encontrado na literatura. Entre essas, destacamos: Gendreau (2003, 2002); Glover e Laguna (1997); Glover et al. (1993); Glover e Laguna (1993); Hertz e de Werra (1990); Glover (1990, 1989); de Werra e Hertz (1989).

O restante deste capítulo está organizado como segue. Na Seção 1 ilustra-se o funcionamento de um algoritmo básico BT considerando um problema simples, mas de muita aplicabilidade, o da Mochila 0-1. Nessa seção, os componentes típicos da BT são progressivamente incluídos e justificados. Na Seção 2 é apresentado o pseudocódigo de um algoritmo básico BT. Na Seção 3 são apresentados exemplos de regras de proibição para alguns problemas combinatórios. Na Seção 4 são mostradas estratégias para evitar a análise da vizinhança completa de uma solução. Na Seção 5 discute-se sobre a implementação eficiente da lista tabu, enquanto na Seção 6 relata-se sobre o tamanho ideal dessa lista. Na Seção 7 são listados os critérios de aspiração normalmente adotados nas implementações baseadas em BT. Na Seção 8 são detalhadas as memórias de longo prazo baseadas em frequência de transição e residência. Na Seção 9 apresenta-se a técnica Oscilação Estratégica, enquanto na última Seção são listadas as referências usadas neste capítulo. A estratégia Reconexão por Caminhos, apesar de normalmente fazer parte de uma implementação baseada em Busca Tabu, não será apresentada, uma vez que está detalhada no capítulo deste livro sobre a técnica GRASP.

1. Funcionamento de um algoritmo BT

Para ilustrar o funcionamento de um algoritmo BT, consideraremos uma instância do PM01 (Problema da Mochila 0-1). Nesse problema, há uma mochila de capacidade b, um conjunto de n itens j, cada qual com um peso w_j e um valor de retorno p_j caso o item j seja alocado à mochila. O objetivo é escolher os itens que devem ser alocados à mochila de forma que o valor de retorno total seja o maior possível, respeitando-se a capacidade da mochila.

Matematicamente, o PM01 pode ser representado pelo modelo de programação linear inteira definido pelas equações (9.1)-(9.3).

$$\max f(s) = \sum_{j=1}^{n} p_j s_j \tag{9.1}$$

$$\sum_{j=1}^{n} w_j s_j \leq b \tag{9.2}$$

$$s_j \in \{0,1\}\ \forall j = 1, \cdots, n \tag{9.3}$$

em que s_j é uma variável binária que assume valor 1 se o item j for alocado à mochila e 0, caso contrário.

A Eq. (9.1) representa a função objetivo, a qual visa à maximização do valor de retorno dos itens alocados à mochila. A restrição (9.2) impede que a capacidade da mochila seja superada, enquanto as restrições (9.3) definem o tipo das variáveis de decisão.

Como exemplo do PM01, seja uma mochila de capacidade $b = 32$ e os dados da Tabela 9.1.

Tabela 9.1: Instância do Problema da Mochila 0-1

Item j	1	2	3	4	5	6	7	8
Peso w_j	4	15	7	9	8	10	9	11
Retorno p_j	2	2	3	4	6	5	8	7

Como é uma técnica de busca local, a BT explora o espaço de soluções S indo de uma solução $s \in S$ a outra s' que esteja em sua vizinhança $V \subseteq N(s)$. A solução $s' \in N(s)$ é dita vizinha de s se ela for obtida pela aplicação de um movimento m a s, operação esta representada por $s' \leftarrow s \oplus m$.

Em muitos problemas, a solução vizinha é alcançada por meio de vários tipos de movimentos. Nesse caso, $N(s) = \bigcup_{i \in M} N^i(s)$, sendo $M = \bigcup m_i$ o conjunto dos diferentes movimentos m_i.

No Problema da Mochila 0-1, uma solução s pode ser representada por um vetor binário n-dimensional $s = (s_1, s_2, \cdots, s_n)$, em que cada componente $s_j \in \{0, 1\}$ assume valor 1 se o item j for alocado à mochila e 0, caso contrário. Para explorar o espaço de soluções deste problema, um movimento m natural é considerar a inversão do valor de um bit. Desta maneira, a vizinhança $N(s)$ de uma solução s do PM01 consiste no conjunto dos vizinhos s' que diferem de s pelo valor de um único bit. Consideraremos, nesta definição de vizinhança, que *todas* as soluções vizinhas serão analisadas, isto é, que $V = N(s)$. Na Seção 4 é apresentada uma estratégia para evitar a análise da vizinhança completa.

É necessário, agora, definir uma função de avaliação para guiar o procedimento de busca no espaço de soluções do problema. Considerando que se deseja maximizar o valor de retorno trazido pela utilização dos itens alocados à mochila, há duas possibilidades para a escolha dessa função.

A primeira é considerar a exploração apenas no conjunto de soluções factíveis. Neste caso, a função de avaliação coincide com a própria função objetivo do problema, dada pela Eq. (9.1), e a solução s satisfaz à condição de que a capacidade da mochila seja respeitada, isto é, que a restrição (9.2) não seja violada.

Outra possibilidade é permitir a geração de soluções infactíveis. Neste caso, ao invés de utilizar a função f da Eq. (9.1), avaliaríamos uma solução s pela função auxiliar g, baseada em penalidade, dada pela Eq. (9.4):

$$g(s) = \sum_{j=1}^{n} p_j s_j - \rho \times \max\{0, \sum_{j=1}^{n} w_j s_j - b\} \tag{9.4}$$

sendo ρ uma penalidade.

A primeira parcela dessa função de avaliação g é a função objetivo f propriamente dita do PM01, enquanto a segunda é uma penalização que tem como objetivo desestimular a colocação na mochila de itens que ultrapassam sua capacidade. Como a função g deve ser maximizada, o sinal desta segunda parcela é negativo de forma a não incentivar a realização de movimentos que gerem soluções infactíveis. O valor de ρ deve ser suficientemente grande para atender a esse objetivo. Para os dados da Tabela 9.1, pode-se tomar, por exemplo, $\rho = \sum_{j=1}^{n} p_j = 15$.

Para o que se segue, será permitida apenas a geração de soluções factíveis. No entanto, pode ser conveniente alternar a exploração do espaço de soluções entre soluções factíveis e infactíveis e, nesse caso, a função g dada pela Eq. (9.4) deve ser usada. Considerações acerca dessa estratégia são feitas na Seção 9.

Consideremos um exemplo do PM01 cuja solução inicial seja $s^0 = (1, 0, 0, 1,\ 0, 1, 1, 0)$, obtida de forma aleatória. A essa solução estão associados os valores $f(s^0) = 19$ e $peso(s^0) = 32$, correspondentes à aplicação da função objetivo f dada pela Eq. (9.1) e da função $peso(s) = \sum_{j=1}^{n} w_j s_j$, respectivamente.

Na Tabela 9.2 estão representados todos os vizinhos segundo a definição de vizinhança adotada. Na primeira coluna desta tabela representa-se o número do vizinho; na segunda, o vizinho; na terceira, seu peso e na quarta, o valor de sua função de avaliação. O sinal "–" indica que a respectiva solução é infactível. Números em negrito indicam o bit modificado.

Tabela 9.2: Vizinhança da solução s^0

viz.	$s' \in N(s^0)$	$peso(s')$	$f(s')$
1	$(\mathbf{0}, 0, 0, 1, 0, 1, 1, 0)$	28	17
2	$(1, \mathbf{1}, 0, 1, 0, 1, 1, 0)$	47	–
3	$(1, 0, \mathbf{1}, 1, 0, 1, 1, 0)$	39	–
4	$(1, 0, 0, \mathbf{0}, 0, 1, 1, 0)$	23	15
5	$(1, 0, 0, 1, \mathbf{1}, 1, 1, 0)$	40	–
6	$(1, 0, 0, 1, 0, \mathbf{0}, 1, 0)$	22	14
7	$(1, 0, 0, 1, 0, 1, \mathbf{0}, 0)$	23	11
8	$(1, 0, 0, 1, 0, 1, 1, \mathbf{1})$	43	–

A solução s^0 representa um ótimo local em relação à vizinhança considerada, uma vez que todos os seus vizinhos têm avaliação pior. Uma heurística de busca local convencional pararia nesse ponto. No entanto, a BT aceita soluções de não-melhora como estratégia para ir além desse ótimo. No exemplo considerado, escolheremos o melhor membro $s' \in N(s^0)$ como *critério de escolha do próximo vizinho*, no caso, o primeiro. Desta forma, ao final da primeira iteração, a nova solução corrente passa a ser $s^1 = (0, 0, 0, 1, 0, 1, 1, 0)$, com $f(s^1) = 17$, sendo o valor da melhor solução dada por $f(s^\star) = 19$.

Vejamos o que aconteceria caso prosseguíssemos com a mesma estratégia para a vizinhança da solução corrente. Seus vizinhos estão explicitados na Tabela 9.3.

Tabela 9.3: Vizinhança da solução s^1

viz.	$s' \in N(s^1)$	$peso(s')$	$f(s')$
1	$(\mathbf{1}, 0, 0, 1, 0, 1, 1, 0)$	32	19
2	$(0, \mathbf{1}, 0, 1, 0, 1, 1, 0)$	43	–
3	$(0, 0, \mathbf{1}, 1, 0, 1, 1, 0)$	35	–
4	$(0, 0, 0, \mathbf{0}, 0, 1, 1, 0)$	19	13
5	$(0, 0, 0, 1, \mathbf{1}, 1, 1, 0)$	36	–
6	$(0, 0, 0, 1, 0, \mathbf{0}, 1, 0)$	18	12
7	$(0, 0, 0, 1, 0, 1, \mathbf{0}, 0)$	19	9
8	$(0, 0, 0, 1, 0, 1, 1, \mathbf{1})$	39	–

Pela Tabela 9.3, a solução $s' = (1, 0, 0, 1, 0, 1, 1, 0)$, com $f(s') = 19$, é o melhor vizinho de s^1. Se movêssemos para essa solução, obtendo $s^2 \leftarrow s'$, voltaríamos à solução inicial s^0 e, posteriormente, prosseguiríamos novamente em direção à s^1, caso repetíssimos a mesma estratégia anterior. Isto é, o algoritmo ciclaria usando-se unicamente a estratégia de mover-se para o melhor vizinho.

Para evitar a ciclagem, ou seja, a geração de uma mesma sequência de soluções já visitadas anteriormente, um algoritmo BT usa uma estrutura de memória, a chamada lista tabu, assim chamada a lista de soluções "proibidas". Sempre que uma solução é gerada, ela é colocada na lista tabu T. Assim, à medida que uma nova solução é gerada, a lista é aumentada. No exemplo dado, ao final da primeira iteração teríamos $T^1 = \{s^0\}$, indicando que s^0 não seria candidato ao melhor vizinho de s^1, por já ter sido gerado anteriormente. A nova solução agora deve ser $s^2 = (0, 0, 0, 0, 0, 1, 1, 0)$, uma vez que este é o vizinho não tabu de melhor avaliação. Com esta estratégia de memória, o algoritmo BT pode ir além do ótimo local e acessar outras regiões do espaço de soluções.

Uma lista tabu clássica armazena $|T|$ soluções e funciona numa estrutura de fila de tamanho fixo, isto é, quando a lista está cheia e uma nova solução entra, a mais antiga sai. Essa estratégia está

fundamentada no fato de que na exploração do espaço de soluções, as soluções geradas há mais tempo possivelmente estão "distantes" da região do espaço sob análise e, como tal, não têm influência na escolha da próxima solução vizinha naquela região. Desta forma, armazendo-se apenas as soluções mais "próximas" da solução atual, a ciclagem estará evitada nessa região. Por armazenar apenas as soluções mais recentes, essa lista tabu é referenciada como uma memória de curto prazo (*short-term memory*). Uma lista de tamanho $|T|$, no entanto, somente impede ciclos de até $|T|$ soluções. Isso significa que, se na k-ésima iteração a lista tabu é $T = \{s^{k-r}, \cdots, s^{k-2}, s^{k-1}\}$, então, quando $r = k$, a ciclagem estará completamente evitada. Entretanto, na iteração seguinte, s^{k-r} sai da lista, podendo retornar caso seja novamente selecionada pelo critério de escolha do próximo vizinho. A definição do tamanho da lista é, pois, um parâmetro crítico da BT. A dimensão da lista não pode ser tão pequena, sob pena de haver ciclagem; nem tão grande, para armazenar desnecessariamente soluções que não estejam ligadas à história recente da busca. Uma ideia do tamanho da lista tabu será discutida mais adiante, na Seção 6.

Em muitas aplicações reais, no entanto, além de requerer muito espaço em memória, é muito dispendioso computacionalmente avaliar se uma solução está ou não presente na lista tabu. Por exemplo, em um problema de programação de tripulações (*crew scheduling*), uma solução é uma escala de trabalho, isto é, um conjunto de jornadas, cada qual representada por um conjunto de viagens a serem realizadas pelos tripulantes. Verificar se uma dada escala está ou não em uma lista tabu de soluções envolveria comparar as jornadas que a compõem com aquelas das escalas armazenadas em T. Essa operação poderia ser simplificada, comparando-se primeiramente o valor da função de avaliação da escala dada com o das escalas da lista. Sendo diferente, então a referida escala não seria tabu; caso contrário, o segundo passo da comparação consistiria em analisar o custo das jornadas. Igualmente, havendo algum custo diferente, concluiríamos que a escala corrente não é tabu. No entanto, no caso de todos os custos serem iguais, então as jornadas seriam comparadas viagem por viagem. Mesmo essa alternativa de comparação é ainda muito custosa computacionalmente.

Por outro lado, quando se move de uma solução para outra que esteja em sua vizinhança, na realidade a solução vizinha difere muito pouco da solução corrente, mais precisamente, apenas do movimento realizado. Em vista disso e das considerações anteriores, ao invés de armazenar toda uma solução na lista, guarda-se nela apenas alguma regra de proibição associada a um atributo (característica) da solução ou movimento realizado para impedir o retorno a uma solução já gerada anteriormente. O atributo selecionado é dito tabu-ativo e uma solução que contém atributos tabu-ativos torna-se tabu. Um movimento é dito tabu se ele der origem a uma solução tabu. Referenciar-nos-emos a tal lista como lista tabu baseada em atributos. Na Seção 3 são mostradas algumas regras de proibição para problemas combinatórios que usam movimentos de inserção e troca.

Voltando ao PM01, quando se muda para um novo vizinho, este difere da solução corrente apenas no valor do bit modificado. Por exemplo, da solução s^0 para a solução s^1, o único bit alterado é o primeiro. Então, nada mais natural do que considerar como atributo a posição do bit na solução e como regra de proibição, a inversão do valor do bit dessa posição. Desta forma, a posição 1 torna-se tabu ativa e o movimento que altera o valor do bit desta posição é dito tabu pois gera uma solução tabu (a solução s^0).

Considerando uma lista tabu de tamanho 2, isto é, $|T| = 2$, na primeira iteração (Tabela 9.2) proibiríamos a inversão do primeiro bit, representando esta operação por $T = \{\langle 1\rangle\}$. Com isso, todos os vizinhos obtidos de s^1 pela alteração do primeiro bit estarão proibidos. Desta forma, o retorno à s^0 fica impedido. Na segunda iteração (Tabela 9.3), o melhor vizinho não tabu de s^1 é obtido pela alteração do quarto bit, então $T = \{\langle 1\rangle, \langle 4\rangle\}$ ao final desta iteração. Agora, está proibida a geração de vizinhos de s^2 que diferem desta solução pela alteração no valor do primeiro ou quarto bit.

Passemos, agora, à terceira iteração, procurando a próxima solução vizinha de s^2. A Tabela 9.4 relaciona os dados da vizinhança completa desta solução.

Na Tabela 9.4, os vizinhos 1 e 4 (assinalados com um t na primeira coluna) estão proibidos de serem

Tabela 9.4: Vizinhança da solução s^2

viz.	$s' \in N(s^2)$	$peso(s')$	$f(s')$
1^t	$(\mathbf{1},0,0,0,0,1,1,0)$	23	15
2	$(0,\mathbf{1},0,0,0,1,1,0)$	34	–
3	$(0,0,\mathbf{1},0,0,1,1,0)$	26	16
4^t	$(0,0,0,\mathbf{1},0,1,1,0)$	28	17
5	$(0,0,0,0,\mathbf{1},1,1,0)$	27	19
6	$(0,0,0,0,0,\mathbf{0},1,0)$	9	8
7	$(0,0,0,0,0,1,\mathbf{0},0)$	10	5
$8^\star$	$(0,0,0,0,0,1,1,\mathbf{1})$	30	20

acessados, uma vez que o vizinho 1 é alcançado alterando-se o primeiro bit e o vizinho 4, o quarto bit, e ambos pertencem à lista tabu T. Como se observa, o melhor vizinho não tabu (assinalado com um asterisco) é o último, dado por $s' = (0,0,0,0,0,1,1,1)$, o qual passa a ser a nova solução corrente s^3. Essa solução é melhor que a melhor solução gerada até então, pois $f(s^3) = 20 > f(s^\star) = 19$. Assim, a melhor solução deve ser atualizada. A estratégia de mover para o melhor vizinho, ainda que esse seja de pior avaliação, associada ao uso de memória, proporcionou à busca a possibilidade de prosseguir além do ótimo local s^0 e encontrar melhores soluções, sem a ocorrência de ciclagem. Como definimos $|T| = 2$ e a lista tabu já tem dois atributos tabu-ativos, então o atributo tabu mais antigo, $\langle 1\rangle$, sai para a entrada de $\langle 8\rangle$, resultando em $T = \{\langle 4\rangle, \langle 8\rangle\}$ ao final desta iteração.

A lista tabu baseada em atributos se, por um lado, visa à eliminação de ciclagem (uma vez que ela garante o não retorno, por $|T|$ iterações, a uma solução já visitada anteriormente); por outro, pode ser restritiva (de Werra e Hertz, 1989). De fato, aplicando-se o movimento tabu $\langle 4\rangle$ à solução s^3 do PM01, obtém-se a solução $s' = (0,0,0,1,0,1,1,1)$ que ainda não foi gerada e, dessa forma, não deveria estar proibida. Em outras palavras, uma lista tabu baseada em atributos pode não somente impedir o retorno a uma solução já gerada anteriormente, como também impedir que algumas soluções do espaço de busca sejam alcançadas.

De forma a contornar essa situação, aplica-se o chamado *critério de aspiração*. Um critério de aspiração é um mecanismo que retira, sob certas circunstâncias, a classificação tabu de um movimento. Um exemplo simples de aplicação dessa ideia é executar um movimento tabu somente se ele conduzir a um vizinho melhor que $s^\star$. Esse é o chamado critério de aspiração por objetivo global. Ele se fundamenta no fato de que se um movimento tabu conduz a uma solução com valor melhor que o da melhor solução encontrada até então, é sinal de que ela ainda não foi gerada. Outros critérios de aspiração são discutidos na Seção 7.

Nas Tabelas 9.5 a 9.10 são registradas as características das soluções geradas pela Busca Tabu aplicada ao PM01 durante as próximas 6 iterações. Em cada tabela são mostrados os dados dos vizinhos, o valor da solução corrente, o valor da melhor solução encontrada até então, bem como a lista tabu ao final da iteração. Considerou-se como critério de parada 3 iterações sem melhora.

A melhor solução obtida ao final das nove primeiras iterações da Busca Tabu foi $s^\star = s^6 = (1,0,0,0,1,0,1,1)$, com valor $f(s^\star) = 23$, sendo ela alcançada na sexta iteração.

Ilustra-se, pela Figura 9.1, a evolução do valor da função objetivo do PM01 ao longo das iterações realizadas. A Figura 9.2, por sua vez, mostra uma situação de ciclagem em Busca Tabu. Observa-se que, a partir da quinta iteração, aparece uma sequência de valores para a função objetivo que se repete de 6 em 6 iterações.

Tabela 9.5: Vizinhança de s^3

viz.	$s' \in N(s^3)$	$peso(s')$	$f(s')$
1	$(\mathbf{1},0,0,0,0,1,1,1)$	34	–
2	$(0,\mathbf{1},0,0,0,1,1,1)$	45	–
3	$(0,0,\mathbf{1},0,0,1,1,1)$	37	–
4^t	$(0,0,0,\mathbf{1},0,1,1,1)$	39	–
5	$(0,0,0,0,\mathbf{1},1,1,1)$	38	–
$6^\star$	$(0,0,0,0,0,\mathbf{0},1,1)$	20	15
7	$(0,0,0,0,0,1,\mathbf{0},1)$	21	12
8^t	$(0,0,0,0,0,1,1,\mathbf{0})$	19	13

$T = \{\langle 8\rangle, \langle 6\rangle\}$; $f(s^4) = 15$; $f(s^\star) = 20$

Tabela 9.6: Vizinhança de s^4

viz.	$s' \in N(s^4)$	$peso(s')$	$f(s')$
1	$(\mathbf{1},0,0,0,0,0,1,1)$	24	17
2	$(0,\mathbf{1},0,0,0,0,1,1)$	35	–
3	$(0,0,\mathbf{1},0,0,0,1,1)$	27	18
4	$(0,0,0,\mathbf{1},0,0,1,1)$	29	19
$5^\star$	$(0,0,0,0,\mathbf{1},0,1,1)$	28	21
6^t	$(0,0,0,0,0,\mathbf{1},1,1)$	30	20
7	$(0,0,0,0,0,0,\mathbf{0},1)$	11	7
8^t	$(0,0,0,0,0,0,1,\mathbf{0})$	9	8

$T = \{\langle 6\rangle, \langle 5\rangle\}$; $f(s^5) = 21$; $f(s^\star) = 21$

Tabela 9.7: Vizinhança de s^5

viz.	$s' \in N(s^5)$	$peso(s')$	$f(s')$
$1^\star$	$(\mathbf{1},0,0,0,1,0,1,1)$	32	23
2	$(0,\mathbf{1},0,0,1,0,1,1)$	43	–
3	$(0,0,\mathbf{1},0,1,0,1,1)$	35	–
4	$(0,0,0,\mathbf{1},1,0,1,1)$	37	–
5^t	$(0,0,0,0,\mathbf{0},0,1,1)$	20	15
6^t	$(0,0,0,0,1,\mathbf{1},1,1)$	38	–
7	$(0,0,0,0,1,0,\mathbf{0},1)$	19	13
8	$(0,0,0,0,1,0,1,\mathbf{0})$	17	14

$T = \{\langle 5\rangle, \langle 1\rangle\}$; $f(s^6) = 23$; $f(s^\star) = 23$

Tabela 9.8: Vizinhança de s^6

viz.	$s' \in N(s^6)$	$peso(s')$	$f(s')$
1^t	$(\mathbf{0},0,0,0,1,0,1,1)$	28	21
2	$(1,\mathbf{1},0,0,1,0,1,1)$	47	–
3	$(1,0,\mathbf{1},0,1,0,1,1)$	39	–
4	$(1,0,0,\mathbf{1},1,0,1,1)$	41	–
5^t	$(1,0,0,0,\mathbf{0},0,1,1)$	24	17
6	$(1,0,0,0,1,\mathbf{1},1,1)$	42	–
7	$(1,0,0,0,1,0,\mathbf{0},1)$	23	15
$8^\star$	$(1,0,0,0,1,0,1,\mathbf{0})$	21	16

$T = \{\langle 1\rangle, \langle 8\rangle\}$; $f(s') = 16$; $f(s^\star) = 23$

2. O Algoritmo Busca Tabu

O Algoritmo 1 resume as ideias básicas apresentadas da Busca Tabu para um problema de minimização. Os parâmetros principais de controle do algoritmo são a cardinalidade $|T|$ da lista tabu, o critério de aspiração, a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração e a regra de parada.

As regras de parada comumente utilizadas são: (a) o número de iterações sem melhora na solução global (*iter - MelhorIter*) atinge um valor máximo $BTmax$ e (b) o tempo de processamento atinge um tempo máximo permitido.

3. Exemplos de regras de proibição

Ilustraremos, nesta seção, algumas regras de proibição comumente aplicadas a problemas que envolvem permutação de elementos de uma solução.

Seja uma permutação (solução) s de n elementos. Em um movimento de troca, um elemento i da posição s_i é permutado com o elemento j da posição s_j, enquanto em um movimento de inserção, um elemento i da posição s_i é inserido na posição s_j.

Exemplo: Dada a solução $s = (2,6,1,5,4,3)$, a solução $s' = (2,6,1,\mathbf{3},4,\mathbf{5})$ é obtida de s pela permutação do elemento 5, da quarta posição, com o elemento 3, da sexta posição. Já a solução $s' = (2,6,1,4,3,\mathbf{5})$ é obtida de s inserindo-se o elemento 5, da quarta posição, na sexta posição.

Um atributo pode ser definido pelo par ⟨elemento, posição do elemento⟩. Considerando a troca do elemento i da posição s_i com o elemento j da posição s_j ou que o elemento i é inserido na posição s_j, podemos ter as seguintes regras de proibição para movimentos de troca e inserção (França, 2009):

1. Impedir movimentos que resultem em uma permutação em que i ocupe a posição s_i e j ocupe a posição s_j. Para o exemplo considerado, supondo a troca de $i = 5$ com $j = 3$, então está

Tabela 9.9: Vizinhança de s^7

viz.	$s' \in N(s^7)$	$peso(s')$	$f(s')$
1^t	$(\mathbf{0},0,0,0,1,0,1,0)$	17	14
2	$(1,\mathbf{1},0,0,1,0,1,0)$	36	–
3	$(1,0,\mathbf{1},0,1,0,1,0)$	28	19
4	$(1,0,0,\mathbf{1},1,0,1,0)$	30	20
5	$(1,0,0,0,\mathbf{0},0,1,0)$	13	10
$6^\star$	$(1,0,0,0,1,\mathbf{1},1,0)$	31	21
7	$(1,0,0,0,1,0,\mathbf{0},0)$	12	8
8^t	$(1,0,0,0,1,0,1,\mathbf{1})$	32	23

$T = \{\langle 8\rangle, \langle 6\rangle\}$; $f(s^8) = 21$; $f(s^\star) = 23$

Tabela 9.10: Vizinhança de s^8

viz.	$s' \in N(s^8)$	$peso(s')$	$f(s')$
$1^\star$	$(\mathbf{0},0,0,0,1,1,1,0)$	27	19
2	$(1,\mathbf{1},0,0,1,1,1,0)$	46	–
3	$(1,0,\mathbf{1},0,1,1,1,0)$	38	–
4	$(1,0,0,\mathbf{1},1,1,1,0)$	40	–
5	$(1,0,0,0,\mathbf{0},1,1,0)$	23	15
6^t	$(1,0,0,0,1,\mathbf{0},1,0)$	21	16
7	$(1,0,0,0,1,1,\mathbf{0},0)$	22	13
8^t	$(1,0,0,0,1,1,1,\mathbf{1})$	42	–

$T = \{\langle 6\rangle, \langle 1\rangle\}$; $f(s^9) = 19$; $f(s^\star) = 23$

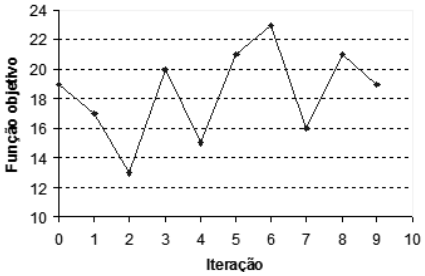


Figura 9.1: Evolução da função objetivo

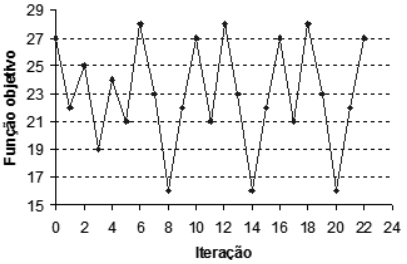


Figura 9.2: Exemplo de ciclagem

Algoritmo 1 Busca Tabu

Entrada: $f(.), N(.), |T|, |V|, s$
Saída: $s^\star$
$s^\star \leftarrow s$ {Melhor solução encontrada até então}
$Iter \leftarrow 0$ {Contador do número de iterações}
$MelhorIter \leftarrow 0$ {Iteração mais recente que forneceu $s^\star$}
$T \leftarrow \emptyset$ {lista tabu}
enquanto Critério de parada não satisfeito **faça**
  $Iter \leftarrow Iter + 1$
  Seja $s' \leftarrow s \oplus m$ o melhor elemento de $V \subseteq N(s)$ tal que o movimento m não seja tabu ($m \notin T$) ou s' atenda a um critério de aspiração
  Atualize a lista tabu T
  $s \leftarrow s'$
  se $f(s) < f(s^\star)$ **então**
    $s^\star \leftarrow s$
    $MelhorIter \leftarrow Iter$
  fim se
fim enquanto
Retorne $s^\star$

proibido que o elemento 5, da quarta posição, ocupe a sexta posição, assim como o elemento 3, da sexta posição, ocupe a quarta posição.

2. Impedir movimentos que resultem em uma permutação em que i ocupe a posição s_i ou j ocupe a posição s_j. Para o exemplo considerado, supondo a troca de $i = 5$ com $j = 3$, então está

proibido que o elemento 5, da quarta posição, ocupe a sexta posição ou, então, que o elemento 3, da sexta posição ocupe a quarta posição.

3. Impedir que o elemento i retorne à posição s_i. No exemplo dado, está proibido que o elemento 5 retorne à quarta posição.

4. Impedir que o elemento i mova para posições $k \leq s_i$. No exemplo considerado, ficaria proibido que o elemento 5 movesse para qualquer posição menor ou igual à quarta.

5. Impedir que o elemento i se mova para posições $k \leq s_j$. No exemplo referenciado, estaria impedida a mudança do elemento 5 para posições inferiores ou igual à sexta. Em outras palavras, neste caso, esta regra impediria qualquer movimentação do elemento 5, uma vez que a sexta posição é a última.

6. Impedir que o elemento i se mova. Para o exemplo dado, o elemento 5 estaria proibido de se mover.

7. Impedir que os elementos i e j troquem de posição com quaisquer outros elementos. No exemplo considerado, ficaria proibida a troca de posição do elemento 5 com qualquer outro elemento, bem como a troca do elemento 3 com qualquer outro.

8. Impedir que os elementos i e j se movam. No exemplo dado, isto significa que o elemento 5 está proibido de se mover para qualquer outra posição, assim como o elemento 3.

As regras anteriores estão listadas em ordem crescente de restritividade, isto é, as primeiras regras impedem a geração de um conjunto menor de soluções, enquanto as últimas proíbem um número bem maior de soluções.

O próximo exemplo considera um problema de alocação de aulas a salas (*Classroom Assignment Problem*). Nesse problema, há um conjunto $H = \{1, 2, 3, 4\}$ de horários para a realização das aulas, um conjunto $S = \{1, 2, 3\}$ de salas nas quais as aulas são ministradas e um conjunto $M = \{A, B, C\}$ de aulas a serem ministradas, cujos horários são previamente fixados. A Tabela 9.11 mostra uma solução s, enquanto a Tabela 9.12 apresenta uma solução s' obtida de s por meio de um movimento de troca de sala envolvendo duas aulas distintas.

Tabela 9.11: solução s

	Salas		
Horário	1	2	3
1	A		
2			D
3		C	D
4	B	C	

Tabela 9.12: solução s'

	Salas		
Horário	1	2	3
1	A		
2		D	
3		D	C
4	B		C

Nas tabelas 9.11 e 9.12, cada célula s_{ij} representa a aula ministrada no horário i e sala j. Por exemplo, na solução s, a aula C é ministrada na sala 2 durante os horários 3 e 4. Como atributo do movimento de troca, podemos considerar o par $\langle i, j \rangle$, indicando uma aula (definida por seu horário i de início) e sua sala j de realização. As seguintes regras de proibição são possíveis, dentre outras: (a) Impedir a troca das aulas envolvendo as salas j_1 e j_2, com início no horário i, sendo $i = \min\{i_1, i_2\}$, i_1 o horário de início da aula da sala j_1 e i_2 o da sala j_2; (b) Impedir que a aula da sala j_1, iniciada no horário i_1, seja mudada para a sala j_2; (c) Impedir que a aula da sala j_1, com início no horário i_1, seja mudada; (d) Aplicar uma das regras anteriores, associadas ao valor da função objetivo antes da execução do movimento.

Para uma classe de problemas de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção, Wan e Yen (2002) desenvolveram um algoritmo baseado em Busca Tabu que faz uso de movimentos de troca envolvendo apenas tarefas adjacentes na ordem de produção. Os autores estabeleceram como atributo tabu a tripla $\langle i, j, fo \rangle$, em que i e j são as tarefas envolvidas na troca e *fo* é o valor da função objetivo da solução antes do movimento realizado.

Exemplos de regras de proibição para os problemas de programação de horários, generalizado de atribuição e n-rainhas são apresentados na Seção 8.

4. Lista de Candidatos

Para muitos problemas combinatórios, a exploração da vizinhança completa é muito custosa computacionalmente. Para esses casos, é essencial utilizar uma Lista de Candidatos.

Em uma Lista de Candidatos, a análise da vizinhança N da solução corrente é restringida a um subconjunto V dessa (linha 9 do Algoritmo 1). Tipicamente, são analisadas apenas as soluções "promissoras", assim referenciadas as soluções que satisfazem a um determinado critério.

Para exemplificar tal estratégia, consideremos o problema de sequenciamento em uma máquina tendo como objetivo a minimização do atraso total. Neste problema, são dados um conjunto de n tarefas, o tempo de processamento p_j da tarefa j e a data de entrega d_j da tarefa j. Como variáveis de decisão tem-se C_j, representando o instante de término da tarefa j, e o atraso T_j dessa tarefa, definido por $T_j = \max\{0, C_j - d_j\}$. O problema consiste, pois, em determinar uma sequência s que minimize a Eq. (9.5):

$$T = \sum_{j=1}^{n} \max\{0, C_j - d_j\} \tag{9.5}$$

Utilizando-se apenas movimentos de troca, tem-se uma vizinhança de tamanho $n(n-1)/2$, ou seja, a análise da vizinhança inteira tem complexidade $O(n^2)$. Seja $\Delta = f(s') - f(s)$ o valor de um movimento e uma instância deste problema, extraída de Glover e Laguna (1997), com os seguintes dados: $p = (6, 4, 8, 2, 10, 3)$ e $d = (9, 12, 15, 8, 20, 22)$. Dada a sequência $s = (1, 2, 3, 4, 5, 6)$, tem-se que o atraso total é $T = 36$.

A Tabela 9.13 mostra a análise da vizinhança completa, a qual envolve 15 soluções vizinhas. Nesta tabela, mostra-se, também, o valor Δ do movimento, o atraso total T da solução vizinha, bem como o valor da diferença entre as datas de entrega das tarefas i e j envolvidas na troca.

Tabela 9.13: Vizinhança de um problema de sequenciamento de tarefas

Tarefas		Δ	T	$\lvert d_i - d_j \rvert$
1	2	1	37	**3**
1	3	6	42	6
1	4	-4	32	**1**
1	5	21	57	11
1	6	4	40	13
2	3	3	39	**3**
2	4	-6	30	4
2	5	20	56	8
2	6	7	43	10
3	4	-6	30	7
3	5	4	40	5
3	6	-6	30	7
4	5	8	44	12
4	6	3	39	14
5	6	-7	29	**2**

Caso fosse criada uma lista de candidatos pela regra $|d_i - d_j| \leq 3$, teríamos analisado apenas 4 soluções vizinhas (os movimentos que dão origem a essas soluções estão destacados em negrito na Tabela 9.13). Dentre essas, a melhor solução (obtida a partir da troca entre as tarefas 5 e 6), estaria incluída. Em outras palavras, com uma regra simples, a de que uma solução vizinha só é analisada se a diferença entre as datas de entrega das tarefas envolvidas na troca for menor ou igual a 3 unidades, a vizinhança é substancialmente reduzida. O valor 3 estabelecido para a distância máxima entre as datas de entrega é, obviamente, um parâmetro a ser considerado.

Ao invés de restringir a análise da vizinhança às soluções "promissoras", poder-se-ia, simplesmente, reduzir a vizinhança. Por exemplo, em problemas de alocação de aulas a salas (*classroom assignment problem*), que envolve um conjunto de salas, um conjunto de turmas e um conjunto de dias da semana para a realização das aulas, uma Lista de Candidatos pode ser um dia da semana. Nesta situação, pode-se aplicar uma estratégia do tipo: na primeira iteração da BT utilizam-se como vizinhança soluções obtidas a partir de movimentos envolvendo apenas as aulas da segunda-feira; na segunda iteração, aulas da terça-feira e assim por diante.

Em problemas de programação de horários em escolas (*school timetabling problem*), poder-se-ia, de forma análoga, considerar na primeira iteração da BT uma vizinhança envolvendo apenas as alocações dos primeiros k professores; na segunda, dos próximos k professores e assim por diante, até atingir os k últimos professores, situação na qual se voltaria para a análise da vizinhança envolvendo os primeiros k professores.

5. Implementação eficiente da lista tabu

Na prática, normalmente não se implementa uma lista tabu na estrutura de uma lista tal como apresentado anteriormente, pois, quase sempre, é possível utilizar uma estrutura de dados mais eficiente. Para mostrar isso, relembremos que no PM01 consideramos como movimento tabu a inversão do valor do bit relativo a um dado item. Isso significa que a cada iteração da Busca Tabu demandamos um máximo de $O(|T|)$ operações para verificar se um movimento é tabu ou não, sendo $|T|$ a cardinalidade da lista. A complexidade dessa verificação pode ser reduzida a $O(1)$ se utilizarmos uma estrutura especial, que armazene apenas o tempo em que o movimento tabu permanecerá na lista. Esse tempo de permanência é conhecido como duração tabu (*tabu tenure*).

Assim, no problema da mochila mencionado, ao invés de armazenarmos uma lista com os movimentos tabus, poderíamos utilizar um vetor T de n posições, com cada célula $T(j)$ representando a iteração até a qual a inversão do bit relativo ao item j estará proibida. Inicialmente, toma-se $T(j) = 0$ $\forall j = 1, \cdots, n$. Considerando *iter* a iteração atual e supondo a proibição durante *TempoTabu* iterações, então $T(j) \leftarrow iter + TempoTabu$.

Exemplificando, ao final da primeira iteração do PM01 considerado na Seção 1, tem-se $j = 1$, *iter*=1 e *TempoTabu* = 2, então $T = (3, 0, 0, 0, 0, 0, 0, 0)$, indicando que qualquer modificação relativa ao item 1 (primeira posição do vetor T) estará proibida até a 3^a iteração da Busca Tabu. Dito de outra maneira, a inversão do bit da primeira posição é um movimento tabu enquanto a iteração corrente $iter \leq T(1) = 3$. Na segunda iteração (*iter*=2), a inversão do bit da quarta posição foi considerada tabu. Assim, aplicando-se também a mesma duração tabu, tem-se a "lista" $T = (3, 0, 0, 4, 0, 0, 0, 0)$, indicando que a alteração dos valores dos bits relativos aos itens 1 e 4 está proibida até a terceira e quarta iterações, respectivamente. Observe que não é necessário atualizar os demais componentes da lista.

Osman (1993) trata um problema de roteamento de veículos com um algoritmo BT que usa uma estrutura de dados na forma de matriz para armazenar os movimentos tabus. Essa matriz, de dimensões $(n + 1) \times v$, consiste de v colunas (uma para cada conjunto R_p de clientes da rota p) e $n + 1$ linhas, sendo uma para cada um dos n consumidores e outra para um consumidor fictício envolvido nas operações de inserção. Um movimento consiste de dois pares (i, R_p) e (j, R_q), os quais identificam que o consumidor i do conjunto R_p de clientes da rota p foi trocado com o cliente j do conjunto R_q de consumidores da rota q. Os atributos $\langle i, R_p\rangle$ e $\langle j, R_q\rangle$ especificam as restrições tabu que proíbem um movimento de ser realizado. Um movimento é considerado tabu se o consumidor i retorna a R_p e j retorna a R_q simultaneamente. Cada elemento $T(i, p)$ da matriz registra o número da iteração na qual o consumidor i foi removido do conjunto R_p. Inicialmente, a matriz T é inicializada com valores negativos altos para evitar a identificação equivocada de consumidores como tabu durante as iterações iniciais. Um movimento é considerado tabu em uma iteração k se nem i retorna a R_p, nem j retorna a R_q durante as próximas *TempoTabu* iterações, isto é, se forem satisfeitas simultaneamente as duas condições a seguir: $k - T(i, p) < TempoTabu$ e $k - T(j, q) < TempoTabu$.

Duas estratégias para escolha da próxima solução vizinha são adotadas em Osman (1993), cada qual dando origem a uma variante diferente do algoritmo Busca Tabu. A primeira, denominada *Best-admissible selection strategy* – BA, consiste em escolher o vizinho de melhor avaliação dentre todos da vizinhança da solução corrente. A segunda, chamada de *First-best-admissible strategy* – FBA, consiste em escolher o primeiro vizinho que melhorar a solução corrente; no caso de não haver soluções de melhora, então o melhor vizinho de não melhora é selecionado. Para aplicar a estratégia BA são usadas duas matrizes, BSTM e RECM, de dimensões $v \times v$ e $[v(v-1)/2] \times 2$, respectivamente. A parte triangular superior da matriz BSTM(p, q), com $1 \leq p < q \leq v$, é usada para armazenar a variação Δ_{ij} no valor da função objetivo associada ao melhor movimento obtido, de troca do consumidor $i \in R_p$ com $j \in R_q$, ou um valor arbitrariamente alto se tal movimento não é permitido. A parte triangular inferior BSTM(q, p) é usada para armazenar o índice da posição l associada com o par R_p e R_q no conjunto de possíveis pares de combinações $\{1, 2, \cdots, v(v-1)/2\}$. Tal índice indica a posição em que os atributos do melhor movimento estão armazenados. Por exemplo, RECM$(l, 1) = i$, RECM$(l, 2) = j$.

Em Glover e Laguna (1997) considera-se o problema da k-árvore mínima. Esse problema consiste em procurar uma árvore[1] de k arestas em um grafo $G = (V, E)$, com $E = \{(i, j), i, j \in V\}$, tal que a soma dos pesos dessas arestas seja mínima. Um movimento consiste em eliminar uma aresta (i, j) da árvore e substituí-la por outra do grafo, que não pertence à árvore, sujeita à condição de que o grafo resultante também seja uma árvore. Há dois tipos de trocas possíveis. O primeiro consiste em manter os mesmos vértices da árvore corrente, mudando-se apenas uma aresta. No segundo tipo, um novo vértice entra em lugar de outro da árvore. Para qualquer das situações (adição ou eliminação de arestas da árvore corrente), os autores sugerem a utilização de uma única matriz tabu T, de dimensões $n \times n$, em que $n = |V|$, para armazenar a iteração até a qual os movimentos de adição e eliminação das arestas permanecem tabu. Importante ressaltar que não há confusão nos dois estados possíveis de uma mesma aresta, já que em uma dada iteração uma aresta pertence ou não à árvore. Uma aresta (i, j) fica tabu-ativo, isto é, proibida de ser manipulada (adicionada, caso ela não pertença à árvore corrente; ou eliminada, caso contrário) na iteração *iter*, se $iter \leq T(i, j)$, sendo $T(i, j) = k + TempoTabu$, k a iteração na qual a aresta (i, j) tornou-se tabu-ativo e *TempoTabu* a duração da regra de proibição. Os autores chamam a atenção para o fato de que é possível atribuir uma duração tabu para a adição de uma aresta e outra, diferente, para a eliminação de uma aresta.

6. Tamanho da lista tabu

Em uma lista tabu, seja ela de soluções ou baseada em atributos de movimentos/soluções, o tamanho da lista indica a quantidade máxima de iterações em que cada solução ou atributo deve lá permanecer para cumprir seu papel de impedir o retorno a uma solução já visitada anteriormente. Por esse motivo, o tamanho da lista tabu é comumente referenciado como a duração das regras de proibição ou duração tabu. Discutiremos, nesta Seção, sobre a duração tabu ideal.

Uma duração pequena permite a exploração de soluções "próximas" à solução corrente, uma vez que poucos movimentos são proibidos; enquanto durações longas fazem com que a busca se "afaste" dessa. O controle da duração tabu permite, portanto, fazer um balanço entre examinar uma região mais profundamente (o que induz a uma estratégia de intensificação) e mover-se para outras regiões do espaço de busca (o que induz a uma estratégia de diversificação). A duração tabu deve ser longa o suficiente para prevenir a ocorrência de ciclagem e pequena o suficiente para não proibir muitos movimentos e parar o procedimento de busca muito prematuramente.

A duração de uma regra de proibição pode ser fixa durante toda a busca ou ser dinâmica, isto é, pode variar ao longo das iterações.

Nos anos 1980, muitos trabalhos científicos adotavam como duração tabu típica, o número "mágico" 7, sendo esse um valor ainda muito usado, como nos trabalhos de Ronconi e Armentano (1997) e Wan e Yen (2002) sobre sequenciamento. Outros autores preferem dimensionar empiricamente esse valor. Por exemplo, Gendreau et al. (1993) trataram o problema da clique máxima[2] por três estratégias de Busca Tabu, combinando lista tabu de soluções com lista tabu de atributos de movimentos. Para explorar o espaço de soluções, os autores usaram dois tipos de movimentos, consistindo um deles em inserir um vértice v na clique Q corrente e outro, em eliminar um vértice v de Q. Na primeira estratégia, consideraram uma única lista tabu contendo as últimas 100 soluções geradas; na segunda, usaram duas listas, sendo uma com as últimas 100 soluções geradas e outra, baseada em atributos, com os 5 últimos vértices eliminados. Na terceira estratégia, de forma semelhante à segunda, também foram usadas duas listas, mudando-se apenas o tamanho da lista de soluções para 150.

Boa parte dos pesquisadores considera a duração como dependente das características do problema

[1] Uma árvore é um grafo conexo (isto é, existe um caminho entre dois quaisquer de seus vértices) e sem ciclos

[2] Dado um grafo não orientado $G = (V, E)$, uma clique (ou subgrafo completo) é um grafo cujos vértices são todos adjacentes entre si. Em outras palavras, Q é uma clique de G se $Q \subseteq V(G)$ e para todo $u, v \in Q$ então $(u, v) \in E(G)$. No problema da clique máxima o objetivo é determinar o tamanho $\omega(G)$ da maior clique de G.

e do tamanho da instância considerada. Em Buscher e Shen (2009), por exemplo, os autores definem uma duração tabu dada por $|T| = 0,8 \times (n \times m \times s)^{0,5}$, em que n é o número de tarefas, m é o número de máquinas e s, o número de sublotes. Já em Osman (1993), são adotadas duas durações, uma para cada um dos dois algoritmos baseados em Busca Tabu desenvolvidos. Em uma delas, considera-se $|T| = 8 + (0,078 - 0,067 \times \rho) \times n \times v$, sendo n o número de consumidores, v o número de veículos e ρ, um parâmetro que representa a razão entre a soma das demandas dos consumidores e a soma das capacidades dos veículos usados nas melhores soluções da literatura até então. Em outra, $|T| = \max\{7, -40 + 0,6 \times ln(n \times v)\}$. Escolhe-se como duração tabu um dos três valores: $t_{\min} = 0,9 \times |T|$, $|T|$ ou $t_{\max} = 1,1 \times |T|$. Após isso, o valor t escolhido é mantido fixo por $2 \times t$ iterações, ao final das quais outro valor é escolhido aleatoriamente. Para resolver a mesma classe de problemas, Gendreau et al. (1994) escolheram uma duração tabu que varia dinamicamente no intervalo $[t_{\min}, t_{\max}]$, sendo $t_{min} = 0,9 \times n$ e $t_{max} = 1,1 \times n$. A seguir, essa duração é mantida fixa por $2 \times t_{\max}$ iterações.

A variação no valor da duração tabu ao longo da busca tem a vantagem de corrigir o erro porventura existente no dimensionamento empírico desse tempo. De fato, se houver ciclagem utilizando-se uma duração $|T|$, então variando-a, haverá alteração na quantidade de movimentos tabus e, assim, diferentes soluções poderão ser geradas. Com essa possibilidade de mudança na trajetória da busca no espaço de soluções, a ocorrência de ciclagem fica reduzida.

A lista tabu dinâmica, por sua vez, pode ser classificada em dois tipos: aleatória ou sistemática. Na aleatória, a duração tabu é selecionada aleatoriamente no intervalo $[t_{\min}, t_{\max}]$ e mantida fixa por $\alpha \times t_{\max}$ iterações, ao final das quais nova duração é sorteada. Outra possibilidade é selecionar uma duração tabu diferente a cada iteração. Na sistemática, a duração segue uma determinada regra previamente estabelecida. Por exemplo, em Ronconi e Armentano (1997), os autores escolhem a duração tabu aleatoriamente em um intervalo $[t_{\min}, t_{\max}]$ e aplicam essa duração como referência para as próximas 20 iterações, ao final das quais nova duração é sorteada. A seguir, essa duração base é ajustada na iteração corrente de acordo com o valor da solução vizinha. Se o valor da solução vizinha é maior que o da solução corrente (aquela a partir da qual foi aplicado o movimento), então a duração base é aumentada em uma unidade. Caso, no entanto, esse valor seja menor, então a duração é diminuída de uma unidade. Outra regra, apontada em França (2009), é definir uma sequência de números inteiros no intervalo $[t_{\min}, t_{\max}]$ em que a duração aumente e diminua de forma alternada. Exemplo: Dado o intervalo [5, 10], ter-se-ia uma sequência $S = \{5, 8, 6, 9, 7, 10\}$, indicando que na primeira iteração da aplicação dessa regra, a duração seria 5, depois aumentaria para 8, a seguir diminuiria para 6 e assim por diante.

Em Ronconi e Armentano (1997), os autores chamam a atenção para o fato de que a duração tabu depende também da regra de proibição e do atributo do movimento considerado. Em um problema de sequenciamento de n tarefas, uma regra que proíbe uma tarefa i de ser movida em um movimento de troca reduz para $n - 1$ o número de tarefas que podem ser trocadas após sua primeira aplicação. Considerando que se proíbe uma tarefa a cada iteração, e que para a realização do movimento de troca há necessidade de pelo menos duas tarefas liberadas, a última troca possível se realizará na iteração $n-1$. Dessa forma, $n-1$ é o limitante superior para a duração dessa regra de proibição. Uma alternativa para corrigir o erro de um limitante mal dimensionado é aplicar o critério de aspiração *default*, isto é, desativar a regra de proibição para o(s) atributo(s) tabu-ativo(s) mais antigo(s).

Os limitantes inferior e superior da duração tabu das regras de proibição relativas aos atributos dos movimentos considerados em Ronconi e Armentano (1997) são explicitados na Tabela 9.14.

Os autores observam, pela Tabela 9.14, que os limitantes superiores destacados em negrito foram determinados após um estudo sobre o número de iterações necessárias para que não mais houvesse movimentos disponíveis (não tabu) para serem escolhidos, considerando que, quando uma regra de proibição é ativada, ela permanece válida ao longo das iterações. Considera-se, além disso, que o critério de aspiração não está ativado. Os demais valores foram estimados por testes computacionais.

Tabela 9.14: Limitantes das durações tabus de Ronconi e Armentano (1997)

Movimento	Atributo	Regra	Limitante inferior	Limitante superior
Inserção	i	i não pode ser movida	$n/4$	$n/2$
	i	i não pode ser escolhida para inserção	$n/2$	**n**
	$i, s(i)$	i não pode retornar à posição $s(i)$	$4n$	$5n$
Troca	i, j	i e j não podem ser movidas	$n/4$	$n/2$
	i	i não pode ser movida	$n/2$	**n-1**
	$i, s(i)$	i não pode ser movida para posições $k \leq s(i)$	$n/2$	$3n/2$

Por esta tabela, observa-se que quanto mais restritiva é a regra de proibição, menor é o limitante superior.

Finalmente, ressalta-se que o tamanho da lista também pode depender do estágio da busca. Em problemas de programação de horários (*timetabling*), por exemplo, conforme relatado em Souza (2000), é recomendável aumentar o tamanho da lista quando a busca se encontra numa região com soluções de igual valor da função objetivo. Com essa estratégia, mais soluções dessa região do espaço ficam proibidas, dando oportunidade ao algoritmo de caminhar em outras direções. Ao sair dessa região, a lista volta ao seu tamanho normal. Em Buscher e Shen (2009), os autores aplicam uma estratégia baseada na proposta de Battiti e Tecchiolli (1994) para alterar o tamanho da lista quando a busca detecta a existência de soluções frequentemente visitadas. Basicamente, a duração tabu é progressivamente aumentada em *INC* unidades sempre que uma solução é visitada por mais do que *REP* vezes. Na ausência de repetições, a duração tabu é diminuída em *DEC* unidades. Detalhes dessa estratégia, que compõe a chamada *Busca Tabu Reativa*, são descritos em Battiti e Tecchiolli (1994) e Battiti (1996).

7. Critérios de aspiração

São apresentados, a seguir, alguns dos critérios de aspiração mais comuns.

Aspiração por objetivo global: Consiste em retirar o *status* tabu de um movimento se for produzida uma solução com a melhor avaliação global.

Aspiração por objetivo regional: Um movimento tabu perde seu *status* quando for gerada uma solução melhor que a melhor encontrada na região atual de busca. Uma forma de se delimitar a região atual de busca é registrar a melhor solução encontrada em um passado recente e utilizar o valor dessa solução como critério para aspiração. Por exemplo, considerando um problema de minimização e supondo $f(s^\star_R)$ a melhor solução encontrada nas últimas *ContIterRegiao* iterações, então um movimento tabu que guia a uma solução s' tal que $f(s') < f(s^\star_R)$ pode ser realizado. Nessa estratégia de aspiracão, *ContIterRegiao* é um parâmetro.

Aspiração *Default*: Se todos os movimentos possíveis são tabus e não é possível aplicar outro critério de aspiração, então o movimento mais antigo perde sua condição tabu. Em uma implementação baseada em tempo de permanência na lista, como o exemplificado para o problema da mochila, se $T(j) \geq iter\ \forall j$, em que *iter* representa a iteração atual, então a inversão do bit da posição k, tal que $T(k) = \min\{T(j),\ \forall j\}$, pode ser realizada, apesar de tabu.

Há outros critérios de aspiração mais sofisticados, como aqueles desenvolvidos em de Werra e Hertz (1989) e Hertz e de Werra (1990). No entanto, apesar de bem sucedidos em suas aplicações, eles raramente são usados. O critério mais usado, presente na quase totalidade das implementações baseadas em Busca Tabu, é o critério de aspiração por objetivo global.

Uma implementação BT precisa prever, sempre que possível, a aplicação do critério de aspiração *default*. Por exemplo, no PM01 considerado na Seção 1, trabalhamos apenas com soluções factíveis. Ainda que sejam levados em consideração os limitantes do tamanho da lista (como os mostrados na Tabela 9.14), haveria a possibilidade de todos os vizinhos factíveis de uma dada solução serem tabus. Nesse caso, a aplicação do critério de aspiração *default* é uma alternativa para a Busca Tabu prosseguir na exploração do espaço de soluções.

8. Memória de Longo Prazo

Uma memória de curto prazo não é suficiente para evitar que a busca fique presa em certas regiões do espaço de soluções. Assim, alguma estratégia de diversificação é necessária. Para cumprir esse objetivo, algoritmos Busca Tabu usam uma memória de longo prazo para encorajar o processo de busca a explorar regiões ainda não visitadas. A vantagem em se utilizar tal estratégia ao invés de promover uma simples reinicialização do algoritmo é que, com o uso de memória de longo prazo, diminui-se o risco de voltar a visitar uma mesma região do espaço de soluções, situação que poderia ocorrer no processo de reinicialização em vista da perda das informações anteriores.

A memória de longo prazo também pode visar à aplicação de uma estratégia de intensificação. Ao contrário da diversificação, o objetivo agora é estimular a busca a gerar soluções que contenham atributos encontrados nas soluções já visitadas que sejam historicamente bons.

A memória de longo prazo pode ser baseada em frequência de transição ou em frequência de residência. Esses tipos de memória são apresentados a seguir.

Memória baseada em frequência de transição

A ideia aqui é usar uma estrutura de memória para computar atributos que mudam de uma solução visitada para outra.

Em Santos et al. (2005) é desenvolvido um algoritmo BT para resolver o problema de programação de horários em escolas (PPHE). No problema considerado, há um conjunto $T = \{1, \cdots, t\}$ de professores, um conjunto $C = \{1, \cdots, c\}$ de turmas e um conjunto $P = \{1, \cdots, p\}$ de horários para a realização das aulas. O objetivo é alocar as aulas dos professores às turmas minimizando o número de horários ociosos na programação semanal do professor e o número de vezes em que cada professor vai à escola, bem como maximizar o atendimento ao número de aulas geminadas requeridas pelos professores para cada turma.

Um quadro de horários é representado por uma matriz $Q_{t \times p}$, em que cada linha representa a programação de aulas de um professor. Uma célula $q_{ik} \in \{0, 1, \cdots, c\}$ indica a alocação do professor i no horário k, sendo que $q_{ik} = j$, com $j \in \{1, \cdots, c\}$, representa a turma para a qual o professor i está lecionando e $q_{ik} = 0$ significa que o professor i está disponível naquele horário.

Na Figura 9.3 mostra-se um fragmento de um quadro de horários envolvendo 5 professores e p horários. A letra "X" indica que o professor está indisponível no horário.

Pela Figura 9.3, o professor 5 está disponível nos dois primeiros horários, dá aula para a turma 2 no terceiro horário e para a turma 3 no quarto horário e está indisponível no quinto e sexto horários.

Os autores exploram o espaço de soluções por meio de movimentos baseados na troca de duas alocações feitas na programação de horários de um professor. Um movimento envolvendo os horários p_1 e p_2 de um professor i é representado por um tripla $\langle i, p_1, p_2 \rangle$, em que $q_{i,p_1} \neq q_{i,p_2}$, $p_1 < p_2$ e $p_1, p_2 \in \{1, \cdots, p\}$. Como o movimento pode dar origem a um quadro de horários infactível, a função

	Horários							
Professor	1	2	3	4	5	6	$\cdots$	p
1	1	0	0	2	2	X	$\cdots$	$\cdots$
2	0	X	X	0	1	2	$\cdots$	$\cdots$
3	X	X	1	0	3	1	$\cdots$	$\cdots$
4	0	1	0	1	0	3	$\cdots$	$\cdots$
5	0	0	2	3	X	X	$\cdots$	$\cdots$

Figura 9.3: Fragmento de um quadro de horários

de avaliação adotada é a soma da função objetivo propriamente dita com uma parcela que mensura o nível de inviabilidade de uma solução.

Uma vez realizado o movimento, ele é mantido tabu pelas próximas *TempoTabu* iterações. Essa duração tabu não é fixa, mas sim variável, sendo escolhida aleatoriamente dentro de um intervalo $[t_{\min}, t_{\max}]$, em que os limitantes são definidos pelas equações (9.6) e (9.7):

$$t_{\min} = \lfloor \mathit{TempoTabuCentral} - \varphi \times \mathit{TempoTabuCentral} \rfloor \tag{9.6}$$

$$t_{\max} = \lceil \mathit{TempoTabuCentral} + \varphi \times \mathit{TempoTabuCentral} \rceil \tag{9.7}$$

No cálculo dessas durações mínima e máxima, *TempoTabuCentral* é um parâmetro de entrada que estima a duração tabu de um movimento e $\varphi \in [0, 1]$, outro parâmetro de entrada que define a variação permitida nessa duração.

Os autores armazenam em uma matriz $Z_{t\times c}$ o número z_{ij} de movimentos feitos envolvendo um professor i e uma turma j. Usando esses valores e considerando $\bar{z} = \max\{z_{ij}, i \in T, j \in C\}$, é definida uma medida de frequência relativa de transição ϵ_{ij}, computada conforme a Eq. (9.8):

$$\epsilon_{ij} = z_{ij}/\bar{z} \tag{9.8}$$

Como um movimento $\langle i, p_1, p_2 \rangle$ envolvendo os horários p_1 e p_2 da agenda do professor i pode ser relativo a dois horários de aula ou a um horário de aula e um horário livre, então a penalidade ψ_{i,j_1,j_2} associada ao movimento é calculada com base na Eq. (9.9), a qual leva em consideração as três situações possíveis:

$$\psi_{i,j_1,j_2} = \begin{cases} \epsilon_{i,j_1} \times f(Q^\star) & \text{se } j_1 \neq 0 \text{ e } j_2 = 0 \\ \epsilon_{i,j_2} \times f(Q^\star) & \text{se } j_1 = 0 \text{ e } j_2 \neq 0 \\ (\epsilon_{i,j_1} + \epsilon_{i,j_2})/2 \times f(Q^\star) & \text{se } j_1 \neq 0 \text{ e } j_2 \neq 0 \end{cases} \tag{9.9}$$

Na Eq. (9.9), $f(Q^\star)$ é o custo da melhor solução encontrada até então, j_1 é a alocação do professor i no horário p_1 (isto é, $j_1 = q_{i,p_1}$) e j_2 é a alocação do professor i no horário p_2 (ou seja, $j_2 = q_{i,p_2}$).

O Algoritmo 2 mostra o pseudocódigo implementado. Nesse algoritmo, a atualização da lista tabu é feita atribuindo-se ao movimento realizado uma duração aleatória calculada no intervalo definido pelas equações (9.6) e (9.7). A penalização de um movimento (linha 10) é calculada com base na Eq. (9.9). Toda vez que é realizado um movimento, a lista tabu de longo prazo é atualizada (linha 21) e sempre que a melhor solução é modificada, ela é reinicializada (linha 26). A estratégia de diversificação é acionada após um número *IterAcionaDiv* de iterações sem melhora e fica ativa por *IterDivAtiva* iterações (linha 9). *IterDivAtiva* e *IterAcionaDiv* são parâmetros do algoritmo.

Algoritmo 2 BT-PPHE

Entrada: Q, *IterAcionaDiv*, *IterDivAtiva*, *TempoTabuCentral*, φ
Saída: $Q^\star$
$Q^* \leftarrow Q$; *InicializeListaTabu*; *IterSemMelhora* $\leftarrow 0$; *Iter* $\leftarrow 0$;
InicializeMemoriaLongoPrazo();
repita
  $\Delta \leftarrow \infty$; *Iter*++; *MelhorMovimento* $\leftarrow$ *MovimentoAleatorio*();
  para todo movimento m tal que $(Q \oplus m) \in \mathcal{N}(Q)$ **faça**
    Penalidade $\leftarrow 0$;
    se *IterSemMelhora* **mod** *IterAcionaDiv* $<$ *IterDivAtiva*
    e *iter* $\geq$ *IterAcionaDiv* **então**
      Penalidade $\leftarrow$ *CalculePenalidade*(m);
    fim se
    $\Delta' \leftarrow f(Q) - f(Q \oplus m)$;
    se $(\Delta' + Penalidade < \Delta$ **e** m não tabu) **ou**
    $(f(Q \oplus m) < f(Q^\star)$ **e** $\Delta' < \Delta)$ **então**
      MelhorMovimento $\leftarrow m$;
      $\Delta \leftarrow \Delta'$;
      se $f(Q \oplus m) \geq f(Q^\star)$ **então**
        $\Delta \leftarrow \Delta +$ *Penalidade*;
      fim se
    fim se
  fim para
  AtualizeMemoriaLongoPrazo(*MelhorMovimento*, Q);
  $Q \leftarrow Q \oplus$ *MelhorMovimento*;
  AtualizeListaTabu(*MelhorMovimento*, *Iter*);
  se $f(Q) < f(Q^\star)$ **então**
    $Q^\star \leftarrow Q$; *IterSemMelhora* $\leftarrow 0$;
    InicializeMemoriaLongoPrazo();
  senão
    IterSemMelhora ++;
  fim se
até Critério de parada atingido;

Memória baseada em frequência de residência

Em uma memória de longo prazo baseada em frequência de residência a ideia é computar os atributos que são frequentes nas soluções visitadas para usá-los como estratégia de intensificação (estimulando-os a comporem a solução corrente) e/ou como uma estratégia para diversificar a busca (neste caso, penalizando-os para desestimular seu uso).

Em Santos et al. (2005), os autores também desenvolveram um algoritmo BT explorando memória de longo prazo baseada em frequência de residência. A medida de residência é armazenada em uma matriz $Y_{t \times c \times u \times p}$, em que $u = \max\{r_{ij}, i \in T, j \in C\}$, t é o número de professores, c é o número de turmas, p é o número de horários de aula e r_{ij} o número de aulas do professor i para a turma j. Uma medida de frequência relativa de residência da m-ésima aula do professor i para a turma j no horário k é calculada com base na Eq. (9.10):

$$\eta_{ijmk} = \frac{y_{ijmk}}{\bar{y}} \tag{9.10}$$

sendo $\bar{y} = \max\{y_{ijmk}, i \in T, j \in C, m \in \{1, 2, \cdots, u\}, k \in P\}$.

A penalidade μ_{ijmk} por alocar a m-ésima aula do professor i para a turma j no horário k é calculada conforme a Eq. (9.11):

$$\mu_{ijmk} = y_{ijmk} \times f(Q) \tag{9.11}$$

O algoritmo BT desenvolvido é o mesmo da Seção anterior, diferenciando-se daquele apenas na forma de calcular a penalização do movimento (linha 10 do Algoritmo 2), a qual é feita considerando-se a Eq. (9.11).

Em Díaz e Fernández (2001), os autores trataram o problema generalizado de atribuição (PGA) por meio de um algoritmo BT que faz uso de memórias de curto e longo prazos. No PGA há um conjunto $J = \{1, \cdots, n\}$ de tarefas que devem ser processadas por um conjunto $I = \{1, \cdots, m\}$ de agentes. Cada tarefa j consome a_{ij} unidades de recurso do agente i, o qual, por sua vez, tem capacidade de processar apenas b_i unidades do recurso. A designação de uma tarefa j a um agente i custa c_{ij} unidades e o objetivo é fazer a alocação de menor custo, de sorte que cada tarefa seja atribuída a um único agente e que a capacidade de cada agente seja respeitada.

O PGA pode ser formulado como o modelo de programação linear inteira, dado pelas equações (9.12) - (9.15). Nesse modelo, a variável x_{ij} assume valor 1 se a tarefa j é designada ao agente i e 0, caso contrário. A função objetivo, dada pela Eq. (9.12), computa o custo das designações. As restrições (9.13) asseguram que cada tarefa é executada por um único agente. As restrições (9.14) garantem que a disponibilidade de recurso de cada agente seja respeitada.

$$\min \quad z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{9.12}$$

$$\text{s. a:} \quad \sum_{i \in I} x_{ij} = 1 \qquad \forall\, j \in J \tag{9.13}$$

$$\sum_{j \in J} a_{ij} x_{ij} \leq b_i \qquad \forall\, i \in I \tag{9.14}$$

$$x_{ij} \in \{0, 1\} \quad \forall\, i \in I, j \in J \tag{9.15}$$

No procedimento proposto, o custo de designação de uma tarefa j a um agente i é calculado de forma relativa pela Eq. (9.16), em que se toma como referência o menor custo de atribuição da tarefa j.

$$\Delta_{ij} = c_{ij} - \min\{c_{ij}, i \in I\} \tag{9.16}$$

A função objetivo pode, então, ser reescrita como:

$$\min \quad z = \sum_{j \in J} \min\{c_{ij}, i \in I\} + \sum_{i \in I} \sum_{j \in J} \Delta_{ij} x_{ij} \tag{9.17}$$

Uma vez que a primeira parcela da Eq. (9.17) é constante, ela pode ser removida, resultando na função objetivo (9.18).

$$\min \quad z' = \sum_{i \in I} \sum_{j \in J} \Delta_{ij} x_{ij} \tag{9.18}$$

Para explorar o espaço de soluções, os autores relaxam as restrições de capacidade dos agentes (Eq. (9.14)), penalizando-as na função objetivo, a qual passa a ser expressa pela Eq. (9.19).

$$\min \quad z'' = \sum_{i \in I} \sum_{j \in J} \Delta_{ij} x_{ij} + \rho \times \sum_{i \in I} e_i \tag{9.19}$$

Na Eq. (9.19), a parcela $e_i = \max\{\sum_{j \in J} a_{ij} x_{ij} - b_i, 0\}$ mensura a quantidade de violação de capacidade do agente i e ρ é um fator de penalidade.

Essa relaxação é utilizada porque, segundo os autores, as restrições (9.14) são difíceis de serem atendidas, pois confinam a busca a uma região estreita do conjunto de soluções factíveis. Desta forma, utilizando-se apenas movimentos simples há poucas soluções vizinhas factíveis, forçando a busca a terminar rapidamente, em geral com soluções de baixa qualidade. Por outro lado, permitindo-se alguma violação de viabilidade, há duas vantagens. Primeiro, permite-se a execução de movimentos menos complexos que aqueles que seriam necessários. Segundo, dá-se à busca um grau maior de flexibilidade e uma região mais ampla do espaço de soluções pode ser então explorada. O desvio de viabilidade é controlado pelo fator de penalidade ρ na função objetivo. Essa mesma estratégia de penalização é adotada por Yagiura et al. (2006) para o PGA.

Uma solução do PGA é representada por um vetor s de n elementos, em que o k-ésimo componente é o agente designado para a execução da tarefa j, isto é:

$$s = (s_1, \cdots, s_n), \text{ em que } s_j \in I; s_j = i \Longleftrightarrow x_{ij} = 1$$

Exemplificando, seja $I = \{A, B, C\}$ e $J = \{1, 2, 3, 4, 5\}$. Uma solução s para o PGA desse exemplo pode ser $s = (C, A, C, B, A)$, o que significa que as tarefas 1 e 3 são executadas pelo agente C, as tarefas 2 e 5 pelo agente A e a tarefa 4, pelo agente B.

São usados dois tipos de movimento, sendo um baseado em troca e outro em substituição, para definir as vizinhanças $N^T(.)$ e $N^S(.)$, respectivamente. No primeiro tipo, dadas duas tarefas distintas j_1 e j_2, são permutados os agentes s_{j_1} e s_{j_2} que as executam, desde que eles não sejam os mesmos, isto é, $s_{j_1} \neq s_{j_2}$. No movimento de substituição, a atribuição de uma tarefa j é alterada de um agente i_1 para outro i_2, sendo $i_1 \neq i_2$. No exemplo considerado, $s'_1 = (A, C, C, B, A)$ é um vizinho obtido pelo movimento de troca, enquanto $s'_2 = (B, A, C, B, A)$, pelo de substituição. Como há $m-1$ possibilidades para substituir o agente que executa uma dada tarefa e há n tarefas, então na vizinhança de substituição há $(m-1)n$ vizinhos. Na vizinhança de troca, por sua vez, há um máximo de $n(n-1)/2$ vizinhos.

Os autores observam que os movimentos de troca não alteram a quantidade de tarefas designadas para cada agente, isto é, se um agente executa, digamos, duas tarefas, continuará a executar duas tarefas e o que muda são as tarefas atribuídas. Por outro lado, movimentos de substituição são mais restritivos em termos de factibilidade. Assim, adotaram como vizinhança de uma solução s a união das vizinhanças de troca e substituição, ou seja, $N(s) = N^T(s) \cup N^S(s)$, por ser uma opção mais abrangente.

Ao se realizar um movimento, seja ele de troca ou substituição, considera-se como atributo o par $\langle i, j \rangle$, em que i é o agente designado para executar a tarefa j. Como memória de curto prazo, considera-se uma matriz tabu T que armazena na posição (i, j) a iteração até a qual tal atributo estará proibido. Exemplo: se durante uma dada iteração k, uma tarefa j do agente i_1 for realocada a outro agente i_2, então o par $\langle i_1, j \rangle$ permanecerá tabu nas próximas *TempoTabu* iterações. Isso é implementado fazendo-se $T_{i_1,j} = k + \textit{TempoTabu}$. No caso de movimentos de troca, em que se envolvem dois pares $\langle i_1, j \rangle$ e $\langle i_2, j \rangle$, com $i_1 \neq i_2$, somente um deles é registrado como atributo tabu-ativo, no caso, aquele associado ao maior valor de custo ($\Delta_{i_1,j}$ ou $\Delta_{i_2,j}$).

Os autores utilizam uma memória de longo prazo baseada em frequência de residência tanto para aplicar uma estratégia de intensificação quanto de diversificação. Durante a busca, armazena-se em uma matriz *fr* o número de vezes em que uma tarefa j ficou alocada ao agente i, tendo em vista o total de soluções visitadas até aquele ponto.

Para a aplicação da fase de intensificação, toma-se a melhor solução encontrada até então e verificam-se as designações de agentes a tarefas dessa solução. Se um agente ficou designado com muita frequência a uma tarefa, no caso 85%, no mínimo, essa atribuição é considerada fixa. Segundo os autores, as atribuições de alta frequência podem ser consideradas boas dado que: primeiro, elas estão presentes na melhor solução encontrada até então. Segundo, se elas aparecem na maioria das soluções geradas até aquela iteração, é porque elas foram selecionadas muitas vezes ou, após selecionadas, elas não foram trocadas por um longo período de tempo. Após a fixação de algumas tarefas a agentes, dá-se início à fase de intensificação, acionando-se a chamada a um procedimento que usa memória de curto prazo. Durante esse período, a matriz fr continua a ser atualizada. A vantagem dessa estratégia é que, com a fixacão de algumas designações, a fase de intensificação atua em um problema de menor dimensão, possibilitando, assim, realizar uma busca mais efetiva nesse espaço.

Para a aplicação da fase de diversificação, à matriz de custo Δ é adicionada uma parcela de penalidade fr que objetiva desestimular atribuições de tarefas a agentes realizadas com muita frequência. Nessa fase, todas as designações voltam a se tornar livres e a matriz de custo é alterada, inicialmente, para $\Delta + fr$. A seguir, aplica-se o procedimento de busca baseado em memória de curto prazo por um número reduzido de iterações. Finalizado esse procedimento, aciona-se novamente o procedimento de busca baseado em memória de curto prazo, mas desta vez usando-se a matriz de custo Δ original.

O Algoritmo 3 mostra o pseudocódigo do algoritmo de Busca Tabu desenvolvido por Díaz e Fernández (2001) para resolver o PGA. Nesse algoritmo, s_0 é a solução inicial, l é o número de iterações em que são aplicadas as fases de intensificação e diversificação e k indica o número de soluções geradas.

Algoritmo 3 `BT-PGA`

1: **Entrada**: s_0, l
2: **Saída**: $s^\star$
3: $T \leftarrow 0$; $fr \leftarrow 0$; $s^* \leftarrow s_0$; $k \leftarrow 0$;
4: Gere solução inicial s_0;
5: Aplique *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ);
6: **para todo** $iter = 1$ até l **faça**
7: { Fase de intensificação }
8: $s^* \leftarrow s^k$;
9: **para todo** tarefa j de s^* **faça**
10: **se** $fr(s^\star_j, j) > 0,85 \times k$ **então**
11: Fixe a tarefa j ao agente i, isto é, faça $x_{s^\star_j, j} \leftarrow 1$;
12: **fim se**
13: **fim para**
14: Aplique *MemoriaCurtoPrazo*(k,s^k);
15: { Fase de diversificação }
16: Libere todas as atribuições;
17: Aplique *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ) por poucas iterações usando $\Delta + fr$ como matriz de custo;
18: Recupere a matriz de custo Δ original;
19: Aplique *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ);
20: **fim para**

O pseudocódigo do procedimento *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ) é apresentado no Algoritmo 4. Nesse algoritmo, o custo de uma solução assume valor $\Delta + fr$ quando ele é acionado para executar a estratégia de diversificação (linha 17 do Algoritmo 3) e valor Δ nos demais casos.

Algoritmo 4 `MemoriaCurtoPrazo`$(k, s^k, s^\star, T, fr, \Delta)$

1: **enquanto** Critério de parada não atingido **faça**
2: Selecione um vizinho $s^{k+1} \in N(s^k)$;
3: **se** $custo(s^{k+1}) < custo(s^\star)$ **então**
4: $s^\star \leftarrow s^{k+1}$;
5: **fim se**
6: Atualize a memória de curto prazo T;
7: Atualize a memória de longo prazo fr;
8: $k \leftarrow k + 1$;
9: **fim enquanto**

Em Laguna (1994) é apresentado um guia bastante didático para a implementação de um algoritmo baseado em Busca Tabu, usando como referência o problema das n-rainhas. Esse problema consiste em encontrar uma disposição de n rainhas do jogo de xadrez em um tabuleiro $n \times n$, de tal modo que nenhuma rainha ataque (colida com) as outras de acordo com as regras do jogo. Uma rainha ataca outra quando elas estão posicionadas na mesma linha ou mesma coluna ou mesma diagonal.

Uma solução do problema é representada por um vetor s de n posições, em que cada célula s_i representa a coluna j em que a rainha da linha i ocupa. Portanto, o par $(i, s_i) = (i, j)$ representa a posição de uma rainha.

A Figura 9.4 ilustra a disposição das rainhas em um problema de 6 rainhas, com a configuração $s = (2, 3, 5, 1, 6, 4)$.

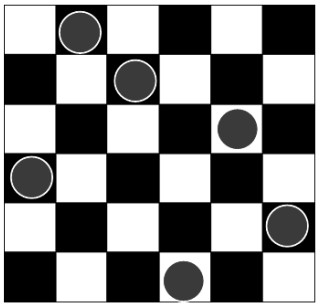


Figura 9.4: Representação de uma solução no Problema das n-Rainhas

Pela Figura 9.4, observa-se que a rainha da primeira linha está na segunda coluna, a da segunda linha está na terceira coluna e assim por diante. Com esta representação, não há colisão de rainhas em linhas ou colunas, apenas em diagonais. Assim, a função objetivo, isto é, o número de colisões, pode ser determinado calculando-se apenas as colisões nas diagonais.

Para calcular o número de colisões nas diagonais, inicialmente são criados dois vetores d_n e d_p, cada qual contendo $2n - 1$ elementos, o primeiro para representar as diagonais negativas e o segundo, as diagonais positivas. Dada uma rainha na linha i e coluna s_i, o índice k de um elemento da diagonal negativa é obtido fazendo-se $k = i + s_i$, enquanto o índice k de um elemento da diagonal positiva é obtida fazendo-se $k = i - s_i$. As rainhas 2 e 4, por exemplo, estão na diagonal negativa de índice 5. De fato, $2 + s_2 = 2 + 3 = 4 + s_4 = 4 + 1 = 5$. De forma semelhante, as rainhas 1, 2 e 6 estão na diagonal positiva de índice -1 pois $1 - s_1 = 1 - 2 = 2 - s_2 = 2 - 3 = 5 - s_5 = 5 - 6 =$ -1. Desta forma, os índices das diagonais positivas variam no intervalo $[1 - n, n - 1]$, enquanto os índices das diagonais negativas pertencem ao intervalo $[2, 2n]$.

O número de rainhas em cada diagonal é, inicialmente, fixado em 0 para todas as diagonais positivas

e negativas, conforme o código a seguir.

para $(i = 2, \cdots, 2n)$ faça
    $d_n(i) \leftarrow 0$;
    $d_p(i - n - 1) \leftarrow 0$;
fim-para

Em seguida, são contabilizados o número de rainhas em cada uma das diagonais (positiva e negativa):

para $(i = 1, \cdots, n)$ faça
    $d_n(i + s(i)) \leftarrow d_n(i + s(i)) + 1$;
    $d_p(i - s(i)) \leftarrow d_p(i - s(i)) + 1$;
fim-para

O número de colisões de uma solução s pode ser, então, determinado pela Eq. (9.20):

$$f(s) = \sum_{k=1-n}^{n-1} \max\{0, d_p(k) - 1\} + \sum_{k=2}^{2n} \max\{0, d_n(k) - 1\} \tag{9.20}$$

Observa-se que o número de colisões em uma dada diagonal é o número de rainhas nessa diagonal menos uma unidade.

A exploração do espaço de soluções do problema é feita utilizando-se movimentos de troca de coluna entre duas rainhas posicionadas em linhas distintas i e j. Adota-se como atributo tabu o par $\langle i, j\rangle$. Exemplo: dada a configuração $s = (2, 4, 6, 5, 7, 3, 1)$ e o movimento de troca das rainhas das linhas 3 e 5, gera-se a solução $s' = (2, 4, 7, 5, 6, 3, 1)$, sendo tabu o movimento $\langle 3, 5\rangle$. Armazena-se na parte triangular superior de uma matriz T, na posição T_{ij} (considerando $i < j$), a iteração até a qual tal movimento permanece tabu. As informações contidas nesta matriz triangular superior são usadas como memória de curto prazo para a Busca Tabu.

Toda vez que é realizada uma troca de colunas envolvendo as rainhas das linhas i e j, é computada na parte triangular inferior dessa matriz T, na posição T_{ji} (considerando $i < j$), o número de trocas realizadas até então. Essas informações representam a memória de longo prazo baseada em frequência de transição e são usadas pelo autor apenas quando não há movimentos de melhora, servindo para diferenciar tais movimentos.

Na Tabela 9.15 mostram-se as memórias de curto e longo prazos, considerando que a iteração atual é $iter = 26$, que a duração tabu de um movimento é 3 e que a solução corrente é $s = (1, 3, 6, 2, 7, 5, 4)$. Para clareza de apresentação, estão representadas na matriz apenas as células que representam movimentos tabu-ativos. Na Tabela 9.16 mostram-se as cinco melhores trocas, o valor do movimento e esse valor penalizado, isto é, a soma do valor atual do movimento com a frequência de trocas armazenada na parte inferior da matriz T.

Exemplificando, a troca $\langle 2, 7\rangle$, que tem valor de movimento igual a 1, é agora penalizada em mais quatro unidades, uma vez que na parte inferior da matriz T, tem-se $T_{72} = 4$. Ou seja, como as rainhas 2 e 7 trocaram de posição 4 vezes durante o histórico da busca, então esse valor é usado para penalizar tal movimento.

No exemplo apontado, a memória baseada em frequência tem influência decisiva na diferenciação dos movimentos de mesmo valor. De fato, o movimento com melhor avaliação, o qual envolve a troca de coluna das rainhas das linhas 1 e 6 e tem valor 0, é tabu, pois $T_{16} = 29 \geq 26 = iter$. Com a penalização adotada, o melhor movimento passa a ser a troca de coluna das rainhas das linhas 1 e 5 (troca destacada em negrito na Tabela 9.16).

Tabela 9.15: Estrutura de dados n-Rainhas

	1	2	3	4	5	6	7
1						29	
2							
3	5					28	
4		3					27
5	2			4			
6	3				2		
7		4	3	1			

Tabela 9.16: Melhores candidatos

Troca		Valor	V. penalizado
1	6	0	1
1	3	1	6
1	**5**	1	**3**
2	7	1	5
3	7	1	4

Como o custo da análise da vizinhança inteira é $O(n^2)$, o autor também propõe a utilização de uma lista de candidatos. A ideia proposta é criar uma lista envolvendo apenas as rainhas que colidem, e reconstruir essa lista por uma busca completa depois de um certo tempo. Exemplo: supondo que rainhas 1, 5 e 7 são as únicas a participar de colisões no problema das 7-rainhas, nas próximas 4 iterações, por exemplo, a lista de candidatos consistiria de trocas envolvendo apenas tais rainhas. Após isso, seria feita busca na vizinhança inteira para reconstruir a lista das próximas rainhas a participar de colisões.

O autor propõe, ainda, a utilização de uma memória de longo prazo baseada em frequência de residência como estratégia de intensificação. A ideia, no caso, seria criar uma segunda matriz, nomeada *FreqBoasSol*, e contabilizar nela apenas os atributos das melhores soluções encontradas, no caso, daquelas que efetivamente estiverem a uma certa distância do valor da melhor solução encontrada até então. Mais precisamente, após um movimento ser selecionado, se o custo da solução gerada (*custoVizinho*) estiver pouco distante (abaixo até $p\%$ do valor da melhor solução até então, denotado por *MelhorCusto*), então os atributos dessa solução gerada seriam contabilizados. O atributo considerado é o par $\langle i, s_i \rangle$, em que i indica a linha em que está uma rainha e s_i a sua coluna. Representa-se, a seguir, o trecho de código para atualizar essa memória de longo prazo.

```
se (custoVizinho < (1 + p) × MelhorCusto) então
    para (i = 1, ··· , n) faça
        FreqBoasSol(i, s(i)) ← FreqBoasSol(i, s(i)) + 1;
    fim-para
fim-se
```

Para exemplificar, seja a solução $s = (2, 4, 6, 5, 7, 3, 1)$ e o movimento de troca envolvendo as rainhas das linhas 3 e 5. Supondo que a solução vizinha $s' = (2, 4, 7, 5, 6, 3, 1)$, resultante desse movimento, satisfaça à condição anterior, então seriam aumentados em uma unidade os valores contidos nas seguintes células da matriz *FreqBoasSol*: (1, 2), (2, 4), (3, 7), (4, 5), (5, 6), (6, 3) e (7, 1).

A fase de intensificação seria acionada em certas ocasiões, como por exemplo, depois de um dado tempo sem melhora. Nessa situação, a matriz *FreqBoasSol* poderia ser usada para proibir trocas de rainhas localizadas em colunas com alta frequência. Dito de outra maneira, as rainhas que estivessem em colunas com elevada frequência permaneceriam fixas em suas posições, e as demais estariam livres para a realização de movimentos. Assim, a busca se restringiria a trocas que envolvessem apenas as rainhas que estivessem em colunas livres.

9. Oscilação estratégica

Para muitos problemas combinatórios, pode não ser eficaz explorar o espaço de soluções por meio apenas de movimentos que respeitem todas as restrições do problema. Isso acontece, por exemplo, em

problemas de roteamento de veículos, nas situações em que os veículos estejam operando próximos à capacidade máxima, limitando, com isso, a movimentação de clientes entre as rotas (Gendreau, 2003). Nesses casos, a relaxação de restrições pode ser uma estratégia atrativa, uma vez que ela cria um espaço de soluções mais amplo, que pode ser explorado por movimentos mais simples. Para implementar essa estratégia, basta desconsiderar a satisfação das restrições selecionadas do espaço de busca e penalizá-las na função objetivo sempre que elas forem violadas.

Quando os pesos atribuídos às violações das restrições relaxadas são sistematicamente modificados ao longo da busca, isto é, ora aumentados, ora diminuídos, tem-se a chamada "oscilação estratégica" ou "relaxação adaptativa". Ao aumentar a penalização de movimentos que conduzem a soluções infactíveis, estimula-se o procedimento de busca a entrar na região de soluções factíveis. Ao contrário, quando tal penalização é reduzida ou mesmo anulada, estimula-se o procedimento a caminhar no conjunto de soluções infactíveis. Portanto, o uso dessa estratégia permite à busca alternar entre soluções factíveis e infactíveis, daí o nome "oscilação". De acordo com Gendreau (2003), tal estratégia teve origem no trabalho de Glover (1977).

Em Gendreau et al. (1994), os autores tratam o problema de roteamento de veículos penalizando o excesso de carga encontrado em uma solução por um peso ρ, o qual é multiplicado por um fator α que varia de acordo com o seguinte esquema:

1. No início da busca $\alpha \leftarrow 1$.
2. A cada k iterações sem melhora:
   - se todas as k soluções visitadas forem factíveis então $\alpha \leftarrow \alpha/\gamma$;
   - se todas as k soluções visitadas forem infactíveis então $\alpha \leftarrow \alpha \times \gamma$;
   - se algumas soluções forem factíveis e outras infactíveis, então α permanecerá inalterado.

O parâmetro γ é fixado em 2. Assim, quando somente soluções factíveis são geradas, o peso ρ é reduzido, estimulando a geração de soluções infactíveis. Ao contrário, quando somente soluções infactíveis são geradas, o peso ρ é aumentado, encorajando a busca a entrar novamente no conjunto das soluções factíveis. O valor de α é limitado por duas constantes $\alpha_{\min}$ e $\alpha_{\max}$, para evitar que a estratégia de oscilação aumente (ou diminua) indefinidamente os pesos.

Estratégia semelhante pode ser aplicada ao problema da mochila 0-1 tratado na Seção 1. Nesse caso, usaríamos a função (9.4), apresentada à página 179, para avaliar uma solução. A penalidade ρ relativa à existência de uma solução infactível seria, assim como no exemplo anterior, multiplicada por um fator α.

Em Díaz e Fernández (2001), os autores variam o peso atribuído ao parâmetro ρ da Eq. (9.19) de acordo com a expressão (9.21).

$$\rho \leftarrow \rho \times \alpha^{(ninv/(niter-1)-1)} \tag{9.21}$$

em que *ninv* é o número de soluções infactíveis obtidas durante as últimas *niter* iterações e α é um parâmetro que varia conforme a seguir se descreve. Essa expressão busca adaptar o parâmetro ρ ao estágio da busca, incorporando uma memória de curto prazo (*ninv*) e uma memória de médio prazo (α).

Para um dado valor atribuído a α, o valor de ρ varia no intervalo $[\alpha^{-1} \times \rho, \alpha^{1/(niter-1)} \times \rho]$. Como $\alpha \geq 1$, então ρ somente aumenta de valor quando *todas* as soluções encontradas nas últimas *niter* iterações são infactíveis. Contudo, mesmo nesse caso, o limite superior do intervalo faz com que o acréscimo no valor de ρ seja bem pequeno.

Quando $\alpha = 1$ na expressão (9.21), ρ se torna fixo durante todo o processo e a história da busca não é levada em consideração. No caso em que $\alpha = 2$, a expressão obtida é bastante similar à de

Gendreau et al. (1996). Observa-se que para α fixo durante a busca, apenas memória de curto prazo é usada.

Os autores observam que o valor atribuído a α influencia consideravelmente o progresso da busca. Quanto maior for este valor, maior é o intervalo de variação para ρ. Portanto, quando somente soluções infactíveis são geradas nas últimas iterações, a razão de incremento de ρ é maior para valores mais elevados de α. A variação no valor de α é, no entanto, pequena para evitar mudanças súbitas no procedimento de busca. Inicialmente, $\alpha = 1$ e $\rho = 1$. Após a estratégia de oscilação encontrar a primeira solução factível, α assume valor 2. Quando a melhor solução encontrada até então não é alterada nas últimas 100 iterações, o valor de α é aumentado em 0,005 a cada 10 iterações, ficando limitado ao seu valor máximo, 3. Se, no entanto, uma solução de melhor qualidade é encontrada, o valor de α é reinicializado em seu patamar mínimo, 2.

CAPÍTULO 10

GRASP: Busca Gulosa Aleatorizada e Adaptativa

Mauricio G. C. Resende [*] *Geraldo R. Mateus* [**] *Ricardo M. A. Silva* [***]

[*] *Algorithms and Optimization Res. Dep.*
AT&T Labs Research, USA

[**] *Departamento de Ciência da Computação*
Universidade Federal de Minas Gerais

[***] *Departamento de Ciência da Computação*
Universidade Federal de Lavras

Busca gulosa aleatorizada adaptativa[1] é uma meta-heurística multi-partida que aplica o método de busca local repetidamente a partir de soluções construídas por um algoritmo guloso aleatório. Este capítulo apresenta os componentes da meta-heurística GRASP, juntamente com hibridizações com o método de religamento de caminhos[2].

[1] GRASP, do inglês *Greedy Randomized Adaptive Search Procedure*.
[2] Do inglês *path relinking*.

1. Introdução

Um problema de otimização combinatória pode ser definido por um conjunto finito $E = \{1, \ldots, n\}$, um conjunto de soluções viáveis $F \subseteq 2^E$ e uma função objetivo $f : 2^E \to \mathbb{R}$, todos definidos para cada problema específico. Neste capítulo consideraremos a versão de minimização do problema, na qual o objetivo consiste em encontrar uma solução ótima $S^* \in F$ tal que $f(S^*) \leq f(S)$, $\forall S \in F$. Dentre os diversos problemas reais e práticos de otimização combinatória podemos citar as mais distintas aplicações, tais como roteamento, escalonamento, planejamento da produção e inventário, localização de instalações, biologia computacional, entre outras.

Embora muito progresso venha sendo alcançado na tarefa de encontrar comprovadas soluções ótimas em problemas de otimização combinatória, empregando técnicas tais como *branch and bound*, planos de corte e programação dinâmica, assim como soluções aproximadas via algoritmos aproximativos, entretanto muitos problemas de otimização combinatória que ocorrem na prática se beneficiam de métodos heurísticos que rapidamente produzem soluções de boa qualidade sem necessariamente garantir a otimalidade. Neste sentido, muitas heurísticas modernas para otimização combinatória são baseadas nos *guidelines* de meta-heurísticas, tais como: algoritmos genéticos, recozimento simulado (*simulated annealing*), busca tabu (*tabu search*), pesquisa em vizinhança variável (*variable neighborhood search*), *scatter search*, religamento de caminhos (*path relinking*), pesquisa local iterativa (*iterated local search*), colônia de formigas (*ant colony optimization*), otimização por enxame de partículas (*particle swarm optimization*) e procedimento de busca guloso aleatorizado e adaptativo (GRASP).

Este capítulo tem por objetivo apresentar os componentes estruturais básicos do GRASP, incluindo diversos esquemas construtivos, métodos de busca local e hibridizações com religamento de caminhos. Os esquemas de busca local são apresentados na Seção 2. Em seguida, a meta-heurística GRASP é revista na Seção 3. Na Seção 4, oito variantes do método construtivo guloso aleatório são descritas. Por fim, as hibridizações de GRASP com religamento de caminhos são explanadas na Seção 5, seguidas pelas conclusões na Seção 6.

2. Esquemas de Busca Local

Considere o grafo $\mathcal{X}(S, M)$ correspondente ao *espaço de soluções vizinhas*, onde o conjunto de vértices S representa todas as soluções viáveis de um problema de otimização combinatória e o conjunto de arestas M, os *movimentos* conectando soluções vizinhas. Uma solução $s \in S$ pertence à vizinhança $N(t)$ de uma solução $t \in S$ se s pode ser obtida a partir de t através de pequenas alterações pré-definidas empregadas em t. Se a solução $s \in N(t)$, então $t \in N(s)$, $(s, t) \in M$ e $(t, s) \in M$. Por último, diferentes estruturas de vizinhança podem ser definidas para cada problema de otimização combinatória.

Os esquemas de busca local partem de uma solução inicial, por sua vez tida como sendo a solução corrente, a procura de uma solução melhor na vizinhança da solução corrente. Se esta solução existir, automaticamente tornar-se-á a nova solução corrente e a busca local recursivamente será aplicada sobre esta solução. O procedimento termina quando nenhuma solução melhor do que a corrente existir na vizinhança desta última. Tal esquema de busca local exige que o tamanho da vizinhança seja tal que sua exploração possa ser feita eficientemente.

O Algoritmo 1 mostra o pseudo-código para o algoritmo de busca local padrão. A busca local começa em algum vértice $s_0 \in S$ tornando-o a solução atual t, ou seja, $t = s_0$. A cada iteração, o algoritmo procura uma solução vizinha que possua um valor da função objetivo melhor do que o valor correspondente à solução corrente, ou seja, uma solução $s \in N(t)$ tal que $f(s) < f(t)$. Se tal solução existir, tornar-se-á a solução corrente, ou seja, $t = s$. Estas iterações são repetidas até não existir nenhuma solução melhor do que a solução corrente t na vizinhança $N(t)$. Neste caso, a solução t é

denominada como um *ótimo local*.

A busca local pode ser vista como partindo de um vértice $s \in S$ e examinado os nós adjacentes no grafo $\mathcal{X}$ em busca de uma solução de melhoria. Na variante *first-improving*, a busca local se move para a primeira solução de melhoria encontrada na vizinhança, enquanto na variante *best-improving* todas as soluções da vizinhança são avaliadas de modo que a busca local se direcione para a melhor solução encontrada. Podemos considerar esta fase da busca como sendo a de *intensificação*, uma vez que apenas uma pequena parte do espaço de soluções está sendo considerada.

Vários enfoques foram propostos para estender a busca local acima citada, tais como descida em vizinhança variável (*variable neighborhood descent*) (Andrade e Resende, 2006; Hansen e Mladenović, 1999; Martins et al., 1999; Ribeiro e Souza, 2002; Ribeiro et al., 2002; Ribeiro e Vianna, 2005), busca em vizinhança variável (*variable neighborhood search*) (Beltrán et al., 2004; Canuto et al., 2001; Drummond et al., 2002; Festa et al., 2002; Hansen e Mladenović, 1999; Ochi et al., 2001), busca tabu de memória curta (Abdinnour-Helm e Hadley, 2000; de Souza et al., 2003; Delmaire et al., 1999; Glover, 1989, 1990; Laguna e González-Velarde, 1991; Li, Guo, Wang e Lim, 2004; Lim e Wang, 2004; Moura e Oliveira, 2005; Pu et al., 2006; Serra e Colomé, 2001; Souza et al., 2001), recozimento simulado (de la Penã, 2004; Kirkpatrick et al., 1983; Liu et al., 2000), pesquisa local iterativa (Baum, 1986a,b; Baxter, 1981; Johnson, 1990; Martin e Otto, 1996; Martin et al., 1991a; Ribeiro e Urrutia, 2007), e busca em vizinhança de muito larga escala(*very-large scale neighborhood search*) (Ahuja et al., 2001, 2002; Geng et al., 2005), os quais admitem, além da exploração da vizinhança da solução corrente, desde movimentos para soluções de custo maior do que a solução corrente, passando pelo cenário de múltiplas vizinhanças, até a exploração de vizinhanças bem maiores.

Algoritmo 1 Busca local padrão

1: Entrada: $s_0 \in S$
2: Resultado: Ótimo local
3: $t \leftarrow s_0$
4: **enquanto** existe $s \in N(t)$ tal que $f(s) < f(t)$ **faça**
5: $\quad t \leftarrow s$
6: **fim enquanto**
7: **retornar** t

3. Procedimentos de Busca Gulosos Aleatorizados Adaptativos

Um método de busca eficaz precisa viabilizar sua diversificação. Portanto, uma boa estratégia para a busca local não deve se ater a uma região particular do espaço de soluções, como por exemplo, em torno de uma solução construída por um algoritmo guloso. Uma alternativa consiste em iniciar uma busca local a partir de várias soluções geradas aleatoriamente, com a expectativa de que haja um caminho de melhoria de custo partindo de uma destas soluções até uma solução ótima ou próxima do ótimo.

Um procedimento de busca guloso, aleatório e adaptativo repetidamente aplica a busca local a partir de soluções construídas com um algoritmo guloso aleatório. O melhor ótimo local dentre todas as buscas locais é retornado como solução da heurística. O Algoritmo 2 apresenta o pseudo-código de um GRASP genérico. Ao fim da fase de construção gulosa aleatória, a solução obtida é viável para diversas instâncias da heurística. Entretanto, para alguns problemas, a solução pode ser inviável, logo requerendo a aplicação de um procedimento de reparo que restaure sua viabilidade. Exemplos de implementações GRASP munidas de procedimentos de reparos para restaurar a viabilidade de soluções podem ser encontrados em Duarte, Ribeiro e Urrutia (Duarte et al., 2007), Duarte et. al (Duarte et al., 2006), Nascimento, Resende e Toledo (Nascimento et al., 2010) e Mateus, Resende e

Silva (Mateus et al., 2009).

Uma característica especialmente atraente da meta-heurística GRASP é a facilidade com que pode ser implementada. Dado que poucos parâmetros precisam ser inicializados e ajustados, logo o desenvolvimento pode se ater à implementação dos algoritmos e estruturas de dados que garantam a eficiência do projeto. Como veremos a seguir, as implementações básicas do GRASP repousam exclusivamente em dois parâmetros. O primeiro controla o número de iterações dos métodos construtivo e de busca local a serem aplicados. O segundo controla a mistura dos enfoques aleatório e guloso do método construtivo. A despeito de sua simplicidade e facilidade de implementação, GRASP é uma meta-heurística muito eficaz e produz as melhores soluções conhecidas para muito problemas.

GRASP foi introduzido pela primeira vez por Feo e Resende (Feo e Resende, 1989). Ver Feo e Resende (Feo e Resende, 1995), Pitsoulis e Resende (Pitsoulis e Resende, 2002), Resende e Ribeiro (Resende e Ribeiro, 2003) e Resende (Resende, 2008) para um revisão da meta-heurística GRASP e Festa e Resende (Festa e Resende, 2002, 2009a,b) para uma bibliografia anotada do assunto.

Algoritmo 2 Pseudo-código de um GRASP básico.

```
1: Entrada: Critério de parada
2: Resultado: Solução x*.
3: x* ← ∞
4: enquanto critério de parada não for satisfeito faça
5:    x ← GulosoAleatorio(·)
6:    se x não for viável então
7:       x ← reparo(x)
8:    fim se
9:    x ← BuscaLocal(x)
10:   se f(x) < f(x*) então
11:      x* ← x
12:   fim se
13: fim enquanto
14: retornar x*
```

4. Método de Construção Guloso Aleatório

Os métodos de construção gulosos aleatórios do GRASP visam produzir um conjunto diversificado de soluções iniciais de boa qualidade para a busca local. Este propósito é alcançado pela adição de aleatoriedade ao algoritmo guloso. São ilustrados aqui oito formas de realizar tal tarefa, nas quais as soluções são construídas pela adição de elementos a uma solução parcial.

Construção semi-gulosa

O Algoritmo 3 mostra o pseudo-código para este método de construção. A cada iteração do primeiro esquema de construção, denominado *algoritmo semi-guloso* por Hart e Shogan (Hart e Shogan, 1987), seja C o conjunto de todos os elementos que podem ser adicionados à solução parcial e seja $LRC(C)$ a lista restrita de candidatos composta por elementos candidatos de alta-qualidade. A qualidade de um elemento candidato é determinada por sua contribuição, naquele ponto, para o custo da solução sendo construída. A função gulosa $g(c)$ mede esta contribuição para cada candidato $c \in C$. Membros do conjunto $LRC(C)$ podem ser determinados pelo *rank* ou pela qualidade relativa aos outros candidatos. Membros segundo o *rank*, também denominados *baseados na cardinalidade*, são obtidos se o candidato é um dos q candidatos com o menor valor da função gulosa, onde q é um parâmetro de entrada que

determina quão gulosa ou aleatória será a construção. Membros *baseados na qualidade* relativa a outros candidatos determinam um valor de corte para a função gulosa e somente consideram candidatos com uma valor guloso que não seja maior do que este valor de corte. Neste caso, a implementação utiliza um parâmetro real $\alpha \in [0,1]$. Seja $\acute{g} = \max\{g(c) \mid c \in C\}$ e $\grave{g} = \min\{g(c) \mid c \in C\}$. Um candidato $c \in C$ é posto na LRC se e somente se $\grave{g} \leq g(c) \leq \grave{g} + \alpha \cdot (\acute{g} - \grave{g})$. O parâmetro de entrada α determina quão gulosa ou aleatória será a construção. Um dentre os candidatos em $LRC(C)$ é selecionado aleatoriamente e inserido na solução parcial. A construção é repetida até não haver mais candidatos.

Algoritmo 3 Método construtivo da meta-heurística GRASP: aleatorização do algoritmo guloso ou semi-guloso.

1: Entrada: $E = \{$ conjunto discreto finito $\}$.
2: Resultado: Solução S.
3: $S \leftarrow \emptyset$; $C \leftarrow E$
4: **enquanto** $|C| > 0$ **faça**
5: Para todo $c \in C$ computar o valor da função gulosa $g(c)$
6: Definir $LRC(C) \leftarrow \{c \in C \mid g(c)$ tam um valor reduzido$\}$
7: Selecionar aleatoriamente $c^* \in LRC(C)$
8: Adicionar c^* à solução parcial: $S \leftarrow S \cup \{c^*\}$
9: Seja C o conjunto de elementos que podem ser adicionados à S
10: **fim enquanto**
11: **retornar** S

Construção por amostragem gulosa

Neste segundo esquema de construção, denominado *construção por amostragem gulosa* por Resende e Werneck (Resende e Werneck, 2004), um algoritmo guloso é aplicado a cada amostragem aleatória de candidatos, ao invés da aleatorização do algoritmo guloso. O Algoritmo 4 mostra o pseudo-código para este procedimento construtivo. A cada passo um subconjunto de tamanho fixo de candidatos em C é amostrada e a contribuição incremental ao custo da solução parcial é computada para cada elemento amostrado. Um elemento com a melhor contribuição incremental é selecionado e adicionado à solução parcial. Esse processo é repetido até quando não existir nenhum candidato.

Algoritmo 4 Construção gulosa aleatória da meta-heurística GRASP: amostragem gulosa

1: Entrada: $p, E = \{$ conjunto discreto finito $\}$.
2: Resultado: Solução S.
3: $S \leftarrow \emptyset$; $C \leftarrow E$
4: **enquanto** $|C| > 0$ **faça**
5: Amostrar aleatoriamente $\min\{p, |C|\}$ elementos
6: a partir de C e inserí-los em $LRC(C)$
7: Selecionar $c^* = \text{argmin}\{g(c) \mid c \in LRC(C)\}$
8: Adicionar c^* à solução parcial: $S \leftarrow S \cup \{c^*\}$
9: Seja C o conjunto de elementos que podem ser adicionados à S
10: **fim enquanto**
11: **retornar** S;

Construção aleatória-gulosa

Uma possível restrição da construção gulosa aleatória baseada em uma lista restrita de candidatos é sua complexidade. A cada passo do método construtivo, cada elemento candidato ainda não selecionado tem que ser avaliado pela função gulosa. Nos casos onde a diferença entre o número de elementos do conjunto E e o número de elementos que compõem uma solução do problema é alto, o método construtivo pode não ser muito eficiente. Como ocorre no caso do esquema de construção por amostragem gulosa, este terceiro esquema introduzido por Resende e Werneck (Resende e Werneck, 2004), denominado *construção aleatório guloso*, também atende esta restrição.

Neste esquema, uma parte da solução é construída pela escolha aleatória de p elementos candidatos e o restante da solução é completado de forma gulosa. A solução resultante é gulosa e aleatória. O valor de p determina quão guloso ou aleatório é o método construtivo. Valores reduzidos de p levam à soluções mais gulosas, enquanto valores maiores levam à soluções mais aleatórios. A amostragem aleatória foi apresentada na Subseção 4.

Construção gulosa proporcional

A cada iteração deste quarto esquema de construção, denominado *guloso proporcional* e introduzido pela primeira vez por Resende e Werneck (Resende e Werneck, 2004), a função gulosa $g(c)$ para todo elemento candidato $c \in C$ é computada e em seguida um candidato é selecionado aleatoriamente, porém de um modo viciado: a probabilidade de um dado candidato $c' \in C$ ser selecionado é inversamente proporcional a $g(c') - \min\{g(c)|c \in C\}$.

Construção GRASP reativa

O quinto esquema construtivo é denominado *GRASP reativo*. Nos procedimentos de construção guloso aleatório, o projetista deve decidir como balancear os enfoques guloso e aleatório. Uma simples alternativa seria balanceá-los aleatoriamente. Por exemplo, na construção semi-gulosa, com membros em LRC segundo a qualidade relativa aos outros candidatos, o parâmetro α pode ser selecionado de modo uniformemente aleatório a partir do intervalo $[0, 1]$, de modo que cada iteração GRASP use um valor diferente de α, resultando em um balanceamento diferente entre os enfoques guloso e aleatório.

Prais e Ribeiro (Prais e Ribeiro, 2000) mostraram que usando um valor fixo para o parâmetro α no esquema semi-guloso baseado na qualidade frequentemente retarda o encontro de soluções de alta qualidade, as quais eventualmente seriam encontradas caso outro valor para α fosse utilizado. Sendo assim, eles propuseram uma extensão do procedimento GRASP básico, denominado *GRASP reativo*, na qual o parâmetro α é selecionado aleatoriamente a cada iteração a partir de um conjunto discreto de possíveis valores. Cabe ressaltar que valores das soluções encontrados ao longo das iterações anteriores servem de guia para o processo de seleção.

Prais e Ribeiro (Prais e Ribeiro, 2000) definem $\Psi = \{\alpha_1, \ldots, \alpha_m\}$ como o conjunto de valores possíveis para α. As probabilidades associadas com a escolha de cada valor são inicialmente iguais a $p_i = 1/m,\ i = 1, \ldots, m$. Além disso, seja z^* a solução titular (a melhor solução encontrada até aquele ponto) e A_i o valor médio de todas soluções encontradas usando $\alpha = \alpha_i,\ i = 1, \ldots, m$. As probabilidades de seleção são periodicamente reavaliadas tomando $p_i = q_i / \sum_{j=1}^{m} q_j$, com $q_i = z^*/A_i$ para $i = 1, \ldots, m$. O valor de q_i será maior para os valores de $\alpha = \alpha_i$ associados às melhores soluções em média. Logo, valores maiores de q_i correspondem a valores mais adequados para o parâmetro α. As probabilidades mais apropriadas a estes valores irão então aumentar quando forem reavaliadas. Cabe observar, que esta estratégia reativa não é limitada aos procedimentos semi-gulosos com elementos em LRC baseados na qualidade relativa. Ou seja, ela pode ser estendida a outros esquemas de construção gulosos aleatório, todos os quais necessitam balancear o aspecto guloso com o aspecto aleatório.

Construção com memória de longo prazo

O sexto esquema de construção, denominado *construção com memória de longo prazo*, introduz uma estrutura de memória de longo prazo. Embora o GRASP reativo também utilize memória de longo prazo (informações coletadas em iterações anteriores) para ajustar o balanceamento entre os aspectos gulosos e aleatórios, Fleurent e Glover (Fleurent e Glover, 1999) observaram que isto não acontecia no GRASP básico. Sendo assim, eles propuseram um esquema de longo-prazo para atender esta questão em heurísticas *multi-partida*. Memória de longo prazo é um dos fundamentos nos quais se apoia por exemplo a busca tabu (Glover, 1989, 1990). A idéia básica consiste em manter um conjunto de soluções elite a ser usado na fase de construção. Mais detalhes a respeito deste enfoque serão fornecidos no decorrer deste capítulo Por enquanto, tudo o que precisamos saber consiste em como uma solução tornar-se uma solução elite: esta deve ser melhor do que o melhor membro do conjunto elite ou melhor do que seu pior membro, contanto que seja suficientemente diferente das outras soluções do conjunto.

Fleurent e Glover (Fleurent e Glover, 1999) definem uma *variável fortemente determinada* como sendo aquela que não pode ser alterada sem comprometer o objetivo ou mudar significantemente outras variáveis; e uma *variável consistente* como a que recebe o mesmo valor em grande parte do conjunto elite. Seja $I(e)$ uma medida das características consistentes e fortemente determinadas do elemento e da solução a partir do conjunto E. Como $I(e)$ torna-se maior na medida em que e aparece mais frequentemente no conjunto de soluções elite, ele é usado na fase de construção da seguinte maneira. Seja $g(e)$ o valor da função gulosa para o candidato $e \in C$, ou seja o custo incremental associado com a incorporação do elemento $e \in C$ na solução em construção. Seja $K(e) = F(g(e), I(e))$ uma função associada aos aspectos gulosos e de intensificação, como por exemplo, $K(e) = \lambda g(e) + I(e)$. O esquema de intensificação que favoresse a seleção dos elementos $e \in C$ com um alto valor para $K(e)$ devem atualizar a probabilidade de seleção para o seguinte valor: $p(e) = K(e)/\sum_{s \in \mathrm{LRC}} K(s)$. Cabe observar que a função $K(e)$ pode variar com o tempo pela mudança dos valores λ, por exemplo, inicialmente λ pode ser inicializado para um valor alto que decresce a medida que a diversifição é chamada.

Construção por amostragem viciada

O sétimo esquema de construção foi introduzido por Bresina (Bresina, 1996) e denominado de *construção por amostragem viciada*. Consiste em outra forma de partir de uma seleção uniforme de elementos candidatos na construção de uma solução gulosa aleatória. Ao invés de escolher de modo uniformemente aleatório o próximo elemento candidato a ser adicionado à solução parcial, a construção por amostragem viciada sugere que qualquer distribuição de probabilidade possa ser usada para favorecer alguns tipos de candidatos em particular.

No mecanismo de construção proposto por Bresina (Bresina, 1996), uma família de distribuições de probabilidade é introduzida. Elas são baseadas na colocação (*rank*) $r[\sigma]$ atribuída a cada elemento candidato σ, de acordo com os valores de suas funções gulosas. O elemento com o menor valor da função gulosa tem *rank* 1, o segundo menor tem *rank* 2, e assim por diante. Várias funções viciadas $b(\cdot)$ foram introduzidas por Bresina, tais como, *vício aleatório* onde $b(r) = 1$, *vício linear* onde $b(r) = 1/r$, *vício logarítmico* onde $b(r) = \log^{-1}(r+1)$, *vício exponencial* onde $b(r) = e^{-r}$ e *vício polinomial* de ordem n onde $b(r) = r^{-n}$. Após todos os elementos em LRC receberem seus respectivos *ranks*, a probabilidade $\pi(\sigma)$ para selecionar o elemento $\sigma \in LRC$ pode ser computada da seguinte forma: $\pi(\sigma) = b(r[\sigma])/(\sum_{\sigma' \in LRC} b(r[\sigma']))$. Cabe observar que de acordo com este esquema, candidatos podem ser selecionados a partir do conjunto LRC ou a partir do conjunto completo de candidatos, ou seja, $LRC = C$.

Construção com pertubação de custos

No oitavo esquema de construção, denominado *construção com pertubação de custos*, pertubações aleatórias são introduzidas no problema original como uma forma de obter aleatoriedade. A idéia de introduzir ruído no custo original com o intuito de aleatorizar o processo construtivo é similar ao *noising method* de Charon e Hudry (Charon e Hudry, 1993, 2002).

Em circunstâncias onde os algoritmos construtivos são insensíveis às estratégias aleatórias padrão, tal como selecionar um elemento aleatoriamente a partir de uma lista restrita de candidatos, a construção com pertubação de custo pode ser mais eficaz do que a estratégia gulosa aleatória do GRASP básico. Ribeiro, Uchoa e Werneck (Ribeiro et al., 2002) mostram que este caso ocorre na heurística de caminho mínimo de Takahashi e Matsuyama (Takahashi e Matsuyama, 1980), usado como um dos principais componentes da fase construtiva de um GRASP híbrido proposto para um problema de Steiner em grafos.

Outra situação onde esta estratégia de pertubação dos custos pode ser eficiente ocorre quando não há um algoritmo guloso a ser aleatorizado, como foi o caso do GRASP híbrido desenvolvido por Canuto, Resende, e Ribeiro (Canuto et al., 2001) para o problema de Steiner com prêmios *prize-collecting Steiner tree problem*, o qual fez uso de um algoritmo de aproximação primal-dual de Goemans e Williamson (Goemans e Williamson, 1996) para construir soluções iniciais usando custos pertubados.

5. Hibridizações com Religamento de Caminhos

A heurítica de religamento de caminhos foi originalmente proposta por Glover (Glover, 1996) como uma estratégia de intensificação para explorar trajetórias conectando soluções elite obtidas pela busca tabu ou *scatter search* (Glover, 2000; Glover e Laguna, 1997; Glover et al., 2000). Partindo de uma ou mais soluções elite na busca por melhores soluções, caminhos que levam a outras soluções elite são gerados e explorados no grafo do espaço de soluções. Para gerar estes caminhos, movimentos são selecionados para introduzir atributos presentes na solução destino na solução corrente. Desta forma, o religamento de caminhos pode ser visto como uma estratégia que procura incorporar atributos de soluções de alta qualidade.

O pseudo-código do Algoritmo 5 ilustra o religamento de caminhos misto aplicado ao par de soluções x_s e x_t. O religamento de caminhos misto (Ribeiro e Rosseti, 2002) alterna os papéis de solução origem e destino a cada movimento.

O religamento de caminhos misto começa computando as diferenças simétricas $\Delta(x_s, x_t)$ entre a solução origem x_s e a destino x_t. Ou seja, o conjunto de movimentos necessários para alcançar x_t a partir de x_s, e vice-versa. Portanto, dois caminhos de soluções são gerados, um partindo de x_s e outro de x_t. Estes caminhos crescem até se encontrarem para formar um único caminho entre x_t e x_s. A busca local é aplicada na melhor solução x^* neste caminho e o mínimo local é retornado pelo algoritmo. Inicialmente, x e y são inicializados como x_s e x_t, respectivamente. A cada passo, o procedimento examina todos os movimentos $m \in \Delta(x, y)$ partindo da solução corrente x em direção às soluções que contêm um atributo adicional de y e seleciona aquele que minimiza $f(x \oplus m)$, com $x \oplus m$ sendo a solução resultante da aplicação do movimento m sobre a solução x. Em seguida, o melhor movimento m^* é realizado produzindo a solução $x \oplus m^*$. Depois, se necessário, a melhor solução x^* é atualizada. Por fim, os conjuntos de movimentos disponíveis são atualizados e os papéis de x e y invertidos. O procedimento termina quando $|\Delta(x, y)| = 1$.

Religamento de caminhos consiste em um grande avanço no procedimento GRASP padrão, levando a melhorias significantes de desempenho e qualidade das soluções. A hibridização entre religamento de caminhos e GRASP foi primeiro proposta por Laguna e Martí (Laguna e Martí, 1999) e posteriormente seguida por várias extensões, melhorias e aplicações de sucesso (Aiex et al., 2005; Canuto et al., 2001; Festa et al., 2006; Oliveira et al., 2004; Resende et al., 2010; Resende e Ribeiro, 2003; Resende e

Algoritmo 5 Religamento de caminhos misto entre as soluções x_s e x_t.

Entrada: $x_s \in S$, $x_t \in S$.
Resultado: Solução x^*.
Calcule as diferenças simétricas $\Delta(x_s, x_t)$ e $\Delta(x_t, x_s)$
$f^* \leftarrow \min\{f(x_s), f(x_t)\}$
$x^* \leftarrow \text{argmin}\{f(x_s), f(x_t)\}$
$x \leftarrow x_s$; $y \leftarrow x_t$
enquanto $|\Delta(x, y)| > 1$ **faça**
  $m^* \leftarrow \text{argmin}\{f(x \oplus m) : m \in \Delta(x, y)\}$
  $x \leftarrow x \oplus m^*$
  Atualize $\Delta(x, y)$ e $\Delta(y, x)$
  se $f(x) < f^*$ **então**
    $f^* \leftarrow f(x)$
    $x^* \leftarrow x$
  fim se
  $t \leftarrow y$; $y \leftarrow x$; $x \leftarrow t$
fim enquanto
$x^* \leftarrow$ `BuscaLocal`(x^*)
retornar x^*;

Algoritmo 6 Uma heurística GRASP-PR básica para minimização.

Entrada: $i_{\max}$.
Resultado: Solução x^*.
$P \leftarrow \emptyset$
$f^* \leftarrow \infty$
para $i = 1, \ldots, i_{\max}$ **faça**
  $x \leftarrow$ `ConstrucaoGulosaAleatoria`()
  se x não for viável **então**
    $x \leftarrow$ `reparo`(x)
  fim se
  $x \leftarrow$ `BuscaLocal`(x)
  se $i \geq 2$ **então**
    Escolha aleatoriamente as soluções elite $\mathcal{Y} \subseteq P$ para religar com x
    para $y \in \mathcal{Y}$ **faça**
      $x_p \leftarrow$ `ReligamentoDeCaminhos`(x, y)
      Atualize o conjunto elite P com x_p
      se $f(x_p) < f^*$ **então**
        $f^* \leftarrow f(x_p)$
        $x^* \leftarrow x_p$
      fim se
    fim para
  fim se
fim para
$P \leftarrow$ `PosOtimizacao`$\{P\}$
$x^* \leftarrow \text{argmin}\{f(x), x \in P\}$
retornar x^*

Werneck, 2004, 2006; Ribeiro et al., 2002). Um resumo de GRASP com religamento de caminhos é

apresentado em Resende e Ribeiro (Resende e Ribeiro, 2005a). Duas estratégias básicas são usadas. Na primeira, o religamento de caminhos é aplicado a todos os pares de soluções elite, seja periodicamente durante as iterações GRASP ou como uma estapa de pós-otimização, após todas as iterações GRASP terem sido realizadas. Na segunda, o religamento de caminhos é aplicado como uma estratégia de intensificação a cada ótimo local obtido após a busca local.

Aplicar religamento de caminhos como uma estratégia de intensificação sobre cada ótimo local geralmente é mais eficaz do que simplesmente usá-lo como uma etapa de pós-otimização. Em geral, combinar intensificação com pós-otimização resulta em uma melhor estratégia. No contexto de intensificação, religamento de caminhos é aplicado em pares (x, y) de soluções, onde x é uma solução ótima local produzida a cada iteração GRASP, após a aplicação da busca local, e y é uma das soluções elite aleatoriamente escolhidas a partir de um conjunto elite com um número máximo `Max_Elite` de elementos obtidos no decorrer da busca. A seleção aleatória uniforme é uma simples estratégia a ser implementada. Desde que a diferença simétrica é uma medida do tamanho do caminho a ser explorado durante o religamento, uma estratégia viciada que privilegia as soluções y do conjunto elite com alta diferença simétrica em relação a x é usualmente melhor do que aquela usando a seleção uniformemente aleatória (Resende e Werneck, 2004).

O conjunto elite inicialmente é vazio. Desde que nós desejamos manter este conjunto com diversificadas soluções de alta qualidade, cada ótimo local obtido pela busca local é considerado como um candidato a ser inserido no conjunto elite, caso seja suficientemente diferente de todas as outras soluções atualmente contidas no mesmo. Se o conjunto já possui `Max_Elite` soluções e o candidato é melhor do que o pior entre eles, então uma simples estratégia consiste em substituir a segunda pela primeira. Outra estratégia, a qual tende a aumentar a diversidade do conjunto elite, consiste em substituir o elemento mais similar à solução candidata dentre todos os elementos do conjunto elite com custo pior do que o custo da solução candidata. Caso o conjunto elite não esteja cheio, o candidato é simplesmente adicionado ao mesmo.

A pós-otimização é realizada sobre uma série de conjuntos. O conjunto inicial P_0 consiste no conjunto P obtido ao final das iterações GRASP. O valor da melhor solução de P_0 é atribuída a f_0^* e um contador de iterações k é inicializado para 0. Na k-ésima iteração, todos os pares de elementos do conjunto P_k são combinados através do religamento de caminhos. Cada resultado do religamento de caminhos é testado para inclusão no conjunto P_{k+1} seguindo os mesmos critérios usados nas iterações GRASP. Se uma nova melhor solução é produzida, ou seja, $f_{k+1}^* < f_k^*$, então $k \leftarrow k + 1$ e uma nova iteração da pós-otimização é realizada. Caso contrário, a pós-otimização é encerrada com $x^* \in \text{argmin}\{f(x) \mid x \in P_{k+1}\}$ como resultado.

O pseudo-código do Algoritmo 6 ilustra este procedimento. Agora, cada iteração GRASP possui três passos principais. Na fase de construção, um procedimento de busca gulosa e aleatória é usado para construir uma solução viável. Na fase de busca local a solução construída na primeira fase é progressivamente melhorada por uma estratégia de busca na vizinhança, até que um mínimo local seja encontrado. Na fase de religamento de caminhos o algoritmo de religamento de caminhos é aplicado sobre a solução obtida pela busca local e uma solução aleatoriamente selecionada a partir do conjunto elite. A melhor solução encontrada ao longo desta trajetória é também considerada como uma candidata a inserção no conjunto elite. Ao final das iterações GRASP, a fase de pós-otimização combina as soluções do conjunto elite na busca por melhores soluções. Mateus, Resende e Silva (Mateus et al., 2009) propõem uma nova versão de GRASP com religamento de caminhos, motivado pelo fato de que em alguns problemas um movimento guiado pela solução destino não garante a viabilidade da nova solução construída.

6. Conclusões

Este capítulo considerou os componentes estruturais básicos necessários para projetar heurísticas baseadas nos princípios que regem a meta-heurística GRASP. Estes componentes incluem esquemas de construção aleatórios, procedimentos de busca local e a introdução de estruturas de memória através do religamento de caminhos. Um tópico importante não abordado neste capítulo é o GRASP paralelo. O leitor interessado é direcionado ao artigo de Resende e Ribeiro (Resende e Ribeiro, 2005b).

(Página deixada propositadamente em branco)

CAPÍTULO 11

Algoritmos de Estimação de Distribuição

Pablo A. D. de Castro [*] *Fernando J. Von Zuben* [**]

[*] *Instituto Federal de Educação, Ciência e Tecnologia de São Paulo*
São Carlos - SP

[**] *Departamento de Engenharia de Computação e Automação Industrial*
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

Nos últimos anos, tem havido um aumento no interesse pelo desenvolvimento e aplicação de um novo paradigma de computação evolutiva, denominado Algoritmo de Estimação de Distribuição. No lugar de usar os operadores tradicionais de mutação e recombinação de código genético, essa classe de algoritmos explora o espaço de busca construindo, a cada iteração, um modelo probabilístico que representa a distribuição de probabilidade das melhores soluções. Em seguida, novas soluções candidatas são amostradas a partir do modelo probabilístico obtido. A estimativa da distribuição de probabilidade das melhores soluções permite capturar correlações entre variáveis do problema por meio das regularidades encontradas nessas soluções. Consequentemente, o algoritmo é capaz de manipular eficientemente blocos construtivos, que correspondem a soluções parciais de boa qualidade contidas em trechos do vetor de soluções, evitando seu rompimento. Se, ao longo das gerações de um algoritmo evolutivo, é possível identificar e preservar blocos construtivos, o número de soluções candidatas que precisam ser avaliadas até se atingir um certo nível de desempenho tende a ser bem menor que o número

de avaliações requerida por algoritmos evolutivos tradicionais, para obter o mesmo desempenho. No entanto, o custo computacional por geração cresce quando se emprega um algoritmo de estimação de distribuição. Este capítulo descreve as principais propostas existentes na literatura e formaliza os modelos gráficos probabilísticos geralmente adotados junto a problemas de otimização de interesse prático, tanto em espaços contínuos quanto discretos.

1. Introdução

Algoritmos genéticos são meta-heurísticas baseadas em população, inspirados na teoria da seleção natural e nos princípios da genética. Apesar de terem sido aplicados com sucesso na resolução de problemas de busca e otimização nas mais diversas áreas, existem duas principais limitações associadas a estes algoritmos que podem produzir um impacto negativo em problemas de otimização mais complexos: (*i*) a definição adequada dos operadores de mutação e recombinação de código genético para cada tipo de problema, bem como os valores para seus parâmetros; e (*ii*) a ausência de um mecanismo capaz de extrair e utilizar conhecimento sobre as regularidades do problema por meio de soluções promissoras. Em muitos casos, as variáveis do problema se relacionam, formando subconjuntos que representam soluções parciais de boa qualidade para o problema. Essas soluções parciais são denominadas blocos construtivos (do inglês *building blocks*)(Goldberg, 1989).

Os algoritmos genéticos, na sua forma mais simples, com codificações fixas e operadores generalizados, independentes do problema, frequentemente rompem os blocos construtivos e, desse modo, tendem a tornar o processo de busca por boas soluções menos eficaz.

Várias propostas surgiram para eliminar ou amenizar estas limitações, desde a mudança na representação das soluções candidatas até a criação de operadores de mutação e recombinação mais específicos. Entretanto, esta abordagem é altamente dependente do problema. Diante deste cenário, na última década uma nova classe de algoritmos evolutivos, chamada Algoritmo de Estimação de Distribuição (AED) (do inglês *Estimation of Distribution Algorithm*), foi concebida como uma extensão dos algoritmos genéticos (Baluja, 1994; Mühlenbein e Paass, 1996). Os AEDs combinam os conceitos de computação evolutiva e aprendizado de máquina para evoluir a população de soluções. Nesses algoritmos, os operadores de mutação e recombinação são substituídos por um modelo probabilístico que representa a distribuição de probabilidade conjunta das melhores soluções. Em seguida, utiliza-se este modelo gerado para amostrar novas soluções. As escolhas do modelo mais apropriado e da forma de construí-lo variam conforme a aplicação.

A habilidade de identificar e preservar os blocos construtivos implicitamente confere a estes algoritmos um potencial diferenciado como ferramentas de otimização. A busca torna-se mais robusta, no sentido de que tende a ocorrer uma redução acentuada na variação de desempenho entre re-execuções do algoritmo para um mesmo problema. Além disso, tende a ocorrer também uma redução acentuada no número de avaliações de soluções candidatas até que se atinja um certo nível de desempenho.

Os objetivos deste capítulo são apresentar as motivações que levaram ao desenvolvimento dos AEDs, explicar detalhadamente seu princípio de funcionamento, elencar algumas aplicações práticas em que os AEDs estão sendo empregados com sucesso e apresentar perspectivas para o desenvolvimento de novos AEDs.

2. Blocos Construtivos

O algoritmo genético, assim como diversos outros algoritmos de busca e otimização, representa as soluções candidatas para um determinado problema como um vetor de atributos. É sabido que, em muitos problemas de otimização, as variáveis podem se relacionar umas com as outras, formando soluções parciais de boa qualidade para o problema. A essas soluções parciais dá-se o nome de blocos

construtivos. Os blocos construtivos podem ser vistos, então, como pequenos conjuntos de posições do vetor de soluções que se relacionam e, juntos, representam soluções parciais para o problema. Esses blocos construtivos contribuem de forma significativa para garantir a qualidade das boas soluções e devem ser sempre preservados.

Para ilustrar melhor o conceito de blocos construtivos, considere o problema artificial Trap-5 (Deb e Goldberg, 1992). Este problema consiste em dividir o vetor de soluções (vetor binário n-dimensional) em partições disjuntas de 5 bits cada. Este particionamento permanece fixo durante todo o processo de otimização e o algoritmo não possui nenhum conhecimento sobre como foi feito tal particionamento. Em seguida, uma função (Equação 11.1) é aplicada para cada grupo de 5 bits.

$$trap_5(u) = \begin{cases} 5, & \text{se } u = 5 \\ 4 - u, & \text{se } u < 5 \end{cases} \tag{11.1}$$

em que u denota a quantidade de vezes que aparece o símbolo 1 no bloco de 5 bits. Na Figura 11.1 é possível visualizar esta função para um bloco de 5 bits.

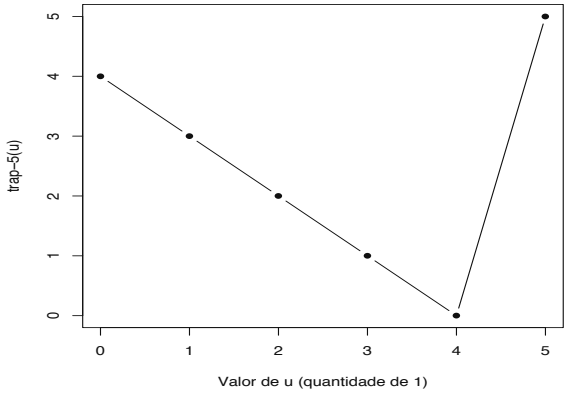


Figura 11.1: Exemplo do problema Trap-5 para um bloco de 5 bits

O objetivo é maximizar a função. Para um vetor de soluções n-dimensional, esta função possui seu ótimo global quando todos os seus bits são iguais a "1", e possui $2^{n/5} - 1$ ótimos locais. A contribuição de todas as funções são combinadas para formar o valor de adaptação ou *fitness* do indivíduo (solução candidata). Portanto, o *fitness* de uma solução candidata n-dimensional X é dado por:

$$fitness(X) = \sum_{i=1}^{n/5} trap_5(S_i) \tag{11.2}$$

em que S_i representa o i-ésimo bloco de 5 bits.

Este exemplo sugere que existem problemas em que o tratamento de variáveis de forma isolada não funciona, pois cada grupo de 5 bits contendo a sequência "11111" é um bloco construtivo. Um algoritmo genético tradicional apresenta dificuldades ao tentar resolver problemas desse tipo (Trap-k, com $k > 2$), pois os operadores de mutação e recombinação não conseguem preservar os blocos construtivos, levando-os constantemente ao rompimento. Esta situação é conhecida como problema da ligação (do inglês, *linkage problem*) (Goldberg et al., 1989).

Diversas tentativas para evitar o rompimento dessas soluções parciais foram realizadas, incluindo a mudança na representação das soluções e o desenvolvimento de operadores evolutivos específicos. Infelizmente, abordagens deste tipo são altamente dependentes do problema, levando ao desenvolvimento de algoritmos muito específicos. Além disso, elas requerem que o usuário possua um conhecimento prévio no domínio do problema. Entretanto, em muitos problemas do mundo real, este conhecimento não está disponível previamente.

Nesse sentido, uma maneira mais eficiente de lidar com os blocos construtivos é deixar que o próprio algoritmo descubra as relações existentes entre as variáveis, usando para tanto informações estatísticas extraídas a partir do conjunto das melhores soluções. Esta abordagem é denominada aprendizado da ligação (do inglês, *linkage learning*).

As melhores soluções podem ser encaradas como amostras geradas a partir de uma distribuição de probabilidade desconhecida. Em estatística, o termo distribuição de probabilidade para um conjunto de dados refere-se à probabilidade desses dados serem observados. O algoritmo poderia, então, estimar a distribuição de probabilidade das melhores soluções e, posteriormente, utilizar esta distribuição para gerar novas soluções candidatas.

Dessa maneira, um algoritmo com esse princípio de funcionamento consegue identificar e manter blocos construtivos do problema, durante sua própria execução, evitando o rompimento de parte das soluções parciais. Cabe ressaltar que nem todos os blocos construtivos são detectados e geralmente a informação disponível para se gerar um modelo de distribuição de probabilidade é limitada, conduzindo assim a modelos aproximados e possivelmente tendenciosos.

Na Figura 11.2, está ilustrado o esquema de evolução de soluções candidatas para um problema fictício, usando um modelo probabilístico em lugar dos tradicionais operadores de mutação e recombinação. Algoritmos baseados neste princípio de funcionamento para explorar o espaço de busca são chamados de Algoritmos de Estimação de Distribuição (AEDs) e serão descritos na seção seguinte.

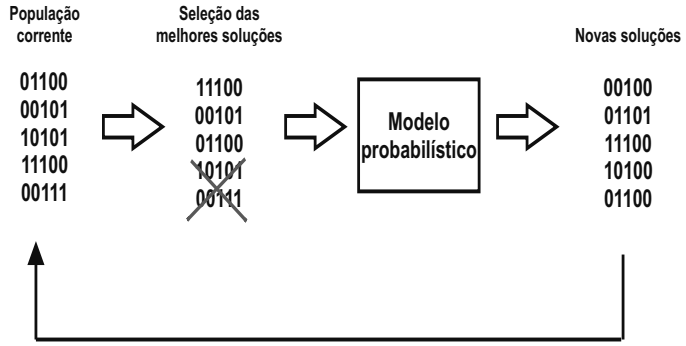


Figura 11.2: Exemplo do uso de um modelo probabilístico para gerar novas soluções candidatas.

3. Algoritmos de Estimação de Distribuição

Algoritmos de Estimação de Distribuição (AEDs) compreendem um conjunto de algoritmos evolutivos que substituem os operadores de mutação e recombinação por um modelo probabilístico que representa a distribuição de probabilidade conjunta para as melhores soluções encontradas até então. Esses algoritmos também são chamados de Algoritmos Genéticos Baseados em Modelos Probabilísticos (Baluja, 1994; Mühlenbein e Paass, 1996).

O funcionamento de um AED é similar ao de um algoritmo evolutivo. A população inicial de soluções é gerada aleatoriamente. A cada iteração, todas as soluções são avaliadas usando a função de

fitness e as melhores são selecionadas para a construção de um modelo probabilístico que represente a distribuição de probabilidade dessas melhores soluções. Após ter construído o modelo, ele é utilizado para amostrar novas soluções candidatas. Este processo é repetido até que o critério de parada seja satisfeito. A diferença, portanto, entre um Algoritmo de Estimação de Distribuição e um algoritmo evolutivo tradicional é a maneira como eles processam as melhores soluções. O pseudocódigo de um AED é apresentado no Algoritmo 1.

Algoritmo 1 Algoritmo de Estimação de Distribuição (AED)

Início
  Iniciar a população aleatoriamente;
enquanto condição de parada não for satisfeita **faça**
  Avaliar a população;
  Selecionar as melhores soluções;
  Construir o modelo probabilístico;
  Amostrar novas soluções usando o modelo probabilístico;
fim enquanto
Fim

Além da manipulação eficiente de blocos construtivos, pode-se citar mais duas motivações para se utilizar essa nova forma de evoluir a população de soluções. A primeira está relacionada com a possibilidade de analisar melhor e entender o processo de evolução do algoritmo. Já a outra reside no fato de não ser mais preciso se preocupar com a escolha dos operadores de mutação e recombinação mais adequados para cada tipo de problema, bem como os valores para seus parâmetros. É evidente que a construção do modelo probabilístico requer a determinação de alguns parâmetros, mas estes são de mais fácil definição, no sentido de que sua variação não altera drasticamente o desempenho do algoritmo.

Ao projetar um AED, surgem três questões fundamentais:

1. Que modelo probabilístico adotar para representar a distribuição de probabilidade das melhores soluções?

2. Como construir este modelo probabilístico?

3. Como usar o modelo probabilístico construído?

A resposta à primeira questão depende da complexidade do problema que se está tentando resolver e do grau de relacionamento entre as variáveis que se deseja capturar. Neste sentido, pode-se classificar os Algoritmos de Estimação de Distribuição em 3 categorias: (*i*) sem dependência entre as variáveis; (*ii*) com dependência entre pares de variáveis; e (*iii*) com dependência entre múltiplas variáveis. Ainda nesta seção, serão apresentados os algoritmos mais relevantes de cada categoria.

Para responder às outras duas questões, o projetista precisa ter em mente que é preciso existir um compromisso entre acuidade do modelo e custo computacional para gerá-lo. Quanto mais abrangente e geral for o modelo, maior será o tempo para construí-lo e também para amostrar novas soluções. Na Seção 5, serão descritas abordagens simples e eficientes para construir redes bayesianas e redes gaussianas, dois modelos gráficos probabilísticos amplamente utilizados por Algoritmos de Estimação de Distribuição e dedicados a capturar relacionamentos entre múltiplas variáveis.

Sem dependência entre as variáveis

Os algoritmos que não consideram nenhuma relação entre as variáveis, ou seja, supõem que as variáveis são independentes, são os mais simples. Devido à simplicidade do modelo probabilístico utilizado, os algoritmos nesta categoria não requerem custo computacional elevado e apresentam bom desempenho quando aplicados a problemas junto aos quais o relacionamento entre as variáveis não é tão significativo.

O primeiro algoritmo proposto nesta categoria foi o aprendizado incremental baseado em população (PBIL, do inglês *Population Based Incremental Learning*) (Baluja, 1994). Nesse algoritmo, as soluções candidatas são representadas por vetores binários, denominados cromossomos. Existe um vetor de probabilidades, da mesma dimensão do cromossomo, denotando a probabilidade de cada gene do cromossomo receber o valor "1". Inicialmente, o vetor de probabilidades contém o mesmo valor para todas as posições, 50%. A cada iteração, as melhores soluções são selecionadas e o vetor de probabilidades é atualizado seguindo uma regra baseada no aprendizado hebbiano. Esse vetor é utilizado, então, para amostrar novas soluções a cada iteração. Sebag e Ducoulombier (1998) desenvolveram uma versão desse algoritmo para lidar com problemas de otimização no espaço contínuo, denominada PBIL_c, na qual cada variável é modelada por um função densidade de probabilidade gaussiana. A cada iteração, a média e a variância de cada gaussiana são estimadas usando as melhores soluções.

Posteriormente, Mühlenbein e Paass (1996) propuseram o algoritmo de distribuição marginal univariada (UMDA, do inglês *Univariate Marginal Distribution Algorithm*). Seu funcionamento é similar ao do PBIL, com a exceção de que o vetor de probabilidades é atualizado calculando-se apenas a frequência com que aparece o número "1" nas melhores soluções selecionadas.

O algoritmo genético compacto (cGA, do inglês *Compact Genetic Algorithm*), proposto por Harik et al. (1999), também utiliza um vetor de probabilidades, o qual é iniciado contendo o valor 50% para todas as posições. Em seguida, duas soluções candidatas são amostradas a partir desse vetor e avaliadas pela função de *fitness*. O vetor de probabilidades é atualizado utilizando o valor dos genes do cromossomo com melhor *fitness*. Se os cromossomos possuírem o mesmo valor num mesmo gene, o vetor de probabilidades não é atualizado para aquela posição. Esse procedimento continua até que o vetor de probabilidades tenha convergido.

O algoritmo de estimação de distribuição com aprendizado por reforço (RELEDA, do inglês *Reinforcement Learning Estimation of Distribution Algorithm*), proposto por Paul e Iba (2003), é similar aos outros três algoritmos já apresentados. A diferença é que o vetor de probabilidades é atualizado de acordo com um método de aprendizado por reforço. Os autores compararam RELEDA com PBIL e UMDA e mostraram que seu algoritmo converge mais rapidamente para a solução ótima que os concorrentes. Os autores também desenvolveram o algoritmo de estimação de distribuição com código real (RECEDA, do inglês *Real-Coded Estimation of Distribution Algorithm*) para otimização de funções com variáveis contínuas. A partir das melhores soluções, a média e a matriz de covariância são calculadas.

O algoritmo de estimação de distribuição usando campo aleatório de Markov (DEUM, do inglês *Estimation of Distribution Algorithm Using MRF*) (Shakya et al., 2004) pode ser visto como uma adaptação dos outros quatro algoritmos. Em vez de calcular a frequência para cada gene do cromossomo, ele atualiza o vetor de probabilidades utilizando um campo aleatório de Markov (MRF, do inglês *Markov Random Field*). Os autores mostraram empiricamente que DEUM apresenta bons resultados para uma série de problemas de otimização (Shakya et al., 2005).

Dependência entre pares de variáveis

Existem algoritmos que consideram dependência entre pares de variáveis. Dessa forma, é preciso definir como representar a estrutura do modelo probabilístico, a fim de expressar o condicionamento das variáveis.

O algoritmo de maximização da informação mútua para agrupamento de entradas (MIMIC, do inglês *Mutual Information Maximization for Input Clustering*) (De Bonet et al., 1997) utiliza uma estrutura em cadeia entre as variáveis para poder representar o relacionamento entre elas, conforme ilustrado na Figura 11.3(a). Para encontrar a melhor estrutura de cadeia de variáveis, o algoritmo utiliza um mecanismo de busca que visa maximizar a informação mútua.

Baluja e Davies (1997) propuseram a combinação de otimizadores com informação mútua hierárquica (COMIT, do inglês *Combining Optimizers with Mutual Information Trees*), um algoritmo que utiliza uma estrutura em árvore para modelar a dependência entre as variáveis. Nesse tipo de estrutura, todas as variáveis precisam ter um pai, com exceção da variável-raiz. Este algoritmo é mais geral que o MIMIC, pois duas ou mais variáveis podem ser condicionadas a uma outra variável em comum. Na Figura 11.3(b) está um exemplo gráfico do modelo probabilístico utilizado pelo COMIT. Para obter a estrutura em árvore, os autores empregaram o algoritmo de árvore geradora máxima ponderada (MWST, do inglês *Maximum Weight Spanning Tree*) (Chow e Liu, 1968).

Já o algoritmo de distribuição marginal bivariada (BMDA, do inglês *Bivariate Marginal Distribution Algorithm*) (Pelikan e Mühlenbein, 1999) pode ser considerado como uma extensão do COMIT. A diferença é que o modelo em árvore utilizado por ele permite que existam mais de um nó-raiz, como mostrado na Figura 11.3(c). Para criar as sub-árvores e decidir quais variáveis devem ser conectadas ou não, o algoritmo emprega o teste do Qui-quadrado (Greenwood e Nikulin, 1996).

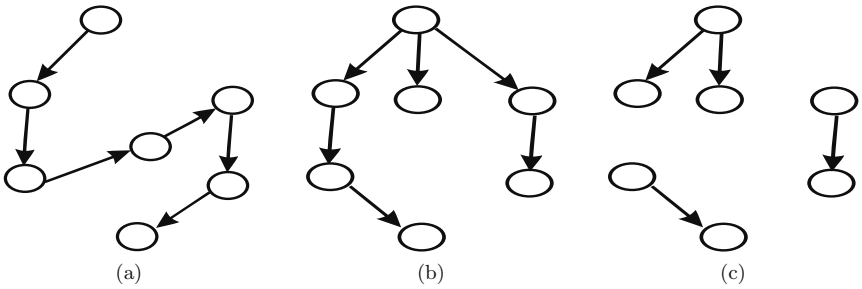


Figura 11.3: Modelos gráficos que consideram dependência entre pares de variáveis: (a) MIMIC, (b) COMIT e (c) BMDA.

Dependências entre múltiplas variáveis

Pelikan et al. (1999) mostraram que, em problemas complexos que possuem dependências entre múltiplas variáveis, os algoritmos pertencentes às duas categorias anteriores não são capazes de solucioná-los de forma satisfatória. Nesse sentido, várias propostas surgiram recentemente utilizando modelos capazes de expressar o relacionamento entre diversas variáveis. Como resultado, o modelo probabilístico é mais poderoso, embora sua obtenção se torne mais complexa.

O algoritmo genético compacto estendido (ECGA, do inglês *Extended Compact Genetic Algorithm*) (Harik et al., 2006) é uma extensão do cGA. A idéia básica é agrupar as variáveis em grupos independentes e, então, aplicar o cGA nesses grupos (veja Figura11.4(a)). O algoritmo de agrupamento primeiramente considera a existência de n grupos, sendo uma variável em cada grupo. Em seguida, tenta-se unificar pares de grupos usando medidas de entropia e a métrica Comprimento Mínimo de Descrição (MDL, do inglês *Minimum Description Length*). Problemas que podem ser decompostos em sub-problemas sem sobreposição são tratados de forma satisfatória pelo ECGA. Por outro lado, se existir sobreposição, o algoritmo falha, pois o problema não pode ser modelado de forma acurada pela simples divisão das variáveis em classes distintas.

Mühlenbein e Mahnig (1999) propuseram o algoritmo de distribuição fatorizada (FDA, do inglês

Factorized Distribution Algorithm). Este algoritmo utiliza uma estrutura em grafo definida previamente e que se mantém fixa durante todo o processo de otimização, sendo atualizados apenas os parâmetros numéricos (probabilidades). Portanto, é preciso ter conhecimento sobre o problema a ser tratado para se construir um grafo adequado. O FDA utiliza a distribuição de Boltzmann para amostrar novos indivíduos. Um exemplo de grafo utilizado pelo FDA pode ser visto na Figura 11.4(b).

O algoritmo de otimização bayesiano (BOA, do inglês *Bayesian Optimization Algorithm*) (Pelikan et al., 1999) utiliza uma rede bayesiana para modelar as dependências entre as variáveis, conforme ilustrado na Figura 11.4(c). A cada iteração, uma rede bayesiana é gerada a partir das melhores soluções, tentando refletir da melhor forma as relações entre as variáveis. Após gerada, a rede é utilizada para amostrar novas soluções candidatas. Para construir a rede bayesiana, BOA começa com um estrutura sem arcos e os vai adicionando até que não ocorra melhora na métrica utilizada para avaliar a qualidade do modelo. Usualmente, BOA emprega a métrica *Bayesian Dirichlet* (BD). O algoritmo de otimização bayesiano com código real (rBOA, do inglês *real-coded Bayesian Optimization Algorithm*) (Ahn et al., 2006) é uma extensão do BOA para otimização em espaços contínuos, empregando modelo de mistura gaussiana. Já o algoritmo de otimização bayesiano hierárquico (hBOA, do inglês *hierarchical Bayesian Optimization Algorithm*) (Pelikan, 2005a) resolve problemas de otimização decompondo-os hierarquicamente, em vez de decompô-los em um único nível.

Outro algoritmo que também utiliza redes bayesianas foi proposto posteriormente por Larrañaga et al. (2000a), sendo denominado algoritmo de estimação de redes bayesianas (EBNA, do inglês *Estimation of Bayesian Networks Algorithm*). Seu funcionamento é similar ao BOA e diversas versões do algoritmo foram criadas com diferentes métricas para avaliar os modelos. Posteriormente, os autores desenvolveram a versão do algoritmo para lidar com otimização em espaços contínuos, denominada algoritmo de estimação de redes gaussianas (EGNA, do inglês *Estimation of Gaussian Networks Algorithm*) (Larrañaga et al., 2000b). EGNA utiliza redes gaussianas para modelar a dependência entre as variáveis.

Bosman e Thierens (2000) propuseram o algoritmo evolutivo com densidade iterativa (IDEA, do inglês *Iterated Density Evolutionary Algorithm*), que é uma arquitetura geral de algoritmos para aplicação em problemas de otimização tanto no domínio discreto quanto contínuo. No caso contínuo, os algoritmos utilizam funções densidade de probabilidade gaussianas multivariadas.

Uma proposta mais recente é o algoritmo de estimação de distribuição com rede de Markov (MN-EDA, do inglês *Markov Network Estimation of Distribution Algorithm*) (Santana, 2005), que utiliza uma rede de Markov como modelo probabilístico, como mostrado na Figura 11.4(d). Essa rede é gerada usando um procedimento estatístico conhecido como aproximação de Kikuchi (Kikuchi, 1951). Novas soluções são produzidas usando amostragem de Gibbs.

Desempenho do BOA no Trap-5

Nesta seção, é apresentado o desempenho do BOA junto ao problema Trap-5, já descrito na seção 2, e comparado com um algoritmo genético (AG). Os algoritmos foram comparados em termos de número de avaliações de função-objetivo necessárias até a população convergir. A escalabilidade dos algoritmos também foi testada, variando o valor de n no Trap-5 (tamanho do vetor de soluções).

Os parâmetros do BOA foram estabelecidos da seguinte forma: o tamanho da população variou de 200 (para n=20) até 600 (para n=80). Foi imposta uma restrição no número de pais que um nó pode ter na rede bayesiana, definido como 2. Para o algoritmo genético, o tamanho da população foi definido igual ao do BOA. Foi utilizado o operador de recombinação de um ponto com taxa de 90%. A taxa de mutação foi definida como 1%. Ambos os algoritmos utilizaram seleção por truncamento com 50%, ou seja, metade da população era selecionada. Estes parâmetros foram definidos empiricamente.

Os resultados médios em 30 execuções podem ser vistos na Tabela 11.1. Todos os algoritmos encontraram o ótimo global em todos os casos. Entretanto, BOA precisou de um número menor de

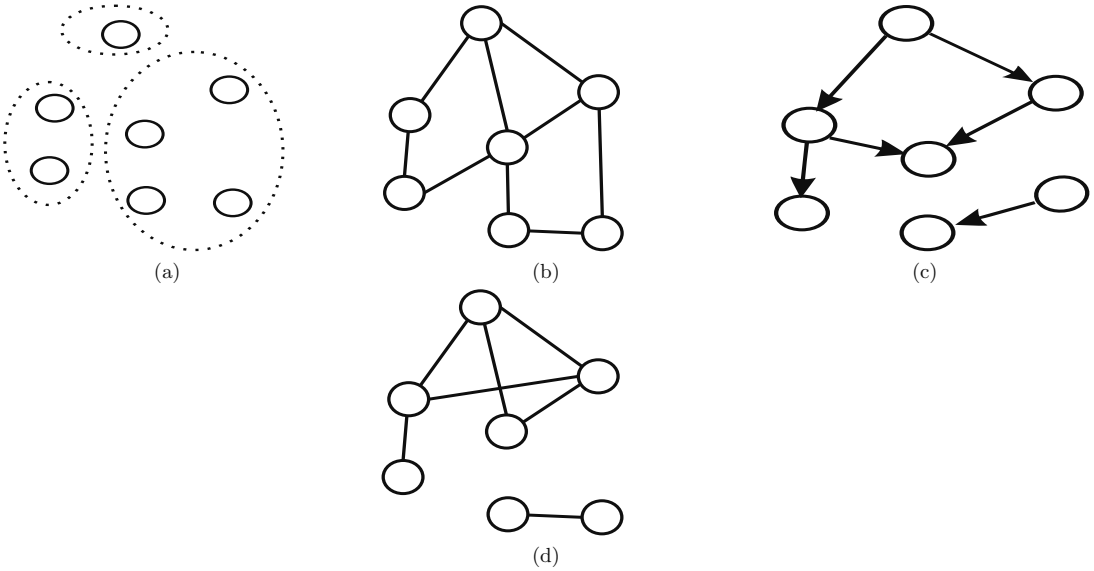


Figura 11.4: Modelos gráficos que consideram múltiplas dependências entre as variáveis: (a) ECGA, (b) FDA, (c) BOA e EBNA e (d) MN-EDA.

avaliações de função que o AG. Isto porque os operadores de mutação e recombinação do AG rompiam os blocos construtivos.

Tabela 11.1: Média dos resultados comparativos entre BOA e AG para o problema Trap-5, em termos de avaliações de *fitness*

n	BOA	AG
20	8100	14300
30	9200	22400
50	40000	70600
80	65400	114800

4. Limitações dos AEDs

Embora os algoritmos de estimação de distribuição (AEDs) possuam diversas características vantajosas e os experimentos nas mais diversas áreas comprovem isto, eles apresentam algumas limitações. A primeira está relacionada ao custo computacional para implementá-los. Já que a cada iteração é necessário construir o modelo probabilístico, os algoritmos não são indicados para problemas de otimização muito simples, pois existem técnicas mais eficientes para resolvê-los.

Outra limitação está relacionada com a perda de diversidade na população. Dado que tais algoritmos geram novas soluções com base na distribuição de probabilidade das melhores soluções selecionadas previamente, eles tendem a explorar o espaço de busca de uma forma polarizada. O algoritmo tende a visitar uma região do espaço de busca já visitada anteriormente, levando-o a mínimos locais facilmente. Dessa forma, qualquer algoritmo de estimação de distribuição básico apresenta desempenho

aquém do desejado quando é aplicado junto a problemas de otimização multimodal.

Por fim, a qualidade do modelo probabilístico influencia diretamente no comportamento do algoritmo. A construção de redes bayesianas ou redes gaussianas torna-se uma tarefa mais difícil frente a problemas de dimensão elevada, com muitas variáveis. Outro fator que pode comprometer a qualidade do modelo é a quantidade de dados disponíveis para construí-lo. Por essa razão, sugere-se que os AEDs trabalhem com populações contendo muitos indivíduos.

Na Seção 7 são apresentadas algumas possíveis abordagens para amenizar estas limitações.

5. Modelos Gráficos Probabilísticos

Esta seção fornece uma definição formal dos dois modelos gráficos probabilísticos mais utilizados em Algoritmos de Estimação de Distribuição: a rede bayesiana, utilizada quando o problema é de otimização com variáveis discretas, e a rede gaussiana, no caso das variáveis serem contínuas. Será descrita também uma possível abordagem para construir os modelos gráficos probabilísticos a partir de um conjunto de dados e, em seguida, utilizá-los para amostrar novos dados. Repare que estas duas tarefas são fundamentais para um Algoritmo de Estimação de Distribuição. A etapa de aprendizado do modelo corresponde à estimação da probabilidade conjunta das melhores soluções encontradas, enquanto a amostragem de novos dados a partir do modelo obtido corresponde à geração de novas soluções candidatas.

Formalmente, um modelo gráfico probabilístico para um conjunto $X=\{X_1, X_2, ..., X_n\}$ de variáveis aleatórias é uma fatoração gráfica da distribuição de probabilidade conjunta de X. Se as variáveis forem discretas, os modelos recebem o nome de rede bayesiana. Se as variáveis forem contínuas, o modelo é chamado de rede gaussiana. O modelo consiste de uma estrutura em grafo que representa as dependências condicionais das variáveis em X e de um conjunto de probabilidades para cada variável (Cowell et al., 1999; Jensen, 1996, 2001; Pearl, 1988).

Tanto as redes bayesianas quanto as redes gaussianas são modelos gráficos probabilísticos cuja estrutura é um grafo acíclico direcionado, no qual:

- cada nó corresponde a uma variável aleatória;
- os arcos ligando os nós indicam as relações de dependência entre as variáveis. Um arco saindo de uma variável X_i e chegando numa variável X_j significa que "X_i é pai de X_j" e que "X_j é filho de X_i";
- cada variável possui uma distribuição de probabilidade.

Na Figura 11.5, é apresentado um exemplo de modelo gráfico probabilístico que ilustra os conceitos definidos previamente. O conjunto de variáveis $X=\{X_1, X_2, X_3, X_4, X_5\}$ retrata as variáveis do modelo, que são representadas pelos nós do grafo.

Além da dependência condicional explícita entre as variáveis, os modelos gráficos probabilísticos representam implicitamente as independências condicionais entre elas. Uma variável é dita ser condicionalmente independente de outras que não são suas filhas, dados os seus pais. Na rede da Figura 11.5, a variável X_5 é condicionalmente independente de X_1, X_2 e X_4, dado seu pai X_3.

Portanto, no caso em que o modelo gráfico probabilístico da Figura 11.5 é uma rede bayesiana, a distribuição conjunta das variáveis pode ser expressa por:

$$p(x_1x_2x_3x_4x_5) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2x_3)p(x_5|x_3), \tag{11.3}$$

em que $p(.)$ representa a função massa de probabilidade.

De uma forma mais geral, a distribuição conjunta de X é dada por:

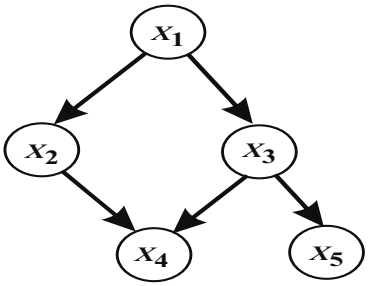


Figura 11.5: Exemplo de rede bayesiana.

$$p(x_1x_2x_3x_4x_5) = \prod_{i=1}^{5} p(x_i|\boldsymbol{pa_i}). \tag{11.4}$$

em que x_i é um possível valor da variável aleatória X e $\boldsymbol{pa_i}$ é o conjunto de pais da variável X_i. Por exemplo, na Figura 11.5, $\boldsymbol{pa_4}$=$\{X_2, X_3\}$. O termo $p(x_i|\boldsymbol{pa_i})$ é a probabilidade condicional de x_i, dados os valores das variáveis contidas no conjunto $\boldsymbol{pa_i}$.

Por outro lado, se for uma rede gaussiana, a distribuição conjunta é representada por uma distribuição gaussiana com vetor de médias $\boldsymbol{\mu}$ e matriz de covariância $\boldsymbol{\Sigma}$, da forma:

$$f(x_1x_2x_3x_4x_5) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-n/2}|\boldsymbol{\Sigma}|^{-1/2}e^{-1/2(x-\boldsymbol{\mu})^t\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})}, \tag{11.5}$$

em que $|\Sigma|$ denota o determinante da matriz de covariância e Σ^{-1} é a inversa da matriz de covariância, também chamada de matriz de precisão e algumas vezes denotada por W.

A decomposição em produtos da distribuição de probabilidade conjunta para X adquire a seguinte forma, quando o modelo utilizado é uma rede gaussiana:

$$f(x_1x_2x_3x_4x_5) = \prod_{i=1}^{5} f(x_i|\boldsymbol{pa_i}), \tag{11.6}$$

em que

$$f(x_i|\boldsymbol{pa_i}) = \mathcal{N}(\mu_i + \sum_{x_k \in \boldsymbol{pa_i}} b_{ki}(x_k - \mu_k), \sigma_i^2), \tag{11.7}$$

sendo que μ_i denota a média de X_i, σ_i^2 é a variância de X_i, condicionada aos pais de X_i e b_{ki} é o coeficiente de regressão linear refletindo o relacionamento entre as variáveis X_k e X_i. Se $b_{ki} = 0$, então não existe relacionamento entre as variáveis X_k e X_i.

Para tornar evidente a relação entre redes gaussianas e distribuições gaussianas multivariadas, Shachter e Kenley (1989) mostraram a transformação de b_{ki} e σ_i^2 da rede gaussiana em matriz de precisão W da distribuição gaussiana equivalente. A transformação é realizada por meio da seguinte fórmula recursiva:

$$W(i+1) = \begin{pmatrix} W(i) + \frac{b_{i+1}b_{i+1}^t}{\sigma_{i+1}^2} & \frac{-b_{i+1}}{\sigma_{i+1}^2} \\ \frac{-b_{i+1}^t}{\sigma_{i+1}^2} & \frac{1}{\sigma_{i+1}^2}, \end{pmatrix} \tag{11.8}$$

em que $W(i)$ denota a submatriz do canto superior esquerdo, $W(1) = \frac{1}{\sigma_i^2}$, b_i é o vetor coluna $(b_{1i}, ..., b_{(i-1)i})^t$ e b_i^t é o seu vetor transposto.

É a independência condicional que permite graficamente decompor a distribuição de probabilidade conjunta de X em produtos, levando a uma redução no número de parâmetros necessários para especificar o modelo.

Aprendizado de modelos gráficos probabilísticos

O aprendizado de modelos gráficos probabilísticos a partir de um conjunto de dados (no caso, as melhores soluções selecionadas) pode ser dividido em duas etapas. A primeira é a definição das relações de interdependência entre as variáveis, ou seja, a estrutura em grafo. Uma vez que a estrutura foi definida, entra em cena a segunda etapa, que é o cálculo das distribuições de probabilidade para as variáveis. Esta segunda tarefa é mais simples que a primeira. Basta maximizar a verossimilhança do modelo, considerando os dados.

Já o aprendizado da estrutura da rede é mais complexo e consiste em determinar relações entre as variáveis, adicionar ou excluir arcos, estabelecer direções, enfim definir um grafo acíclico direcionado para o qual serão calculadas distribuições de probabilidade. Geralmente, o aprendizado da estrutura pode ser considerado como um problema de otimização, em que um mecanismo de busca explora o espaço de busca contendo todos os possíveis grafos acíclicos (todas as possíveis topologias de redes), enquanto uma métrica de qualidade avalia cada solução candidata.

Com relação ao mecanismo de busca, geralmente é utilizado um algoritmo guloso não-populacional, como o de subida da encosta (do inglês *hill climbing*). Embora simples, ele tem sido aplicado com sucesso e apresentado bons resultados, tanto para redes bayesianas quanto para redes gaussianas. A busca pela melhor estrutura da rede bayesiana se inicia considerando uma estrutura de grafo básica (geralmente um grafo sem arcos ou completamente conectado). Em seguida, são realizadas algumas perturbações na estrutura corrente do grafo até que não seja mais possível obter melhoras na qualidade da rede, de acordo com a métrica escolhida. Perturbações típicas são adição, remoção e inversão de arcos. É necessário garantir que, após cada alteração da estrutura, ela ainda seja um grafo acíclico.

Na Figura 11.6, está ilustrado o esquema desta busca para um problema com 4 variáveis. O algoritmo começa sem ligação entre os nós (a) e uma avaliação desta rede não conectada é feita. Em seguida, adiciona-se um arco e a rede resultante é avaliada novamente (b). Se a nova estrutura for melhor, mantém-se o arco. Caso contrário, ele é retirado. Este processo continua até que nenhuma estrutura seja melhor que a anterior. No exemplo, a rede (e) é retornada como solução.

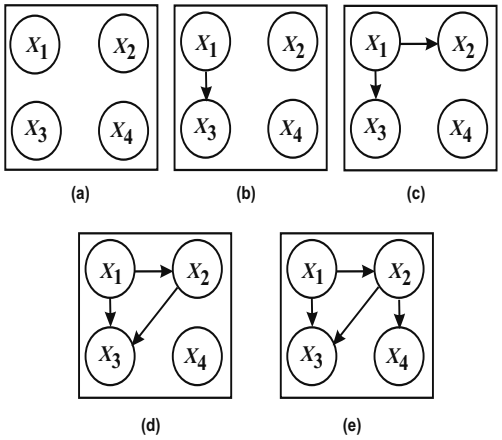


Figura 11.6: Processo iterativo para construção do modelo gráfico probabilístico.

Alguns trabalhos, no intuito de reduzir o espaço de busca, ordenam as variáveis numa lista de

modo que os possíveis pais de uma determinada variável só podem ser aqueles que aparecem antes dele na lista. Outra forma de reduzir a complexidade da busca é limitar a quantidade de pais que uma variável pode ter.

Um ponto que merece destaque é o fato de um AED ser um algoritmo de busca que utiliza outro algoritmo de busca mais simples para obter o modelo gráfico probabilístico. O leitor deve estar se perguntando por que não utilizar uma meta-heurística baseada em população para realizar esta tarefa. A razão para isso (utilizar algoritmo simples de busca local) é que o aprendizado do modelo probabilístico não visa resolver o problema de otimização que o AED está tentando resolver. Os AEDs não precisam da melhor rede bayesiana ou rede gaussiana, somente necessitam que estes modelos expressem algumas relações importantes entre as variáveis. Conforme exposto anteriormente, algoritmos como o de subida da encosta apresentam bons resultados e trocá-lo por um algoritmo baseado em população pode implicar em um aumento de custo computacional não condizente com o possível ganho de desempenho.

Com relação às métricas de avaliação da qualidade da rede bayesiana, as mais utilizadas são as métricas bayesianas. Estas métricas medem a qualidade da rede computando a sua verossimilhança com respeito aos dados, permitindo incluir conhecimento a priori sobre a estrutura e os parâmetros numéricos. Sua forma geral é:

$$P(B|D) = \frac{P(D|B)P(B)}{P(D)}, \tag{11.9}$$

em que B é a rede bayesiana sendo avaliada e D é o conjunto de dados disponível. Como $P(D)$ é constante para todas as redes, pode ser omitido. Portanto, as métricas maximizam $P(D|B)P(B)$.

Métrica para avaliar redes bayesianas

Heckerman et al. (1995) propuseram uma métrica que calcula a verossimilhança marginal e usa conhecimento a priori que segue uma distribuição de *Dirichlet*. A essa métrica, eles deram o nome de *Bayesian Dirichlet* (BD):

$$P(D|B) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(N'_{ij})!}{(N'_{ij} + N_{ij})!} \prod_{k=1}^{r_i} \frac{(N'_{ijk} + N_{ijk})!}{(N'_{ijk})!}, \tag{11.10}$$

em que D representa o conjunto de dados, B é a rede bayesiana sendo avaliada, n é o número de variáveis, q_i denota o número de possíveis valores que os pais de X_i podem assumir, r_i é o número de possíveis valores de X_i, N_{ijk} é o número de casos em que X_i assume o k-ésimo valor com seus pais assumindo o j-ésimo valor, e $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. Os termos N'_{ij} e N'_{ijk} representam conhecimento a priori sobre as estatísticas N_{ij} e N_{ijk}.

Outras métricas utilizadas são a métrica baseada na entropia (Acid e Campos, 1996), a métrica do Comprimento Mínimo de Descrição (MDL, do inglês *Minimum Description Length*) (Grünwald, 2000; Lam e Bacchus, 1994; Suzuki, 1996) e outros critérios clássicos para seleção de modelos, como BIC (do inglês *Bayesian Information Criterion*) (Schwarz, 1978) e AIC (do inglês *Akaike's Information Criterion*) (Akaike, 1974).

Métrica para avaliar redes gaussianas

A métrica mais utilizada para medir a qualidade de redes gaussianas foi derivada da métrica BIC, a qual utiliza o logaritmo da verossimilhança junto com uma função para penalizar redes muito complexas:

$$P(D|G) = \left[\prod_{l=1}^{N} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi v_i}} e^{-1/2v_i (x_{li} - \mu_i - \sum_{x_k \in \boldsymbol{pa_i}} b_{ki}(x_{lk} - \mu_k))^2} \right] - f(N) * dim(G), \tag{11.11}$$

em que D representa o conjunto de dados, G é a rede gaussiana sendo avaliada, N denota a quantidade de instâncias, n é o número de variáveis, b_{ki} é o coeficiente de regressão linear refletindo o relacionamento entre X_k e X_i, $\boldsymbol{pa_i}$ é o conjunto de pais da variável X_i e v_i é a variância condicional de X_i dados $X_1, ..., X_{i-1}$ $\forall i, k$. A função $f(N) = \frac{1}{2}\ ln\ N$ é responsável por penalizar modelos complexos e $dim(G) = 2n + \sum_{i=1}^{n} \boldsymbol{pa_i}$, que representa a quantidade de parâmetros a serem estimados.

Amostrando novas soluções

Uma vez que o modelo probabilístico foi gerado, novas soluções candidatas podem ser amostradas de acordo com a distribuição de probabilidade expressa pelo modelo. Para amostrar novas soluções, é utilizado o método da Amostragem Probabilística Lógica (PLS, do inglês *Probabilistic Logic Sampling*) (Henrion, 1986). Esse método amostra primeiramente todas as variáveis que não são dependentes de nenhuma outra. Em seguida, amostram-se as variáveis-pais para depois amostrar as variáveis-filhas.

6. Aplicações de AEDs

Por muito tempo, os AEDs permaneceram restritos a problemas de otimização mono-objetivo com variáveis discretas. Nesta seção, são apresentados os principais trabalhos reportados na literatura para outros tipos de problemas, como otimização com variáveis contínuas, otimização multiobjetivo e otimização dinâmica. São apresentadas também propostas em duas áreas de aplicações: bioinformática e aprendizado de máquina. O propósito desta seção é mostrar que os AEDs representam uma alternativa interessante em relação a outros algoritmos evolutivos e estão recebendo ótima aceitação da comunidade científica. Mais detalhes destas e outras aplicações podem ser encontradas em (Larrañaga e Lozano, 2002), (Pelikan, 2005a) e (Pelikan, 2005b).

Otimização com variáveis contínuas

Em se tratando de problemas de otimização com variáveis contínuas, existem os AEDs que utilizam uma função densidade de probabilidade gaussiana para modelar cada variável do problema, como no caso do RECEDA (Paul e Iba, 2003) e do PBIL$_c$ (Sebag e Ducoulombier, 1998). A cada iteração, a média e a variância de cada variável são atualizadas. Repare, portanto, que estes algoritmos não são capazes de capturar o relacionamento entre as variáveis.

Outras abordagens, como o EGNA (Larrañaga et al., 2000b) e IDEA (Bosman e Thierens, 2000), utilizam redes gaussianas para tentar capturar relacionamento entre múltiplas variáveis. A cada iteração, uma rede gaussiana é construída utilizando algum algoritmo de busca junto com uma métrica de avaliação das redes candidatas. Assim como nas versões discretas, novas soluções são amostradas a partir do modelo obtido. Estes algoritmos consideram que os dados disponíveis para construir o modelo probabilístico foram gerados por uma única distribuição gaussiana multivariada. Para problemas com funções unimodais, o algoritmo consegue estimar corretamente a distribuição de probabilidade para os dados. Entretanto, para funções multimodais, em que os ótimos locais não estão concentrados em uma única região, mas sim espalhados, uma única distribuição gaussiana multivariada não é capaz de modelar os dados de forma satisfatória.

Para sanar esta deficiência, alguns algoritmos passaram a trabalhar com modelo de mistura usando funções densidade de probabilidade gaussiana , como o *real-coded Bayesian Optimization Algorithm* (rBOA) (Ahn et al., 2006, 2004). Com esta modificação, a função densidade de probabilidade conjunta das melhores soluções é expressa por:

$$f(x) = \sum_{i=1}^{k} \alpha_i f_i(x), \qquad \sum_{i=1}^{k} \alpha_i = 1, \quad \alpha_i \geq 0, \tag{11.12}$$

em que $f_i(x)$ é uma única função densidade de probabilidade gaussiana multivariada, k é a quantidade de *clusters* e, consequentemente, a quantidade de componentes do modelo de mistura, e α_i é o coeficiente do i-ésimo componente da mistura. Os parâmetros da mistura podem ser obtidos via algoritmo EM (*Expectation Maximization*) ou por meio de algoritmos de agrupamento.

Os AEDs estão sendo aplicados com sucesso a várias funções comumente utilizadas na literatura, como a Griewank, Rosembrock, Rastrigin, Schwefel, Michalewicz e Ackley, superando outros algoritmos de otimização. Apenas para fornecer uma impressão da complexidade destes problemas, são apresentadas as definições e os gráficos de cada uma destas funções para duas dimensões. O emprego de duas dimensões aqui é para efeito de visualização gráfica, pois são consideradas na prática dezenas e até centenas de dimensões junto a essas funções de teste.

- **Griewank**: é uma função que possui muitos ótimos locais. Ela é definida como:

$$F(x) = 1 + \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}). \tag{11.13}$$

O valor do mínimo global desta função, para qualquer número de dimensões (d), é zero, o qual é obtido quando todas as variáveis assumem valor zero. Na Figura 11.7, é ilustrado o gráfico da função Griewank para duas dimensões.

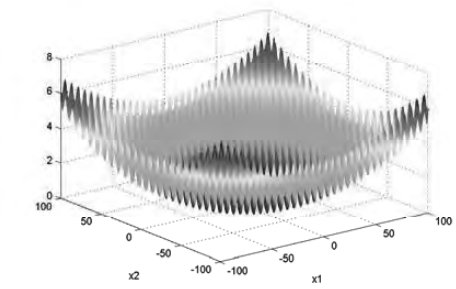


Figura 11.7: Gráfico da função Griewank para duas dimensões.

- **Rosenbrock**: é uma função bastante desafiadora para muitos algoritmos de otimização. Ela possui um vale longo, estreito e curvo, no qual está o ótimo global. A função é definida como:

$$F(x) = \sum_{i=1}^{d-1} 100(x_{i-1} - x_i^2)^2 + (x_i - 1)^2. \tag{11.14}$$

O valor do ótimo global, para qualquer número de dimensões (d), é zero, o qual é obtido quando todas as variáveis assumem o valor um. O gráfico desta função, para duas dimensões, é apresentado na Figura 11.8.

- **Rastrigin**: esta função é baseada na função esfera, a qual recebeu um termo modulador para permitir a existência de muitos ótimos locais. A função Rastrigin é definida por:

$$F(x) = 10d \sum_{i=1}^{d-1} (x_i^2 - 10\cos(2\pi x_i)). \tag{11.15}$$

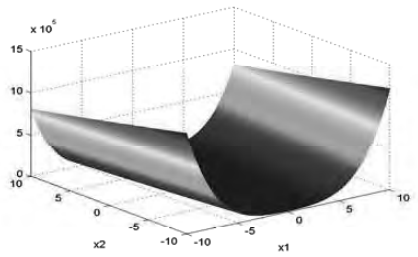


Figura 11.8: Gráfico da função Rosenbrock para duas dimensões.

O valor do ótimo global, para qualquer número de dimensões (d), é zero, o qual é obtido quando todas as variáveis assumem o valor zero. Na Figura 11.9, é ilustrado o gráfico da função Rastrigin, para duas dimensões.

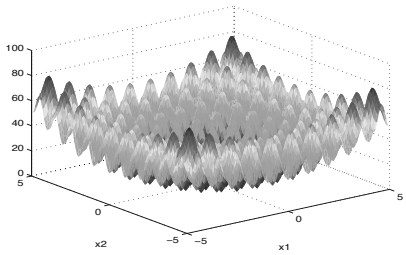


Figura 11.9: Gráfico da função Rastrigin para duas dimensões.

- **Schwefel**: é uma função composta por um grande número de picos e vales. O mínimo global se encontra distante dos mínimos locais, fazendo com que os algoritmos tendam a ficar presos nesses mínimos. Esta função é definida como:

$$F(x) = \sum_{i=1}^{d} -x_i \ \text{sen}(\sqrt{|x_i|}). \tag{11.16}$$

O valor do ótimo global é -418,9829d, o qual é obtido quando todas as variáveis assumem o valor 420,9687. O gráfico desta função, para duas dimensões, é apresentado na Figura 11.10.

- **Michalewicz**: esta função possui muitos ótimos locais e é definida por:

$$F(x) = -\sum_{i=0}^{d-1} \text{sen}(x_i)\text{sen}^{20}\left(\frac{(i+1)x_i^2}{\pi}\right). \tag{11.17}$$

O ótimo global depende do número de dimensões (d). Na Figura 11.11, é ilustrado o gráfico da função Michalewicz para duas dimensões.

- **Ackley**: é uma função considerada de dificuldade moderada, a qual é definida por:

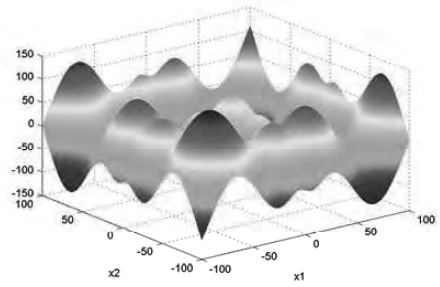


Figura 11.10: Gráfico da função Schwefel para duas dimensões.

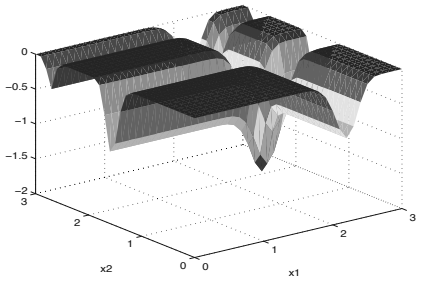


Figura 11.11: Gráfico da função Michalewicz para duas dimensões.

$$F(x) = -ae^{-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i}} - e^{\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)} + a + e^1. \tag{11.18}$$

O criador da função sugere utilizar os seguintes valores para os parâmetros: $a = 20$, $b = 0{,}2$ e $c = 2\pi$. O valor do ótimo global, para qualquer número de dimensões (d), é zero, o qual é obtido quando todas as variáveis assumem o valor zero. Na Figura 11.12, é ilustrado o gráfico da função Ackley para duas dimensões.

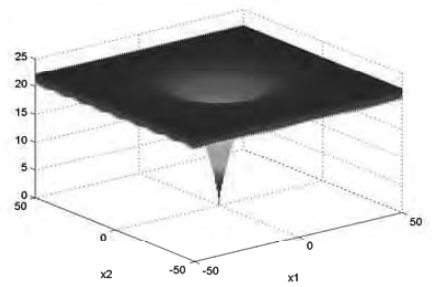


Figura 11.12: Gráfico da função Ackley para duas dimensões.

Maiores detalhes sobre AEDs para problemas com variáveis contínuas podem ser encontrados em (Ahn et al., 2006).

Otimização multiobjetivo

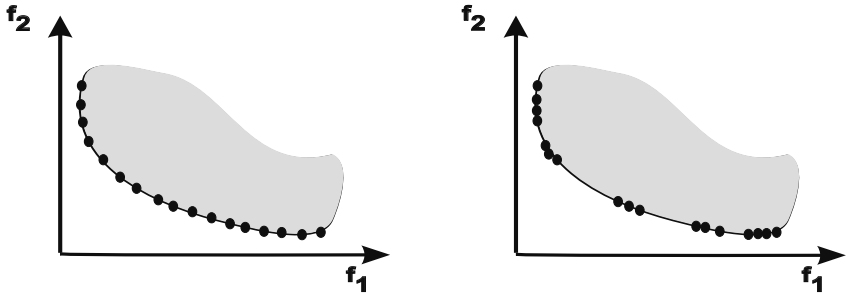
Um problema de otimização multiobjetivo consiste em otimizar simultaneamente duas ou mais funções-objetivo, possivelmente conflitantes, no sentido de que a melhora no valor de um dos objetivos pode degradar o valor dos outros (Deb, 2001; Zitzler et al., 2000). Nesse caso, a noção de otimalidade comumente adotada na literatura é aquela associada com a otimalidade de Pareto, a qual pode ser expressa usando o conceito de dominância. Uma solução domina outra se não é pior que a outra em nenhum objetivo e é estritamente melhor em pelo menos um objetivo. Uma solução é não-dominada se não existem soluções factíveis que a dominem, no sentido de que a melhora em algum objetivo vai implicar a piora em algum outro objetivo. Maiores detalhes podem ser encontrados no capítulo 17 deste livro.

A fronteira de Pareto é formada pelos valores das funções-objetivo correspondentes a cada solução que compõe o conjunto de soluções não-dominadas. Na ausência de informações adicionais sobre a relevância dos objetivos, todas essas soluções são igualmente importantes. Nesse sentido, duas metas práticas devem ser alcançadas por um algoritmo de otimização multiobjetivo:

1. Encontrar um conjunto de soluções que pertença ou que pelo menos esteja o mais próximo possível da fronteira de Pareto;

2. Encontrar um conjunto de soluções com a maior diversidade possível, de preferência uniformemente distribuídas ao longo da fronteira de Pareto, como no exemplo ilustrado na Figura 11.13(a). Já na Figura 11.13(b), é apresentado um exemplo de distribuição não desejável ao longo da fronteira de Pareto, pois embora possua a mesma quantidade de soluções que a proposta mostrada na Figura 11.13(a), as soluções possuem pouca diversidade entre si, ficando concentradas em algumas regiões da fronteira.

A primeira meta é bastante óbvia, já que a fronteira de Pareto satisfaz as condições de não-dominância ou otimalidade de Pareto. Já a segunda meta é necessária para evitar qualquer polarização em favor de uma função-objetivo específica. Somente com um conjunto diverso de soluções cuja distribuição se aproxima de uma distribuição uniforme pela fronteira de Pareto é que se pode promover um "equilíbrio" no que diz respeito a satisfazer os objetivos. A diversidade pode ser promovida nos espaços de decisão e de objetivos, uma vez que a proximidade de duas soluções no espaço de decisão não implica proximidade no espaço de objetivos, e vice-versa.



(a) (b)

Figura 11.13: Exemplos de resultados em otimização multiobjetivo (no espaço dos objetivos): (a) várias soluções distribuídas uniformemente e (b) soluções concentradas em algumas regiões.

A maioria dos AEDs capazes de tratar problemas de otimização multiobjetivo foi concebida tendo como inspiração os algoritmos NSGA-II (Deb et al., 2002) e SPEA2 (Zitzler et al., 2001).

Thierens e Bosman (2001) propuseram o *multiobjective mixture-based iterated density estimation algorithm* (mMIDEA), que utiliza seleção baseada no conceito de dominância. O algoritmo foi aplicado ao problema da mochila (Zitzler e Thiele, 1999) e vários outros com variáveis contínuas, obtendo bons resultados em todos os casos.

Laumanns e Ocenasek (2002) deram origem ao *multiobjective mixed BOA* (mmBOA) incorporando ao algoritmo BOA os operadores do SPEA2 e o conceito de uma segunda população. Quando aplicado ao problema da mochila, mmBOA produziu resultados que dominavam os resultados gerados pelo NSGA-II e SPEA2. Outra extensão do BOA foi proposta por Khan et al. (2002), só que desta vez utilizando o operador de seleção do NSGA-II.

Li e Zhang (2008) apresentaram um AED baseado em decomposição. Simultaneamente, o algoritmo resolve problemas mono-objetivos e depois agrega as soluções. Experimentos em problemas complexos mostraram que o algoritmo supera o NSGA-II significativamente.

Algoritmos capazes de se beneficiar das regularidades contidas na fronteira de Pareto para alguns tipos de problemas foram desenvolvidos por Zhang et al. (2008) e Zhou et al. (2008). Os algoritmos foram aplicados a vários tipos de problemas e comparados com outras propostas encontradas na literatura. Os resultados indicaram que os algoritmos capazes de identificar as regularidades do problema apresentam melhor desempenho.

Martí et al. (2008) e Sastry et al. (2005) apresentam as limitações dos AEDs quando aplicados a problemas de otimização multiobjetivos e apontam novas direções para a pesquisa nesta área.

Otimização em ambientes dinâmicos

No mundo real, muitos problemas são dinâmicos e requerem algoritmos capazes de se adaptarem às mudanças do ótimo global ao longo do tempo. Formalmente, a função objetivo para este tipo de problema é representada como $F(x, t)$, expressando a sua dependência não somente do vetor de variáveis x, mas também do tempo t.

Muitos trabalhos surgiram investigando a eficácia de AEDs junto a esta classe de problemas. Yang e Yao (2005) realizaram uma modificação no algoritmo PBIL, incorporando um mecanismo para manutenção de diversidade. Por meio de vários problemas, eles comparam PBIL com um algoritmo genético e levantaram os pontos positivos e negativos desse algoritmo quando aplicado a problemas de otimização dinâmica.

Liu et al. (2008) apresentaram uma versão do algoritmo UMDA que contém um mecanismo de manutenção de diversidade na população. O algoritmo proposto foi testado em vários *benchmarks* e os resultados mostraram que ele foi capaz de se adaptar às mudanças do ambiente rapidamente.

Fernandes et al. (2008) também adaptaram o UMDA adicionando um mecanismo de memória de soluções, a fim de manter diversidade e mesmo reinicializar o vetor de probabilidade quando ocorre mudança no ambiente. Resultados obtidos nos experimentos mostraram que o algoritmo é eficaz na resolução de problemas de otimização dinâmica, superando outros algoritmos evolutivos.

Já Yuan et al. (2008) propuseram um AED para problemas com variáveis contínuas. Eles estudaram um modelo gaussiano com matriz de covariância diagonal e, portanto, não expressam relacionamentos entre as variáveis. Os autores utilizam a estratégia de manutenção de diversidade na população, amostrando novas soluções aleatoriamente a cada n iterações.

Problemas de bioinformática

Algoritmos de Estimação de Distribuição têm se mostrado eficazes na resolução de uma variedade de problemas de bioinformática. Os pesquisadores nesta área alegam que o uso desta classe de algoritmos oferece vantagens sobre algoritmo evolutivos mais simples, como por exemplo, a possibilidade de estudar os modelos probabilísticos obtidos ao longo do processo de busca a fim de encontrar informações

adicionais. Devido à elevada dimensão do espaço de busca nesses tipos de problemas, a maioria dos trabalhos utiliza algoritmos mais simples, os quais consideram que as variáveis são independentes.

Peña, Lozano e Larrañaga (2004) apresentam a aplicação do UMDA para agrupar dados de expressão gênica. O algoritmo foi avaliado em conjuntos de dados artificiais e reais. Em ambos os casos, grupos de genes relevantes foram obtidos. Outra proposta de agrupamento utilizando o UMDA pode ser encontrada em (Cano et al., 2006).

Posteriormente, Palacios et al. (2006) aprimoraram o UMDA para trabalhar com *biclusterização* de dados de expressão gênica. Quando comparado com algoritmos genéticos, o algoritmo proposto apresentou melhores resultados.

Aplicações junto à predição de estrutura de proteínas usando modelos probabilísticos markovianos podem ser encontradas em (Santana et al., 2004) e (Santana et al., 2008b). Posteriormente, Santana et al. (2008a) combinaram um Algoritmo de Estimação de Distribuição com um processo de busca local para a mesma tarefa.

Para uma extensa lista de aplicações em bioinformática, consulte (Armañanzas et al., 2008).

Aprendizado de máquina

Os Algoritmos de Estimação de Distribuição estão sendo aplicados com sucesso também a problemas de aprendizado de máquina, especialmente em seleção de atributos, aprendizado de redes neurais artificiais e de sistemas nebulosos.

A seleção de atributos é um problema de busca e otimização multimodal em que os atributos possuem relacionamento. Por exemplo, considere um conjunto de dados com 10 atributos. Suponha também que foi verificado que qualquer subconjunto que possuir os atributos 3, 4 e 7, dentre outros, é tido como boa solução. Neste caso, a presença dos atributos 3, 4 e 7 no subconjunto forma um bloco construtivo (solução parcial). Inza et al. (2000) e Cantú-Paz (2002) trataram desse tipo de problema usando a abordagem *wrapper* (Kohavi e John, 1997). Os autores utilizaram AEDs para explorar o espaço de busca e o classificador naïve Bayes (Theodoridis e Koutroumbas, 2006) para avaliar cada solução candidata. Em seus algoritmos, o modelo probabilístico empregado foi a rede bayesiana.

Cotta et al. (2001) avaliaram as versões dos algoritmos UMDA e MIMIC para problemas com variáveis contínuas na tarefa de evoluir pesos de redes neurais artificiais MLP. Durante os experimentos, os autores compararam os algoritmos de estimação de distribuição com algoritmos genéticos e estratégias evolutivas. Já Galic e Höhfeld (1996) utilizaram o algoritmo PBIL para evoluir a arquitetura e os pesos de redes neurais.

O uso do UMDA e do MIMIC para a geração de regras SE-ENTÃO de sistemas nebulosos foi investigado por delaOssa et al. (2009). Os termos linguísticos para cada variável foram definidos previamente por meio de funções de pertinência triangulares distribuídas uniformemente ao longo do universo de discurso.

7. Novas Perspectivas em AEDs

Nesta seção, são apresentados alguns tópicos relevantes relacionados ao desenvolvimento de AEDs capazes de resolver problemas mais complexos, de explorar o espaço de busca da melhor forma e com um custo computacional associado reduzido. Os trabalhos citados a seguir representam as primeiras tentativas de se obter AEDs com estas características e, evidentemente, investigações futuras devem ser realizadas.

Informações adicionais sobre direções a serem estudadas no desenvolvimento de AEDs podem ser encontradas em (Santana et al., 2009) e (Sastry et al., 2006).

Algoritmos híbridos

No intuito de melhorar a capacidade de exploração do espaço de busca dos AEDs, surgiram algumas propostas combinando esta classe de algoritmos com técnicas simples de busca local (Li, Zhang, Tsang e Ford, 2004; Tang e Lau, 2005; Zhang et al., 2007, 2003). Mais recentemente, os AEDs estão sendo combinados com algoritmos que apresentam propriedades complementares, como evolução diferencial (Chen et al., 2008; Sun et al., 2005), sistemas imunológicos artificiais (Castro e Von Zuben, 2009a,b, 2008; Chang et al., 2009), enxame de partículas (Wang et al., 2009; Wang, 2007; Zhou et al., 2007) e algoritmos genéticos (Peña, Robles, Larrañaga, Herves, Rosales e Pérez, 2004; Robles et al., 2006, 2005).

Algoritmos paralelos

Visando distribuir o custo computacional entre vários processadores e, consequentemente, acelerar o tempo de execução do algoritmo, alguns trabalhos desenvolveram AEDs paralelos (Jaros e Schwarz, 2007; Ocenasek et al., 2006; Schwarz e Jaros, 2008). Geralmente, a arquitetura adotada é a mestre-escravo (do inglês *master-slave*), na qual existe um processador central incumbido de coordenar a divisão de tarefa entre os processadores escravos. Se o gargalo do algoritmo está na etapa de avaliação das soluções candidatas, esta tarefa é dividida entre p processadores escravos, enquanto o restante do algoritmo é executado no processador mestre. Por outro lado, se o gargalo está na construção do modelo probabilístico, o processador mestre realiza todas as outras etapas, deixando a construção do modelo sob responsabilidade dos processadores escravos.

Aprendizado incremental e esporádico do modelo probabilístico

Nos AEDs, a tarefa de construção do modelo probabilístico a cada iteração é, geralmente, a que consome mais tempo. Visando aliviar este custo computacional, alguns trabalhos têm se dedicado a projetar algoritmos em que o modelo probabilístico é construído esporadicamente, e não em todas as iterações (Pelikan et al., 2008b). O que permanece em todas as iterações é a atualização dos parâmetros numéricos do modelo probabilístico. Pelikan et al. (2008a) investigaram o aprendizado incremental do modelo probabilístico. A cada iteração, o modelo probabilístico obtido na iteração anterior é utilizado como ponto de partida.

8. Material Adicional

Em nenhum dos itens desta seção é apresentada uma lista exaustiva, sendo que podem existir outros bons *softwares*, congressos e periódicos que tratam de AEDs, além daqueles mencionados na sequência.

Softwares

- *Bayesian Optimization Algorithm* (BOA): `http://medal-lab.org/`.
- *Hierarchical Bayesian Optimization Algorithm* (hBOA): `http://medal-lab.org/`.
- *Mixed Bayesian Optimization Algorithm* (mBOA): `http://jiri.ocenasek.com/`.
- *Real-coded Bayesian Optimization Algorithm* (rBOA): `http://www.evolution.re.kr/`.
- *Multiobjective Real-coded Bayesian Optimization Algorithm* (mrBOA): `http://www.evolution.re.kr/`.

- *Regularity Model Based Multiobjective Estimation of Distribution Algorithm* (RM-MEDA): `http://dces.essex.ac.uk/staff/zhang/`.

Congressos

Existem duas principais conferências em que os pesquisadores na área de AEDs apresentam seus trabalhos:

- *IEEE Congress on Evolutionary Computation* (CEC).
- *Genetic and Evolutionary Computation Conference* (GECCO).

Periódicos

Os seguintes periódicos publicam com frequência trabalhos sobre AEDs:

- *Evolutionary Computation. MIT Press.*
- *IEEE Transactions on Evolutionary Computation. IEEE Press.*
- *Informations Sciences. Elsevier.*
- *International Journal of Approximate Reasoning. Elsevier.*

9. Conclusões

Neste capítulo, foram abordados os conceitos básicos de uma nova classe de algoritmos evolutivos, denominada Algoritmos de Estimação de Distribuição (AEDs). Em vez de explorar o espaço de busca evoluindo a população de soluções por meio de operadores de cruzamento e mutação, os AEDs utilizam um modelo gráfico probabilístico que representa a distribuição de probabilidade conjunta para as melhores soluções encontradas até então. A cada iteração, este modelo probabilístico é construído e, posteriormente, utilizado para amostrar novas soluções. Operando desta maneira, estes algoritmos são capazes de identificar as regularidades do problema e utilizar este conhecimento ao longo do processo de busca. Consequentemente, os AEDs podem manipular blocos construtivos (soluções parciais para o problema) de forma eficiente.

O capítulo apresentou as motivações que levaram ao desenvolvimento dos AEDs, seguido de um pseudocódigo para um AED canônico. Em seguida, foram apresentados os algoritmos mais importantes, conforme o modelo probabilístico utilizado para expressar o relacionamento entre as variáveis. Foram elencados os principais trabalhos reportados na literatura para problemas de otimização com variáveis contínuas, otimização multiobjetivo e otimização dinâmica. Algumas propostas na área de bioinformática e aprendizado de máquina também foram citadas. O capítulo forneceu também novas tendências para o desenvolvimento de AEDs com melhor capacidade de exploração do espaço de busca e com reduzido custo computacional. Por fim, fontes para informações adicionais foram incluídas, caso o leitor deseje se aprofundar no assunto.

CAPÍTULO 12

Pesquisa Local Iterativa e em Vizinhança Variável

Luís Paquete

Departamento de Engenharia Informática
Universidade de Coimbra

As metaheurísticas têm tido grande sucesso na resolução de imensos problemas de optimização que surgem tanto a nível académico como a nível industrial. Contudo, o desenho de metaheurísticas não é um exercício trivial. Para obter um desempenho competitivo com metaheurísticas é necessário ter um conhecimento muito aprofundado do estado-da-arte dos métodos existentes e do próprio problema em estudo.

Nas várias metaheurísticas, como a pesquisa local iterativa (Lourenço et al., 2002), pesquisa tabu (Glover, 1989) e pesquisa em vizinhança variável (Mladenović e Hansen, 1997), assume-se que existe uma *heurística* que toma decisões locais com base no conhecimento do problema.[1] O modo de funcionamento desta classe de metaheurísticas é "guiar" essa heurística no espaço de pesquisa com base na escolha apropriada de determinados parâmetros e métodos que resultam muitas vezes de experimentação algoritmica. A clara vantagem destas abordagens é que são conceptualmente simples de parameterizar e bastante eficazes, uma vez encontrada uma boa heurística para o problema em estudo.

Tanto a pesquisa local iterativa (ILS) como a pesquisa de vizinhança variável (VNS)[2] são duas das

[1] É possível encontrar o termo "Procura" e "Busca" em vez de "Pesquisa" em trabalhos científicos publicados em Portugal e no Brasil, respectivamente.

[2] Neste capítulo utilizaremos as iniciais da denominação anglo-saxónica, respectivamente, ILS (*Iterated Local Search*) e VNS (*Variable Neighborhood Search*).

metaheurísticas mais simples de parameterizar, sem que isso signifique uma perda de desempenho. Em particular, ILS tem demonstrado um desempenho bastante elevado em vários problemas clássicos nas Ciências da Computação e Investigação Operacional, tal como o problema do caixeiro viajante (Martin et al., 1991b; Codenotti et al., 1996; Johnson e McGeoch, 1997; Stützle e Hoos, 2002), problemas de escalonamento (Lourenço e Zwijnenburg, 1996; Balas e Vazacopoulos, 1998; Yang et al., 2000), o problema de afectação quadrática (Stützle, 2006) e o problema de coloração de grafos (Paquete e Stützle, 2002).

O *modus operandis* da ILS consiste em construir, de uma forma iterativa, uma sequência de soluções geradas por uma heurística subordinada. Em cada iteração, a solução inicial da heurística é obtida através de uma *perturbação* na solução retornada por essa heurística numa iteração anterior. Nas primeiras abordagens de ILS, a heurística consistia de um método estocástico de pesquisa local que retornava unicamente uma solução. Contudo, é possível encontrar variantes mais recentes de ILS em que a heurística subordinada é determinística, constructiva e/ou produz uma população de soluções tal como os algoritmos evolutivos. A ILS foi proposta por vários investigadores de forma independente, sendo conhecida por *large-step Markov chains* (Martin et al., 1991b) ou *iterated Lin-Kernighan* (Johnson e McGeoch, 1997). Recentemente, um grupo de investigadores considerou que estas abordagens seguiam princípios muito semelhantes, daí resultando a denominação de ILS numa tentativa de unificação (Lourenço et al., 2002).

A VNS segue um princípio muito semelhante ao da ILS. Ambas executam uma sequência de heurísticas subordinadas, alternadas com pequenas perturbações nas soluções retornadas por essas heurísticas. Por esta razão, a VNS é considerada como um caso particular de ILS. A ideia foi proposta por Mladenović e Hansen (1997) e embora não tenha ainda um desempenho tão marcante como ILS para o problema do caixeiro viajante, o conceito é bastante simples de implementar e tem apresentado resultados bastante promissores em vários problemas combinatórios, como para o problema de *p-mediana* (Hansen e Mladenović, 1997), problemas de optimização em *clustering* (Hansen e Mladenović, 2001) e de satisfiabilidade (Hansen et al., 2000).

O objectivo deste capítulo é apresentar uma descrição detalhada dos princípios da ILS e VNS e explicar casos de sucesso da sua aplicação em problemas de optimização. Assume-se que o leitor tenha conhecimentos básicos de optimização combinatória e métodos de pesquisa local (ver, p.e. Papadimitriou e Steiglitz (1982) e Hoos e Stützle (2004)). O capítulo está organizado da seguinte forma. As Secções 1 e 2 descrevem os fundamentos de ILS e os componentes algorítmicos desta metaheurística, respectivamente. Seguidamente, a Secção 3 apresenta um caso de estudo. A VNS é apresentada na Secção 4. Finalmente, temas para investigação futura nesta área são apresentados na Secção 5.

1. Fundamentos de Pesquisa Local Iterativa

Esta secção descreve os princípios básicos de ILS, tal como descritos em (Lourenço et al., 2002). Começamos por introduzir alguns conceitos introdutórios que permitirão uma compreensão mais alargada do funcionamento desta metaheurística.

Dado um determinado problema de optimização discreta , assume-se que existe uma heurística subordinada que devolve uma solução admissível para esse problema. Iremos denominar essa heurística de `PesquisaLocal`, visto que toma decisões a nível local.

Para um dado problema P com uma função objectivo f definimos como S o conjunto finito de soluções admissíveis desse problema. Sem perda de generalidade, assumimos que o objectivo do problema P é encontrar uma solução em S que minimize a função f. Para manter a simplicidade da exposição, assumimos igualmente que `PesquisaLocal` apresenta as seguintes especificidades: (i) é um método de pesquisa local; (ii) inicia o processo de pesquisa numa solução escolhida aleatoriamente em S; (iii) em cada iteração escolhe a solução vizinha com menor valor da função f (menor custo) e; (iv) termina num óptimo local.

Associada ao método de pesquisa local e ao problema P existe uma noção de vizinhança N que, quando *adicionada* ao conjunto S, forma uma determinada estrutura topológica $\mathcal{S}_N$, também conhecida por *espaço de pesquisa*. Em problemas discretos, esta estrutura pode ser representada por um grafo dirigido em que cada solução admissível é um nó e cada arco liga um nó u a outro nó v se e só se u é vizinho de v de acordo com a vizinhança N. Se considerarmos a informação da função f, este grafo passa a ter ponderações nos arcos que indicam a diferença de custo entre cada par de nós vizinhos, isto é, o peso do arco que liga o nó u ao nó v corresponde a $f(v) - f(u)$. A heurística `PesquisaLocal` pode ser entendida como um algoritmo que atravessa o grafo $\mathcal{S}_N$ a partir de um determinado nó inicial. Este algoritmo escolhe, iterativamente, o nó vizinho cujo o arco que o liga ao nó actual tem o menor peso e termina quando encontra um nó $s^\ell \in S_N$ com arcos com pesos não-negativos, isto é, s^ℓ é um óptimo local. Com base nesta representação do problema de pesquisa local podemos afirmar que a heurística `PesquisaLocal` mapeia cada solução admissível em S num elemento do conjunto $S^\ell \subseteq S$ de óptimos locais. Realçamos que a solução que minimiza a função objectivo também é um óptimo local, mas que está muito pouco acessível a `PesquisaLocal` devido às características do grafo S_N.

A distribuição do custo dos elementos em S e dos elementos em S^ℓ são tipicamente diferentes para o mesmo problema. É de esperar que esta última seja caracterizada por uma média e variância significativamente inferiores à média e variância do custo dos elementos em S. Por esta razão, será preferível efectuar uma amostragem em S^ℓ do que em S. Obviamente, esta observação só será válida se, na prática, o tempo computacional para obter uma solução em S^ℓ, por exemplo através de `PesquisaLocal`, for suficientemente curto. Por esta razão, é particularmente relevante considerar métodos incrementais de avaliação de soluções vizinhas de uma forma eficiente. De facto, existem muitas formas de o fazer em problemas discretos. Infelizmente, vários investigadores têm demonstrado que o número de vizinhos a visitar até encontrar um óptimo local é exponencial relativamente ao tamanho do problema no cenário de pior caso (Johnson et al., 1988). Contudo, outros investigadores têm observado que este tempo é negligível na prática, o que indica que a complexidade do problema de encontrar um óptimo local seja limitada por uma função polinomial no caso médio, tal como no caso do algoritmo de Simplex para problemas de programação linear (Klee e Minty, 1972).

Dada a vantagem em efectuar uma amostragem de S^ℓ, a principal questão que se coloca agora é como efectuar esse procedimento. O método mais simples consiste em executar `PesquisaLocal` múltiplas vezes a partir de uma solução escolhida aleatoriamente em S. Logo, cada óptimo local gerado por `PesquisaLocal` em cada uma dessas execuções é independente. Neste caso, podemos considerar que `PesquisaLocal` está a efectuar uma amostragem independente em S^ℓ. Contudo, a média amostral do custo dos óptimos locais está tipicamente localizada a uma percentagem fixa do custo do óptimo global (Lourenço et al., 2002). Por esta razão, supõe-se que este processo de amostragem tem uma probabilidade cada vez menor de encontrar bons óptimos locais com o aumento do tamanho da instância. Uma possibilidade de ultrapassar esta desvantagem é recorrer a um processo de amostragem enviesada, tirando partido de alguma estrutura do problema.

É possível idealizar um método de pesquisa local que, de acordo com uma determinada noção de vizinhança N^ℓ, pesquisa unicamente em S^ℓ até encontrar um óptimo local relativamente a N^ℓ. Espera-se que a distribuição do custo desses óptimos locais de acordo com a vizinhança N^ℓ tenha uma média e variância inferiores à média e à variância da distribuição dos custos dos óptimos locais em S^ℓ. Claramente, este princípio não é possível de ser implementado na prática porque tal noção de vizinhança não é conhecida *a priori*. Contudo, podemos definir o que seria desejável nessa vizinhança N^ℓ: Dado dois conjuntos de soluções S^1 e S^2 que pertencem a duas bacias de atracção distintas[3], estes dois conjuntos são vizinhos em N^ℓ se existe pelo menos uma solução $s^1 \in S^1$ e outra solução $s^2 \in S^2$ e s^1 e s^2 são vizinhas relativamente a N. Por outras palavras, as bacias de atracção são vizinhas se ambas se *tocam*. Claramente, o ideal é conceber um algoritmo que efectue uma caminhada em $\mathcal{S}_N$ que

[3] Para qualquer solução inicial numa bacia de atracção, o algoritmo `PesquisaLocal` irá sempre encontrar o mesmo óptimo local.

Algoritmo 1 Pesquisa Local Iterativa

1: $s \leftarrow$ `Gera`()
2: $s^{\ell} \leftarrow$ `PesquisaLocal`(s)
3: **repita**
4: $\quad s \leftarrow$ `Perturba`(s^{ℓ}, memória)
5: $\quad s^{\ell *} \leftarrow$ `PesquisaLocal`(s)
6: $\quad s^{\ell} \leftarrow$ `Aceita`($s^{\ell}, s^{\ell *}$, memória)
7: **até** condição de paragem ser verdadeira

permita sair de um bacia de atracção e entrar noutra bacia de atracção vizinha. Contudo, sabemos que é impossível encontrar esse caminho de um modo determinístico em tempo razoável, assim como saber a que bacia de atracção é que uma determinada solução pertence.

ILS tenta efectuar uma caminhada entre duas bacias de atracção de uma forma estocástica e heurística em cada iteração. A ideia é perturbar um óptimo local $s^{\ell} \in S^{\ell}$ para gerar uma solução $s \in S$; seguidamente, `PesquisaLocal` inicia a pesquisa a partir de s para retornar outro óptimo local $s^{\ell *} \in S^{\ell}$; finalmente, um determinado critério de aceitação decide qual dos óptimos locais encontrados, s^{ℓ} ou $s^{\ell *}$, é que vai ser perturbado na iteração seguinte. Claramente, ILS efectua uma pesquisa em S^{ℓ} mas sem utilizar uma noção explícita de vizinhança entre bacias de atracção.

O Algoritmo 1 apresenta o pseudo-código de ILS. Para evitar que ILS entre facilmente em ciclos durante a pesquisa em S^{ℓ}, consideramos "memória" como uma estrutura de dados adicional que retorna informação sobre as soluções já visitadas. Esta estrutura está associado à perturbação (`Perturba`) e ao critério de aceitação (`Aceita`). Estudos recentes indicam que a presença de memória melhora o desempenho de ILS. Realçamos que a definição da perturbação é naturalmente dependente do problema, mas o critério de aceitação pode ser definido de uma forma independente. Por exemplo, o critério de aceitação pode escolher a solução com o menor custo, tal como é efectuado pelo método de pesquisa local. É possível também considerar outros critérios de aceitação baseados em arrefecimento simulado ou pesquisa tabu.

2. Componentes de Pesquisa Local Iterativa

Uma grande vantagem do ILS, de um ponto de vista mais pragmático, é a sua modularidade. Como pode-se observar no Algoritmo 1, é possível instanciar várias configurações de ILS ao modificar os componentes algorítmicos `Gera`, `PesquisaLocal`, `Perturba` e `Aceita`. A optimização do ILS consiste em encontrar uma combinação destes quatro componentes que maximize o seu desempenho. Naturalmente, o investigador pode optimizar cada um destes componentes de um modo sequencial, sem considerar as interações entre estes, o que produz uma configuração sub-óptima de ILS. Realçamos que o problema de encontrar uma configuração óptima de uma metaheurística, quer seja ILS, algoritmos evolutivos ou colónia de formigas, é ainda uma questão em aberto. Contudo, os resultados experimentais obtidos por um grande número de investigadores indicam que uma boa afinação de metaheurísticas está dependente da experiência do investigador na abordagem que escolheu e no conhecimento que este tem do problema em estudo. Actualmente existem vários métodos que afinam metaheurísticas de uma forma automática com base em inferência estatística (ver Birattari et al. (2002)). Contudo, mesmo nestes métodos automáticos é necessário conhecer os efeitos de cada componente no desempenho de uma metaheurística. No texto que se segue apresentamos algumas sugestões que podem apoiar o investigador a efectuar as primeiras escolhas algorítmicas para um dado problema.

Geração de Solução Inicial O componente `Gera` é responsável pela geração da solução inicial da ILS. Existem duas maneiras conhecidas de gerar esta solução: aleatoriamente ou com base numa

heurística gulosa. Vários investigadores consideram que a segunda opção é preferível, principalmente porque a solução retornada pela heurística gulosa já apresenta algum nível de qualidade e porque estas heurísticas são muito eficientes do ponto de vista de tempo computacional. Contudo, é necessário ter algum cuidado com a escolha da heurística gulosa, pois esta pode retornar soluções que não são possíveis de serem melhoradas através do componente `PesquisaLocal`. Por exemplo, os resultados experimentais de Johnson e McGeoch (1997) no problema do caixeiro viajante indicam que o método de pesquisa local utilizado por estes autores apresenta melhor desempenho quando a solução inicial é gerada aleatoriamente do que através da heurística de Clarke-Wright. Por outro lado, tem-se verificado que existe pouca relação entre a qualidade da solução inicial e a qualidade da solução final quando se consideram execuções muito longas da ILS (Lourenço et al., 2002).

Perturbação A perturbação é normalmente definida por um parâmetro a que chamamos *força*. Por exemplo, se a perturbação efectua k iterações de uma pesquisa local que selecciona vizinhos de uma forma aleatória, então k é a força desta perturbação. A escolha de uma perturbação está claramente relacionada com a escolha da heurística para `PesquisaLocal`. Um requisito essencial é escolher uma perturbação que impeça a heurística `PesquisaLocal` de retornar uma solução numa bacia de atracção já visitada. Contudo, as perturbações não podem ser demasiado *fortes*, caso contrário, ILS não será nada mais do que um algoritmo de reinicialização aleatória. Normalmente, é necessário efectuar uma análise experimental com alguma profundidade para encontrar boas perturbações e/ou valores aceitáveis para o parâmetro de força. Resultados experimentais indicam que é preferível utilizar uma força pequena para o problema do caixeiro viajante, tal como a perturbação *double-bridge* (ver Secção 3), e para o problema de escalonamento de "flow shop" (Lourenço et al., 2002). Contudo, para o problema de afectação quadrática, a perturbação tem de alterar aproximadante 75% dos componentes da solução (Stützle, 2006).

Infelizmente, não é sempre possível obter uma perturbação simples e eficaz para muitos problemas. É possível utilizar um tipo de perturbação mais complexa em que, por exemplo, a força varie de acordo com o tempo computacional dispendido. Estes princípios adaptativos aproximam ILS de outras abordagens como pesquisa em vizinhança variável (Mladenović e Hansen, 1997) ou pesquisa reactiva (Battiti e Tecchiolli, 1994). Outra alternativa é considerar uma perturbação nos dados de entrada do problema e executar `PesquisaLocal` no problema modificado, tal como sugerido por Codenotti et al. (1996) para o problema do caixeiro viajante. Outra possibilidade é efectuar uma perturbação que resolve um subproblema, quer através de heurísticas ou mesmo de uma forma exacta, como proposto por Lourenço e Zwijnenburg (1996) para um problema de escalonamento.

Critério de Aceitação O componente `Aceita` determina qual dos dois óptimos locais, o actual ou aquele que foi retornado por `PesquisaLocal` numa iteração anterior, é escolhido para ser perturbado. Este procedimento permite ao investigador obter um balanço entre *exploração* e *intensificação* do ILS. Por exemplo, um critério de aceitação que aceita o melhor óptimo local para perturbação favorece claramente a intensificação. Por outro lado, um critério de aceitação que aceita sempre o óptimo local mais recente favorece a exploração do espaço de pesquisa.

Existem outras alternativas aos dois critérios de aceitação acima descritos. Por exemplo, é possível considerar um critério de aceitação baseado no algoritmo de arrefecimento simulado (Martin et al., 1991b). Neste caso existe uma probabilidade de aceitar um óptimo local com qualidade inferior de acordo com um parâmetro de temperatura. Este componente também pode ser combinado com alguma estrutura de dados adicional que mantém a memória dos passos anteriores efectuados por ILS. Por exemplo, se o critério de aceitação continua a aceitar o mesmo óptimo local por um determinado número de iterações, então fará sentido recorrer ao procedimento `Gera` para gerar outra solução para reinicializar ILS. Realçamos que, em qualquer caso, é necessário guardar a melhor solução encontrada até ao momento, pois essa é a solução que deverá ser retornada pelo ILS.

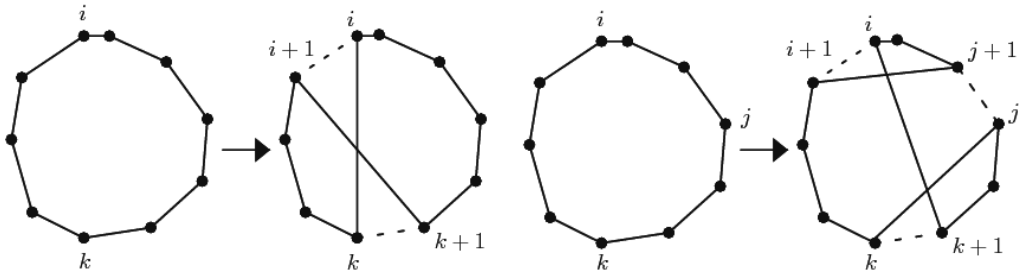


Figura 12.1: Ilustração da modificação do circuito com base no 2-opt e no 3-opt.

Pesquisa Local Para cada problema e para cada representação computacional desse problema podem existir diversas noções de vizinhança. Para cada uma dessas vizinhanças podem ainda existir diversas maneiras de a explorar, quer seja de um modo exaustivo, ou de um modo aleatório[4]. Finalmente, podem também existir diversas variantes para o critério de aceitação, semelhantes às descritas na secção anterior para o ILS. Infelizmente, não é possível determinar *a priori* qual é a melhor escolha para um dado problema e respectiva representação. Por isso, é necessário recorrer muitas vezes à experimentação. Espera-se que uma boa escolha para um método de pesquisa local seja também a melhor escolha para o componente `PesquisaLocal` do ILS. Contudo, é necessário ter em conta o tempo computacional disponível para executar um ILS, pois pode ser preferível ter um método de pesquisa local que retorne um óptimo local de inferior qualidade em muito pouco tempo do que um método que retorne um óptimo local de boa qualidade mas que requer bastante tempo computacional; no primeiro caso, ILS efectua mais iterações e pode explorar o espaço de pesquisa de um modo mais eficaz.

Um aspecto bastante importante no método de pesquisa local é a utilização de estruturas de dados e algoritmos que permitem a computação eficiente do custo das soluções vizinhas. Quanto mais eficiente é o método de pesquisa local, mais iterações de ILS é possível efectuar. Também não estamos restritos a métodos de pesquisa local no componente `PesquisaLocal`, pois é possível utilizar uma metaheurística como arrefecimento simulado ou mesmo pesquisa tabu (Lourenço e Zwijnenburg, 1996).

3. Um caso de estudo - O Problema do Caixeiro Viajante

O problema do caixeiro viajante é um problema clássico nas Ciências da Computação e na Investigação Operacional. Curiosamente, é neste problema que a metaheurística ILS apresenta o seu melhor desempenho. Na sua forma mais simples, este problema consiste em encontrar o circuito mais curto num conjunto de cidades em que cada cidade só pode ser visitada uma só vez. Os dados de entrada do problema podem ser representados por um grafo em que as cidades correspondem aos nós do grafo, os caminhos entre cidades correspondem às arestas do grafo e a distância entre cidades é o peso associado às arestas. O problema consiste em encontrar o ciclo Hamiltoniano mais curto nesse grafo.

Existem vários métodos de pesquisa local para este problema, mas os mais utilizados na literatura são o 2-opt e o 3-opt. O 2-opt baseia-se na troca de duas arestas não-adjacentes no circuito por

[4] Note que se o método de pesquisa local só considera a vizinhança de um modo parcial, então não é possível garantir que esse método retorna um óptimo local; contudo, como poderemos observar no caso de estudo da Secção 3, este pormenor é muitas vezes deixado de parte por questões de eficiência.

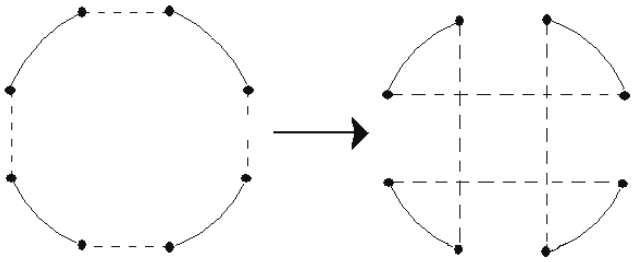


Figura 12.2: Ilustração da perturbação *double-bridge*.

outras duas arestas não-adjacentes que não estão presentes no circuito. O 3-opt é semelhante mas considera três arestas. A Figura 12.1 ilustra as modificações efectuadas no circuito. Um método de pesquisa mais complexo mas com melhor desempenho em termos da qualidade da solução retornada é o Lin-Kernighan. Esta abordagem pode ser entendida como um k-opt em que o algoritmo decide o valor de k para cada iteração com base em determinadas condições verificadas na solução (ver Lin e Kerninghan (1973) para uma descrição mais detalhada).

A primeira ILS para este problema foi proposta por Baum (1986b). Esta abordagem continha vários métodos de pesquisa local que escolhiam a primeira solução com menor custo encontrada na vizinhança (estratégia de *primeira-melhoria*). A perturbação consistia em aplicar uma iteração de 2-opt de um forma aleatória.

Martin et al. (1991b) apresentaram o primeiro ILS a obter um grande desempenho neste problema. Esta abordagem utilizava um método de pesquisa local baseado em 3-opt com estratégia de primeira-melhoria. Mais tarde, os mesmos autores observaram que o algoritmo de Lin-Kernighan era preferível ao 3-opt em termos de desempenho do ILS. A perturbação escolhida em todas abordagens foi a *double-bridge*, que consiste em efectuar uma só iteração de 4-opt, como é ilustrado na Figura 12.2. De acordo com os autores, esta perturbação só deve ser efectuada se a soma dos pesos das quatro arestas a remover é inferior a um valor proporcional ao peso médio das arestas da solução actual. Não só é impossível reproduzir esta perturbação através de pequeno número de iterações de 2-opt, 3-opt ou Lin-Kerningham, como o custo da solução perturbada não difere muito do custo da solução actual. Por esta razão, esta perturbação tem aparecido em vários estudos de ILS para este problema e tem sido extremamente eficaz. Finalmente, os autores utilizaram um critério de aceitação baseado em arrefecimento simulado. É necessário realçar que o bom desempenho do ILS final é também resultante das técnicas inovadoras e eficientes que foram propostas para a avaliação parcial da vizinhança sem grande prejuízo da qualidade final.

Uma variante da ILS descrita acima foi proposta por Johnson e McGeoch (1997), também denominada *Iterated Lin-Kernighan*. Nesta abordagem, a perturbação consistia de um movimento *double-bridge* sem as condições impostas nos pesos. O critério de aceitação seleccionava a solução com menor custo e a solução inicial era obtida por uma heurística gulosa aleatorizada. Os resultados obtidos por estes autores tornaram-se estado-da-arte para problemas do caixeiro viajante com milhares de cidades.

Outra abordagem semelhante foi proposta por Applegate et al. (2003). Esta ILS, denominada *Chained Lin-Kernighan*, utilizava também o método de pesquisa local de Lin-Kernighan, tal como os seus sucessores, mas considerava um número restrito de arestas. A perturbação é baseada em *double-bridge* mas só é aplicada a uma pequena parte da solução actual. As variantes seguintes reportadas por estes autores deram origem a um dos algoritmos mais rápidos para o problema do caixeiro viajante.[5] A implementação final consegue retornar soluções a 1% do óptimo para problemas com 25 milhões de cidades em menos de um dia de processamento num só computador pessoal.

[5] As implementações estão disponíveis em `http://www.tsp.gatech.edu/concorde.html`

A última versão mais conhecida do ILS para este problema foi proposta por Helsgaun (2000) e é conhecida por *Iterated Helsgaun Algorithm*. Esta ILS utiliza igualmente o método de pesquisa local de Lin-Kernighan. Contudo, diferente dos seus sucessores, utiliza uma perturbação que constrói uma solução com base na heurística de vizinho mais próximo, aproveitando algumas arestas da solução actual. O critério de aceitação também só aceita a solução com menor custo. Esta abordagem é actualmente umas das mais competitivas e encontra o óptimo para problemas com dezenas de milhares de cidades em poucos minutos.

4. Pesquisa de Vizinhança Variável

A metaheurística VNS utiliza um método de pesquisa local que alterna entre uma sequência de vizinhanças $N^1, N^2, \ldots, N^m$ (Mladenović e Hansen, 1997). De acordo com estes autores, esta abordagem explora três factos: i) um óptimo local para uma determinada vizinhança não é necessariamente um óptimo local para uma vizinhança diferente; por outras palavras, VNS está constantemente a modificar as bacias de atracção; ii) o óptimo global é também um óptimo local para qualquer uma das m noções de vizinhança; iii) em muitos problemas discretos verifica-se que óptimos locais para uma ou várias noções de vizinhança estão relativamente "perto" entre si. De realçar que este último facto é, na realidade, uma observação empírica mas que tem sido constatada por vários autores. Por exemplo, este parece ser o caso para o problema do caixeiro viajante para determinadas noções de vizinhança (Boese et al., 1994).

Na primeira versão de VNS, denominada *Descida em Vizinhança Variável* (VND), a mudança de vizinhança é efectuada a cada iteração do método de pesquisa local subordinado. Esta mudança segue uma determinada sequência definida *a priori*. A metaheurística termina quando não é possível encontrar uma solução vizinha com custo inferior para todas as vizinhanças. Numa versão de VNS mais avançada, denominada *VNS Reduzida*, a mudança para a vizinhança seguinte é efectuada se não existir uma solução vizinha com custo inferior. Normalmente, a solução vizinha é escolhida de um forma aleatória. A partir do momento que uma solução com menor custo é encontrada, o método de pesquisa local volta a utilizar a vizinhança que ocupa a primeira posição da sequência de vizinhanças. Esta abordagem tem grandes vantagens relativamente à anterior em termos de tempo computacional, pois evita a exploração total da vizinhança e permite que a primeira vizinhança, normalmente de menor complexidade, seja utilizada mais vezes. De notar que esta variante termina de acordo com algum critério definido pelo utilizador, iterações ou tempo computacional, dado que não é possível identificar um óptimo local quando a solução vizinha é escolhida aleatoriamente.

A abordagem mais conhecida actualmente é a *VNS Básica* e é a que se aproxima mais da ILS. Inicialmente, uma solução é gerada aleatoriamente ou de uma forma heurística. Esta é a solução inicial para um método de pesquisa local que utiliza a vizinhança N^1 e que retorna um óptimo local. Seguidamente, este procedimento é repetido por várias iterações em que a solução inicial é escolhida aleatoriamente na vizinhança do óptimo local da iteração anterior de acordo com uma vizinhança N^k previamente seleccionada. O critério de aceitação de VNS selecciona, entre o novo óptimo local e o anterior, aquele que tiver menor custo; se for o novo óptimo local, então a vizinhança seleccionada para a próxima próxima iteração será a que ocupa a primeira posição, caso contrário, será a vizinhança seguinte na sequência. É de realçar que o processo de gerar solução inicial em cada iteração da VNS Básica não é nada mais do que uma perturbação típica de ILS. A maior diferença entre as duas abordagens é que VNS Básica explora as noções de vizinhança de um modo mais explícito, sendo por isso um caso particular do ILS.

Existem muitas variantes de VNS na literatura. Por exemplo, é possível considerar diversas ordens na sequência de vizinhanças. Também é possível encontrar critérios de aceitação baseados em arrefecimento simulado ou com base numa determinada medida de distância entre soluções. Outra versão mais complexa é a *VNS de Decomposição* cuja ideia principal é aplicar o método de pesquisa local só

em determinados componentes da solução.

5. Conclusões

Com este capítulo pretendeu-se introduzir, de um modo breve, os princípios básicos de ILS e VNS. Claramente, ainda há muito mais para ler e investigar. O leitor mais interessado poderá explorar os artigos que serviram de base para este texto tais como (Lourenço et al., 2002; Mladenović e Hansen, 1997) ou o livro de Hoos e Stützle (2004) e recolher informação mais actualizada em actas de conferências da *Metaheuristics International Conference*, *Stochastic Local Search Workshop* e *Learning and Intelligent Optimization Workshop*.

ILS ocupa um lugar invejável relativamente às outras metaheurísticas, não só pela sua simplicidade mas também no que diz respeito ao seu desempenho para o problema do caixeiro viajante. Contudo, é possível também encontrar abordagens de ILS com desempenho bastante competitivo para problemas de escalonamento, problemas de afectação quadrática, problema de coloração de grafos, problema de MAX-SAT e muitos mais. Existem ainda muitos problemas a serem abordados por esta metaheurística, entre os quais, problemas de optimização que lidam com múltiplos objectivos e em que os dados de entrada são estocásticos e/ou são fornecidos dinamicamente.

Dado o bom desempenho do ILS numa grande classe de problemas, é necessário entender a razão deste desempenho além da explicação intuitiva fornecida por este texto. Uma possibilidade é recolher informação pertinente sobre a estrutura do espaço de pesquisa, por exemplo, através de determinadas estatísticas do grafo subjacente ao tuplo *(problema, representação, vizinhança)*.

Finalmente, as interações entre os vários componentes de ILS e VNS necessitam de ser melhor entendidas. Existe algum *know-how* sobre como optimizar os vários componentes, mas não existe nenhuma metodologia que o faça de um modo sistemático para um dado problema. O recurso a técnicas estatísticas, tal com desenho experimental, permitem quantificar os efeitos e interações dos diversos componentes no desempenho global de ILS e VNS.

(Página deixada propositadamente em branco)

PARTE III

Tópicos Avançados

(Página deixada propositadamente em branco)

CAPÍTULO 13

Distâncias Generalizadas: Espaços Combinatórios

Eduardo G. Carrano

Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais

Os algoritmos desenvolvidos para otimização de variáveis contínuas realizam a busca explorando propriedades favoráveis do espaço vetorial, no qual estão definidas as variáveis (Luenberger, 1969). Por exemplo, algoritmos de direção de busca se baseiam na possibilidade de definir uma direção que possibilita decrescimento da função, enquanto que, algoritmos de otimização unidimensional dependem da definição de uma distância ao longo de uma direção (Luenberger, 1984).

Mesmo no caso de problemas não-convexos, não-diferenciáveis ou multimodais, as características do espaço vetorial podem ser exploradas por algoritmos de otimização. Por exemplo, direções em que existam tendências de decrescimento, são exploradas de alguma forma por algoritmos evolucionários (AE's), como *Algoritmos Genéticos Reais* (Takahashi et al., 2003), *Algoritmos Particle Swarm Optimization* (Kennedy, 1997; Suganthan, 1999), *Algoritmos de Evolução Diferencial* (Storn e Price, 1997), etc. Uma vez definidos em problemas contínuos, estes algoritmos herdam diretamente o conceito de vizinhança inerente ao espaço vetorial, tornando a busca local implicitamente definida.

Estas operações baseadas nas propriedades do espaço vetorial garantem aos AE's contínuos uma certa generalidade: um mesmo algoritmo geralmente pode ser aplicado em problemas com funções objetivo de diferentes características.

Por outro lado, algoritmos evolucionários em que as soluções são representadas por códigos, como por exemplos algoritmos evolucionários voltados à otimização de problema discretos, não herdam a

capacidade de exploração do espaço vetorial. Uma particularidade destes algoritmos é que sua estrutura de vizinhança no espaço de busca é definida pela forma com que os operadores evolutivos percorrem as soluções, o que depende da representação adotada e a forma com que a operação é definida. A escolha do conjunto representação evolucionária/medida de distância ou representação evolucionária/operadores evolucionários afeta consideravelmente o desempenho do algoritmo de otimização (Moraglio et al., 2007; Carrano, Fonseca, Takahashi, Pimenta e Neto, 2007; Carrano et al., 2010). Essa alteração de desempenho pode ser creditada ao ordenamento relativo das soluções, que é dependente da medida de distância adotada ou os operadores utilizados, e afeta diretamente a forma com que o espaço é percorrido. O uso de uma medida de distância adequada para a representação adotada, e a escolha de operadores que se guiem eficientemente por essa distância, favorecem a otimização, pois permitem a realização de buscas aproximadamente convexas no espaço de busca, reduzindo a ocorrência de ótimos locais (Moraglio e Poli, 2004; Moraglio et al., 2007; Carrano, 2007; Carrano et al., 2010).

Este capítulo discute a generalização de AE's para o tratamento de problemas discretos utilizando operações inspiradas na otimização contínua. São relatados esforços no sentido de criar métricas de distância adequadas e operadores que percorrem as estruturas de vizinhança induzidas por essas métricas. Essas adaptações geralmente permitem a extensão de ferramentas que dependem de operações inicialmente restritas à espaços contínuos, como por exemplo exploração de vizinhanças, interpolações de soluções, etc. O capítulo apresenta a seguinte estrutura:

- A Sec. 1 discute o conceito de *panorama de aptidão*, identificando as relações que existem entre espaço de códigos / domínio da função / imagem da função.
- A Sec. 2 apresenta alguns exemplos de distâncias generalizadas propostas na literatura para problemas discretos.
- A Sec. 3 discute o conceito de operadores geométricos, proposto em (Moraglio e Poli, 2004, 2005; Moraglio et al., 2007), que visa unificar os conceitos que suportam o desenvolvimento de algoritmos evolucionários.
- Na Sec. 4 são apresentadas operações contínuas generalizadas para otimização de redes em árvore. Essas operações são utilizadas para construção de um algoritmo genético (GA), que pode ser aplicado na solução de problemas dessa classe.
- Na Sec. 5 é apresentado um estudo de caso, onde o GA proposto na Sec. 4 é aplicado na solução do *Optimal Communication Spanning Tree* (OCST).

1. Panorama de Aptidão

Seja um espaço de configuração qualquer $\mathcal{C} = (\mathcal{G}, Nhd)$, onde $\mathcal{G}$ é o conjunto de configurações sintáticas (genótipos no caso de algoritmos evolucionários) e $Nhd : G \to 2^G$ é uma função sintática de vizinhança que mapeia para cada configuração $G \in \mathcal{G}$ o conjunto de configurações vizinhas, que podem ser obtidos a partir da mesma utilizando um operador sintático de modificação unitária. A definição de modificação unitária é estabelecida através da escolha de uma função sintática de vizinhança, que identifica quais configurações diferem da configuração G em uma unidade. O operador sintático de modificação unitária deve ser simétrico ($y \in Nhd(x) \Leftrightarrow x \in Nhd(y)$) e conexo (uma configuração pode ser transformada em qualquer outra através da aplicação desse operador um número finito de vezes).

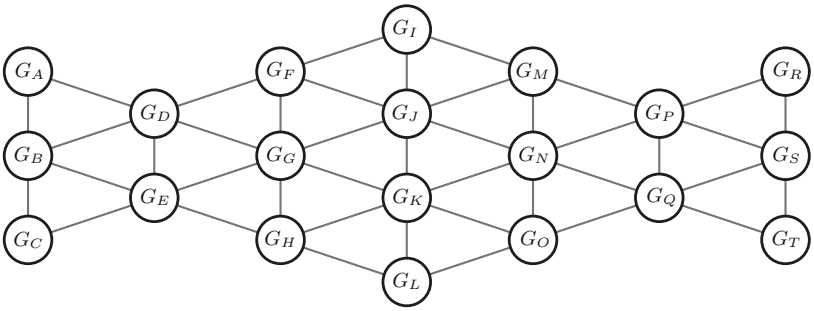
A função sintática de vizinhança confere ao espaço $\mathcal{C}$ uma estrutura de vizinhança que pode ser representada por um grafo não direcionado $W(V, E)$, onde V é o conjunto de vértices que representa as configurações e E é o conjunto de arestas que representa as relações entre essas configurações.

Uma mudança na função sintática de vizinhança adotada implica na alteração da estrutura de vizinhança do problema, e consequentemente altera a forma com que o operador de modificação unitária percorre o espaço. A Fig. 13.1 ilustra dois grafos induzidos por duas funções sintáticas de vizinhança distintas. Em ambos os casos G_G foi utilizado como exemplo para mostrar a variação do conjunto de vizinhos conforme muda a função sintática de vizinhança. Neste exemplo fica clara a alteração na forma de percorrer os espaço do operador de modificação unitária:

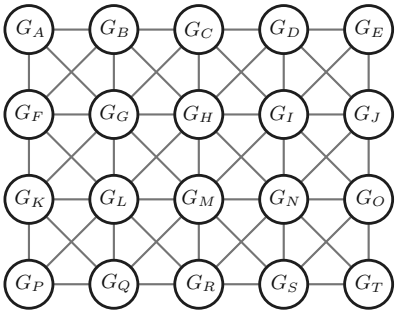
- Para a função nhd_1, a configuração G_E é vizinha de G_G, e portanto pode ser obtida através de uma única modificação unitária;

- Já para a função nhd_2, a configuração G_E não é vizinha de G_G, e são necessárias no mínimo três modificações unitárias para encontrar G_E a partir de G_G: $G_G \rightarrow G_C \rightarrow G_D \rightarrow G_E$.

Uma vez que a estrutura de vizinhança é simétrica e conexa, o espaço definido é um espaço métrico, munido de uma função de distância d, que é induzida pela função sintática de vizinhança (Bäck et al., 1997). Então, se torna equivalente definir $\mathcal{C} = (\mathcal{G}, Nhd)$ ou $\mathcal{C} = (\mathcal{G}, d)$. Entretanto, é importante notar que as distâncias originadas de um espaço de configuração sintático têm uma natureza sintática, e portanto podem não obedecer exatamente os axiomas que definem uma medida de distância em espaços vetoriais.



(a) $Nhd_1(G_G) = \{G_D, G_E, G_F, G_H, G_J, G_K\}$



(b) $Nhd_2(G_G) = \{G_A, G_B, G_C, G_F, G_H, G_K, G_L, G_M\}$

Figura 13.1: Exemplos de grafos induzidos por duas funções sintáticas de vizinhança distintas, Nhd_1 e Nhd_2.

Um "mapeamento de representação" é uma função $r : \mathcal{G} \rightarrow \mathcal{S}$ que associa cada configuração sintática G a uma solução formal $S \in \mathcal{S}$, onde $\mathcal{S}$ é o espaço onde está definido o domínio da função em estudo. Idealmente esse mapeamento deve ser bijetivo, mas existem casos em que $|\mathcal{G}| \neq |\mathcal{S}|$.

O *panorama de aptidão* (do inglês *fitness landscape*) F é um par $(\mathcal{C}, f)$, onde $\mathcal{C} = (\mathcal{G}, d)$ é um espaço de configurações e $f : \mathcal{G} \rightarrow \mathcal{R}$ é uma função que associa cada configuração sintática G ao seu valor de aptidão, definido em $\mathcal{R}$. A função de aptidão é um valor, geralmente positivo, que pode ou não coincidir com o valor de função objetivo correspondente ao genótipo G. Para efeito de simplicidade, será assumido neste texto que estes valores são idênticos, e portanto o valor de aptidão f é uma composição do "mapeamento de representações" r e a função objetivo g: $f = r \circ g$.[1]

Localidade no Espaço de Representações *vs.* Localidade no Domínio da Função: Reflexo na Imagem da Função

Como já foi discutido ao longo dessa sessão, a representação adotada para codificação das soluções e a função de distância escolhida no espaço de representações afetam o panorama da função objetivo e, consequentemente, o ordenamento relativo das soluções no espaço de busca do AE. Neste caso, é alterado o panorama da função objetivo, já que a mesma passa a ter como parâmetro de entrada não um ponto do domínio da função, mas uma representação, que mapeia um ponto no domínio da função através da função de mapeamento r:

$$f(S \in \mathcal{S}) = f(r(G \in \mathcal{G})) \tag{13.1}$$

A estrutura de vizinhança imposta pela escolha da representação e da função sintática de vizinhança atua fortemente na forma com que os operadores percorrem os espaço, o que pode ser favorável ou desfavorável ao processo de otimização. Um exemplo de como a representação medida de distância afetam a estrutura de vizinhança de um problema é apresentado na sequência, para um problema contínuo onde as soluções são representadas por cadeias binárias em um AE qualquer.

Exemplo 1

Seja o seguinte problema de minimização:

$$x^* = \arg\min_x \left[2 + (x-3)^2 + \sin(5 \cdot \pi \cdot x)\right] \quad , \quad 0 \leq x \leq 8 \tag{13.2}$$

Suponha que este problema seja resolvido utilizando um AE com representação binária de 5 bits por variável, e utilizando como operador de modificação unitário um operador que se orienta pela distância de Hamming (i.e. este operador é capaz de gerar a partir de um ponto P pontos cuja distância de Hamming em relação a P seja 1) (Hamming, 1950, 1980).

A distância de Hamming é brevemente apresentada aqui para facilitar o entendimento do exemplo. A mesma será discutida com mais detalhes na Sec. 2.

Distância de Hamming: Sejam duas cadeias binárias A e B, compostas de n bits cada uma. A distância de Hamming entre estas duas cadeias, $D_H(A, B)$, é determinada por:

$$D_H(A, B) = \sum_{i=1}^{n} \delta_H(A_i, B_i) \tag{13.3}$$

[1] Descrição adaptada de (Moraglio e Poli, 2004).

onde:

$$\delta_H(A_i, B_i) = \begin{cases} 0 & \text{se} \quad A_i = 0 \quad e \quad B_i = 1 \\ 0 & \text{se} \quad A_i = 1 \quad e \quad B_i = 0 \\ 1 & \text{se} \quad A_i = 0 \quad e \quad B_i = 0 \\ 1 & \text{se} \quad A_i = 1 \quad e \quad B_i = 1 \end{cases}$$

Uma vez que foi utilizada uma representação de 5 bits, o espaço de busca desse problema é composto por 32 soluções possíveis. A Fig. 13.2 ilustra essa função e os 32 pontos de espaço de busca. A representação binária, o valor real e o valor de função objetivo correspondentes à cada uma dessas soluções são apresentados na Tab. 13.1.

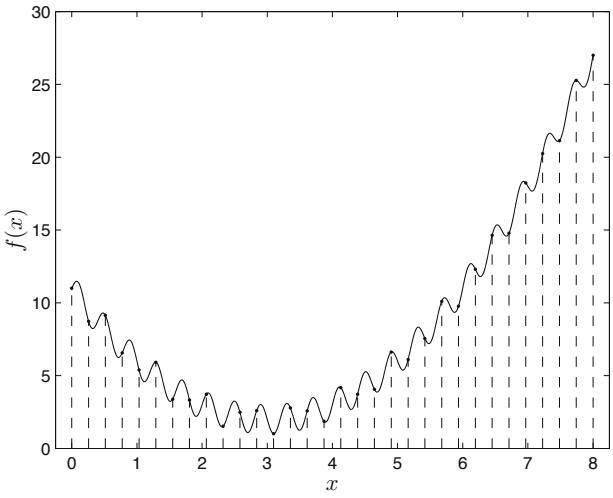


Figura 13.2: Exemplo 1: Função objetivo e pontos correspondentes à representação de 5 bits.

A análise de vizinhança desse conjunto de soluções pode ser realizada no domínio da função, $\mathbb{R}$, onde é avaliada a função objetivo, ou no espaço de genótipos, $\mathbb{B}^5$, onde as soluções são representados no AE[2].

Vizinhança no domínio da função: Devido à discretização proposta, no domínio da função cada solução candidata possui dois vizinhos, sendo um a esquerda e um a direita (exceto os dois extremos, que possuem um vizinho cada). Por exemplo, o ponto $x_4 = 1.0323$ é vizinho dos pontos $x_3 = 0.7742$ e $x_5 = 1.2903$. Este ponto constitui um mínimo local, uma vez que seu valor de função objetivo é menor que o de todos os seus vizinhos $f(x_4) = 5.3867$, $f(x_3) = 6.5599$ e $f(x_5) = 5.9115$. A distância euclideana entre x_4 e seus vizinhos é $D_E(x_3, x_4) = D_E(x_4, x_5) = 0.2581$.

Vizinhança no espaço de genótipos/representações: Uma vez que neste caso a estrutura de vizinhança é definida pela distância de Hamming, cada solução candidata possui 5 vizinhos, sendo cada vizinho obtido através da variação de cada um dos bits da cadeia individualmente. Neste espaço, o ponto $x_4 = 00100$ é vizinho de $x_5 = 00101$, $x_6 = 00110$, $x_0 = 00000$, $x_{12} = 01100$ e $x_{20} = 10100$. A distância de Hamming entre x_4 e seus vizinhos é necessariamente 1. Por outro lado, no espaço de genótipos x_4 não é vizinho de x_3, pois $D_H(x_3, x_4) = 3$. Outro fato importante

[2] $x \in \mathbb{B}$ se $x \in \{0, 1\}$.

a ser observado é que, este ponto não constitui um mínimo local no espaço de genótipos, uma vez que x_6 e x_{12} têm valor de função objetivo menor que o mesmo ($f(x_4) = 5.3867$, $f(x_6) = 3.3824$ e $f(x_{12}) = 1.0106$).

Tabela 13.1: Exemplo - Código binário, valor real correspondente e valor de função

	x_{bin}	x	$f(x)$		x_{bin}	x	$f(x)$
x_0	00000	0.0000	11.0000	x_{16}	10000	4.1290	4.1725
x_1	00001	0.2581	8.7274	x_{17}	10001	4.3871	3.7227
x_2	00010	0.5161	9.1377	x_{18}	10010	4.6452	4.0552
x_3	00011	0.7742	6.5599	x_{19}	10011	4.9032	6.6210
x_4	00100	1.0323	5.3867	x_{20}	10100	5.1613	6.0999
x_5	00101	1.2903	5.9115	x_{21}	10101	5.4194	7.5539
x_6	00110	1.5484	3.3824	x_{22}	10110	5.6774	10.1063
x_7	00111	1.8065	3.3234	x_{23}	10111	5.9355	9.7684
x_8	01000	2.0645	3.7238	x_{24}	11000	6.1935	12.2999
x_9	01001	2.3226	1.5211	x_{25}	11001	6.4516	14.6384
x_{10}	01010	2.5806	2.4752	x_{26}	11010	6.7097	14.7732
x_{11}	01011	2.8387	2.5973	x_{27}	11011	6.9677	18.2283
x_{12}	01100	3.0968	1.0106	x_{28}	11100	7.2258	20.2518
x_{13}	01101	3.3548	2.7773	x_{29}	11101	7.4839	21.1370
x_{14}	01110	3.6129	2.5769	x_{30}	11110	7.7419	25.2767
x_{15}	01111	3.8710	1.8608	x_{31}	11111	8.0000	27.0000

Com base no exemplo apresentado é possível perceber que a utilização de diferentes representações e medidas de distância afeta o comportamento da função, devido à alteração da estrutura de vizinhança do algoritmo. A análise desse aspecto é fundamental no estudo de problemas discretos, uma vez que, nestes problemas, as soluções candidatas são sempre representadas por códigos. O espaço em que estes códigos estão definidos não é necessariamente dotado de uma caracterização geométrica trivial, o que pode dificultar a construção de operadores que explorem adequadamente as propriedades do mesmo. O estudo de métricas de distância generalizadas, aplicáveis a estes problemas, pode facilitar substancialmente a construção de AE's eficientes para cada problema. Na Sec. 2 são apresentados alguns exemplos de métricas de distância aplicáveis a problemas discretos.

É importante destacar que existem vários trabalhos que estudam o estabelecimento de analogias entre os espaços contínuo e discreto. Como exemplo, pode-se citar:

- (Christodoulakis et al., 2005; Guo et al., 2008) utilizam métricas específicas para detectar similaridades de cadeias de caracteres.
- (Lafage e Lang, 2005) propõem uma família de medidas de distâncias que são utilizadas para comparação de preferências em tomadas de decisão.
- (Galton, 1999) propõe uma estrutura topológica para um espaço discreto genérico. A estrutura topológica proposta caracteriza uma topologia espacial viável do ponto de vista matemático.
- (Galton, 2000, 2003) introduzem o conceito de "deslocamentos aproximadamente contínuos" em espaços discretos.
- (Billera et al., 2001) apresentam um modelo que permite representar árvores filogenéticas binárias no espaço contínuo. A representação proposta permite o calculo de distâncias entre árvores filogenéticas, além de facilitar o processo de combinação das mesmas.
- (Moraglio e Poli, 2004) apresentam um ambiente (*framework*) para desenvolvimento de algoritmos evolucionários gerais, em que se torna necessária apenas a definição de uma representação evolucionária e uma medida de distância ou uma representação evolucionária e um operador de

mutação. Este trabalho é discutido na Sec. 3, juntamente com outros trabalhos dos mesmos autores.

- (Deza e Huang, 1998) apresentam métricas para estimar diferenças entre permutações.
- (Carrano et al., 2010) propõem uma representação que transforma redes genéricas em vetores embarcados no espaço contínuo $\mathbb{R}^n$. Com base nessa representação se torna possível estimar a posição relativa, produto escalar e a distância entre redes. Esses conceitos permitem a extensão de operadores de buscas local, interpolação e busca unidimensional para o problema de otimização de redes. Este trabalho é melhor discutido nas Secs. 2 e 4.

2. Exemplos de Medidas de Distância em Problemas Combinatórios

Conforme citado anteriormente, a existência de medidas de distância capazes de proporcionar um melhor ordenamento das soluções em problemas discretos pode ser útil na construção de AE eficientes. Isso serviu de motivação para vários estudos, cujo intuito principal era estabelecer medidas de distância adequadas aos mais diversos tipos de problemas discreto. No entanto, antes de introduzir algumas dessas métricas, é importante apresentar as propriedades necessárias para que uma métrica possa ser considerada uma medida de distância.

Definio 13.1 (Métrica de Distância) *Uma função D, que calcula um escalar que expressa a diferença entre dois vetores quaisquer $\overrightarrow{A}, \overrightarrow{B} \in \mathbb{X}^m$ (onde $\mathbb{X}^m$ é um espaço qualquer m-dimensional qualquer), define uma métrica de distância se, e somente se, a mesma cumpre os axiomas A.1, A.2, A.3 e A.4, listados abaixo:*

- *A.1: $D(\overrightarrow{A}, \overrightarrow{B}) \geq 0$*
- *A.2: $D(\overrightarrow{A}, \overrightarrow{B}) = 0$ se, e somente se, $\overrightarrow{A} = \overrightarrow{B}$ (identidade)*
- *A.3: $D(\overrightarrow{A}, \overrightarrow{B}) = D(\overrightarrow{B}, \overrightarrow{A})$ (simetria)*
- *A.4: $D(\overrightarrow{A}, \overrightarrow{C}) \geq D(\overrightarrow{A}, \overrightarrow{B}) + D(\overrightarrow{A}, \overrightarrow{C})$ (desigualdade triangular)*

Métricas em que o axioma A.2 é relaxado, de forma que a distância zero não implica em igualdade, mas a igualdade sempre implica em distância 0, são chamadas pseudo-métricas.

Algumas métricas de distância propostas na literatura para problemas discretos são apresentadas na sequência dessa seção:

distância de Hamming Generalizada

A distância de Hamming (Hamming, 1950, 1980), apresentada na Sec. 1 para estimar a diferença entre duas cadeias binárias, pode ser estendida para um caso mais geral, onde podem ser comparadas duas cadeias de qualquer tipo, desde que possuam o mesmo número de elementos. Essa extensão é apresentada em (13.4).

$$D_{HG}(A, B) = \sum_{i=1}^{n} \delta_{HG}(A_i, B_i) \tag{13.4}$$

onde:

$$\delta_{HG}(A_i, B_i) = \begin{cases} 0 & \text{se} \quad A_i = B_i \\ 1 & \text{se} \quad A_i \neq B_i \end{cases} .$$

Com essa adaptação se torna possível comparar vetores de qualquer tipo, como por exemplo cadeias de caracteres, inteiros, reais, binários, etc. A única restrição imposta é igualdade de comprimento dos dois vetores.

Exemplo 2

Sejam duas cadeias, X_1 e X_2, de seis números inteiros cada, conforme apresentado abaixo:

$$\begin{aligned} X_1 &= [1\ 3\ 2\ 4\ 6\ 5] \\ X_2 &= [2\ 3\ 4\ 1\ 6\ 5] \end{aligned}$$

Conforme a Eq. (13.4), a distância de Hamming Generalizada entre X_1 e X_2 é igual ao número de posições em que essas cadeias são diferentes, portanto:

$$D_{HG}(X_1, X_2) = 3$$

Distância *Labeling-Independent*

No problema de particionamento de grafos, que é um problema de agrupamento, cada solução é representada por um vetor X de tamanho n (n é o número de elementos que devem ser agrupados). Em cada posição i desse vetor se encontra um inteiro X_i, que indica a que grupo o termo i pertence.

Exemplo 3

Um exemplo de solução candidata para o problema de agrupamento é ilustrado abaixo:

$$X = [2\ 1\ 2\ 1\ 3\ 2]$$

Nesta solução existem seis elementos, que são divididos em 3 grupos distintos: os elementos 2 e 4 pertencem ao grupo 1; os elementos 1, 3 e 6 pertencem ao grupo 2 e o elemento 5 pertence ao grupo 3.

Este tipo de representação é altamente redundante, uma vez que qualquer solução que agrupe o elemento 5 em um grupo, os elementos 1, 3 e 6 em outro grupo e os elementos 2 e 4 em um terceiro representa, do ponto de vista prático, a mesma solução que X. A solução $Y = [3\ 2\ 3\ 2\ 1\ 3]$ é um exemplo de código que representa uma solução idêntica à X no domínio da função.

A alta redundância desse tipo de representação faz com que a distância de Hamming não seja adequada para comparar soluções em problemas de agrupamento, uma vez que a mesma geralmente vai detectar diferenças em soluções que do ponto de vista prático são idênticas. Visando contornar as limitações da distância de Hamming para este problema, (Moraglio et al., 2007) propõem a distância *labeling-independent*, conforme descrita abaixo:

Sejam A e B duas cadeias de tamanho fixo n, representadas no alfabeto K-ário ($A, B \in \mathcal{U} = \{1, 2, \ldots, K\}$), e δ_{LI} uma métrica definida em $\mathcal{U}$. A distância *labeling-independent* relacionada à métrica δ_{LI} é definida como segue:

$$D_{LI}(A, B) = \min_{\sigma \in \sum_K} \delta_{LI}(A, B_\sigma) \tag{13.5}$$

onde:
$\sum_K$ é o conjunto de todas as permutações possíveis de comprimento K;
B_σ é a cadeia B permutada conforme a permutação σ.

Continuação do Exemplo 3

No Exemplo 3, apresentado anteriormente nessa seção, os elementos estão divididos em 3 grupos distintos. Com isso, existem 6 permutações possíveis para cada solução: $\sigma_1 = [1\ 2\ 3]$, $\sigma_2 = [1\ 3\ 2]$, $\sigma_3 = [2\ 1\ 3]$, $\sigma_4 = [2\ 3\ 1]$, $\sigma_5 = [3\ 1\ 2]$ e $\sigma_6 = [3\ 2\ 1]$. Portanto, X possui seis permutações cujo sentido prático da solução é exatamente o mesmo: $X_{\sigma_1} = [1\ 2\ 1\ 2\ 3\ 1]$, $X_{\sigma_2} = [1\ 3\ 1\ 3\ 2\ 1]$, $X_{\sigma_3} = [2\ 1\ 2\ 1\ 3\ 2]$, $X_{\sigma_4} = [2\ 3\ 2\ 3\ 1\ 2]$, $X_{\sigma_5} = [3\ 1\ 3\ 1\ 2\ 3]$ e $X_{\sigma_6} = [3\ 2\ 3\ 2\ 1\ 3]$ representam a mesma solução.

A distância de Hamming entre X e cada uma dessas cadeias é:

$$D_{HG}(X,Y) = \begin{cases} 6 & \text{se} \quad Y = X_{\sigma_1} \\ 6 & \text{se} \quad Y = X_{\sigma_2} \\ 0 & \text{se} \quad Y = X_{\sigma_3} \\ 6 & \text{se} \quad Y = X_{\sigma_4} \\ 6 & \text{se} \quad Y = X_{\sigma_5} \\ 6 & \text{se} \quad Y = X_{\sigma_6} \end{cases}$$

Com base nesses valores, fica fácil constatar que a distância de Hamming é ineficiente para esse tipo de problema: $D_{HG}(X,Y)$ assume valor 0 apenas para o caso em que Y é exatamente igual à X ($Y = X_{\sigma_3}$), e assume valor máximo, 6, para todas as outras permutações de X. Este não é comportamento esperado para este tipo de problema, uma vez que, do ponto de vista prático, X, X_{σ_1}, X_{σ_2}, X_{σ_3}, X_{σ_4}, X_{σ_5} e X_{σ_6} representam a mesma solução.

Já a distância *labeling-independent* entre X e qualquer uma de suas permutações é necessariamente 0. Caso seja considerada uma permutação efetivamente distinta de X, como por exemplo $Z = [1\ 1\ 1\ 2\ 2\ 3]$, essa distância é não nula. Neste caso, a permutação de Z que mais se aproxima de X é $Z_{\sigma_3} = [2\ 2\ 2\ 1\ 1\ 3]$, e portanto:

$$D_{LI}(X,Y) = 2$$

A distância *labeling-independent* elimina completamente o problema da redundância da representação utilizada em problemas de agrupamento, uma vez que ela testa todas as $K!$ permutações possíveis da cadeia B. (Moraglio et al., 2007) demonstram que a distância *labeling-independent* obedece todos os axiomas que caracterizam uma pseudo-métrica, e pode ser calculada de forma eficiente utilizando o Método Húngaro (Kuhn, 2004).

Distância de Levenshtein (*Edit Distance*)

A distância de Levenshtein, também conhecida como distância de edição (do inglês *edit distance*), é utilizada para detecção de similaridades em cadeias de caracteres (Guo et al., 2008). Esta distância calcula o mínimo de operações de edição necessárias para transformar uma cadeia de caracteres A em outra cadeia de caracteres B. As operações de edição compreendem inserções, apagamentos e substituições, sendo que todas possuem peso 1 (Christodoulakis et al., 2005). Matematicamente, a distância de Levenshtein pode ser definida como:

$$D_{LV}(A,B) = \min_{\sigma_A \in \sum_A\ ,\ \sigma_B \in \sum_B} \delta_{LV}\left(A_{\sigma_A}, B_{\sigma_B}\right) \tag{13.6}$$

onde:
$\sum_A$ e $\sum_B$ são os conjuntos de todos os alinhamentos das cadeias A e B respectivamente;
S_{σ_i} é a cadeia S alinhada conforme o alinhamento σ_i.

Suponha que se calcular a distância de Levenshtein entre as cadeias *VINTNER* e *WRITERS*. Neste caso é necessário determinar o número mínimo de operações de edição necessárias para transformar

a palavra VINTNER na palavra WRITERS, ou vice-versa. O alinhamento que apresenta a menor número de operações para estas duas palavras é:

$$A^* : \begin{cases} \texttt{V-INTNER-} \\ \texttt{WRI-T-ERS} \end{cases}$$

Neste caso são necessárias 5 operações: uma substituição (`V` pelo `W` na primeira posição), duas inserções (`R` na segunda posição e `S` na última posição) e dois apagamentos (`N` na quarta posição e `N` na sexta posição). Portanto, a distância de Levenshtein entre essas cadeias de caracteres é 5:

$$D_{LV}(\mathit{VINTNER}, \mathit{WRITERS}) = 5.$$

Com base neste exemplo, fica fácil perceber que outros alinhamentos conduziriam à outras distâncias, necessariamente maiores que a encontrada para o alinhamento ótimo A^*. Outros possíveis alinhamentos para essas duas cadeias são apresentados abaixo:

$$A_1 : \begin{cases} \texttt{VINTNER} \\ \texttt{WRITERS} \end{cases} \quad A_2 : \begin{cases} \texttt{VINTNER-} \\ \texttt{-WRITERS} \end{cases} \quad A_3 : \begin{cases} \texttt{-VINTNER-} \\ \texttt{WRIT--ERS} \end{cases}$$

Para estes três alinhamentos o número de operações de edição necessárias é sempre seis.[3]

A distância de Levenshtein cumpre todos os axiomas estabelecidos no início da Sec. 2 para definição de medidas de distância. Apesar do alto número de alinhamentos possíveis, segundo (Moraglio et al., 2006) o alinhamento ótimo da distância de Levenshtein pode ser calculado de forma eficiente utilizando programação dinâmica.

Distância *T-norma*

(Carrano et al., 2010) propõem um procedimento que permite transformar soluções de problemas combinatórios modelados por redes em vetores esparsos, embutidos no espaço vetorial $\mathbb{R}^m$ de Hilbert. A transformação das soluções é baseada em uma definição de norma, chamada de *T-norma* (do inglês *T-norm*) que é possível devido à definição de um produto escalar estabelecido adequadamente. Esse procedimento é apresentado na sequência.

Seja $G(\mathcal{V}, \mathcal{A}, \mathcal{B})$ um grafo não direcionado com $|\mathcal{V}| = n$ vértices, $|\mathcal{A}| = m$ arestas e $|\mathcal{B}| = b$ tipos de arestas distintos e ordenados (1 se refere ao tipo de aresta mais fraco e b o tipo de aresta mais forte). Seja N um sub-grafo de G que contém necessariamente todos os vértices do conjunto $\mathcal{V}$. É possível representar N como um vetor $\overrightarrow{N}$ embutido no espaço $\mathbb{R}^m$ de Hilbert, conforme proposto em (13.7).

$$\overrightarrow{N} = \sum_{i=1}^{m} w_i^N \cdot (p + \phi(N_i)) \cdot \overrightarrow{e_i} \tag{13.7}$$

onde:
w_i^N é o *peso topológico* da aresta i na árvore N. Este peso é um valor real positivo se a conexão i existe ou 0 caso contrário;
N_i é o tipo de aresta da conexão i na árvore N, com $0 \leq N_i \leq b$. Os tipos de arestas devem estar ordenados de forma que as conexões "mais fortes" recebam valores maiores de N_i;
$\phi(\cdot)$ é uma função monotônica crescente, com $\phi(1) = 0$ no caso de problemas *multi-branch*, e é a função constante nula $\phi(\cdot) \equiv 0$ em problemas *mono-branch*;
p é um fator constante tal que $p \geq \phi(b)$;
$\overrightarrow{e_i}$ é o i-ésimo vetor da base canônica.

[3] Exemplo extraído de (Moraglio et al., 2006).

Esta expressão atribui à cada aresta possível no grafo uma dimensão do espaço $\mathbb{R}^m$. As referências (Carrano, 2007; Carrano et al., 2010) mostram que os vetores definidos por essa expressão obedecem os axiomas necessários para definição de vetores em espaços vetoriais (Lima, 1995).

Esse procedimento permite o tratamento tanto de problemas *mono-branch*, em que existe apenas um tipo de aresta possível, quanto de problemas *multi-branch*, em que uma mesma aresta pode assumir mais de um tipo. Problemas *mono-branch* ocorrem em casos em que se necessita definir apenas se duas arestas estão conectadas ou não, como por exemplo em redes de computadores e redes de sensores sem fio. Problemas *multi-branch* surgem em casos em que dois vértices podem ser conectados fisicamente por arestas de diferentes tipos, como por exemplo redes de energia elétrica, onde existem condutores de diferentes bitolas, ou redes de água, onde podem ser utilizados tubos de diferentes diâmetros.

No caso de problemas *mono-branch*, em que $\phi(\cdot) \equiv 0$, o valor de p pode ser definido, sem perda de generalidade, como $p = 1$, reduzindo (13.7) para:

$$\overrightarrow{N} = \sum_{i=1}^{m} w_i^N \cdot \overrightarrow{e_i} \tag{13.8}$$

Se a conexão i não existe, então w_i^N recebe 0. Caso contrário, é atribuído ao *peso topológico* w_i^N um valor que pode ser interpretado como a importância topológica da conexão i na rede N. Por exemplo, em redes estruturadas em árvores, é esperado que mudanças próximas à raiz tenham maior impacto no fluxo da rede que mudanças próximas as folhas. Portanto, deve-se associar valores maiores de w_i^N para as conexões mais próximas à raiz. No caso geral, as regras específicas para definir w_i^N devem ser elaboradas tendo em conta as características de cada problema.

Em (13.7), a função crescente $\phi(N_i)$, que tem como argumento o inteiro N_i, expressa a força do tipo de conexão da aresta i (tipos de arestas mais fortes são associados à valores mais altos de N_i e, portanto, de $\phi(N_i)$ também). A desigualdade $p \geq \phi(b)$ faz com que o efeito de acrescentar ou retirar uma aresta seja maior que o efeito de mudar um tipo de aresta, na avaliação da expressão $(p + \phi(N_i))$.

Sejam agora A e B duas redes quaisquer, tais que:

$$\begin{aligned} \overrightarrow{A} &= \sum_{i=1}^{m} w_i^A \cdot (p + \phi(A_i))\, \overrightarrow{e_i} \\ \overrightarrow{B} &= \sum_{i=1}^{m} w_i^B \cdot (p + \phi(B_i))\, \overrightarrow{e_i} \end{aligned}$$

O vetor diferença que representa a posição relativa de A em relação a B pode ser definido como:

$$\begin{aligned} \overrightarrow{(B,A)} &= \overrightarrow{A} - \overrightarrow{B} \\ &= \sum_{i=1}^{m} \left[w_i^A \cdot \phi(A_i) - w_i^B \cdot \phi(B_i) + p \cdot \left(w_i^A - w_i^B \right) \right] \overrightarrow{e_i} \end{aligned} \tag{13.9}$$

Um produto escalar dos vetores $\overrightarrow{A}$ e $\overrightarrow{B}$ pode ser definido por:

$$\overrightarrow{A} \cdot \overrightarrow{B} = \sum_{i=1}^{m} \left[w_i^A \cdot (p + \phi(A_i)) \right] \cdot \left[w_i^B \cdot (p + \phi(B_i)) \right] \tag{13.10}$$

Este produto escalar define uma norma:

$$\overrightarrow{A} \cdot \overrightarrow{A} = \| \overrightarrow{A} \|^2 \tag{13.11}$$

que, por sua vez, define uma distância:

$$D_{TN}(A,B) = \sqrt{\sum_{i=1}^{m} \left(w_i^B \cdot \phi(B_i) - w_i^A \cdot \phi(A_i) + p \cdot \left(w_i^B - w_i^A \right) \right)^2} \tag{13.12}$$

Exemplo 5

Seja o grafo $G(\mathcal{V}, \mathcal{A}, \mathcal{B})$, com $|\mathcal{A}| = 10$, $|\mathcal{V}| = 19$ e $\mathcal{B} = \{1\}$, apresentado na Fig. 13.3 e na Tab. 13.2. Busca-se nesse grafo redes com topologia de árvore, assumindo o vértice 1 como raiz.

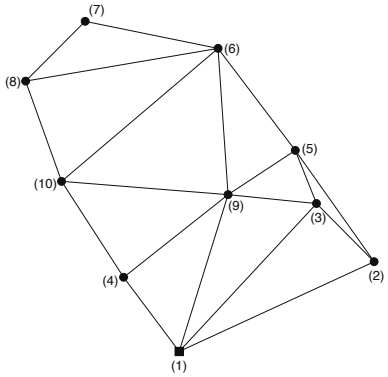


Figura 13.3: Exemplo 5: Grafo de busca com 10 vértices e 19 arestas.

Tabela 13.2: Exemplo 5: Componentes do espaço de arestas possíveis

Componente	$\overrightarrow{e_1}$	$\overrightarrow{e_2}$	$\overrightarrow{e_3}$	$\overrightarrow{e_4}$	$\overrightarrow{e_5}$	$\overrightarrow{e_6}$	$\overrightarrow{e_7}$	$\overrightarrow{e_8}$	$\overrightarrow{e_9}$	$\overrightarrow{e_{10}}$
do vértice:	1	1	1	1	2	2	3	3	4	4
para o vértice:	2	3	4	9	3	5	5	9	9	10

Componente	$\overrightarrow{e_{11}}$	$\overrightarrow{e_{12}}$	$\overrightarrow{e_{13}}$	$\overrightarrow{e_{14}}$	$\overrightarrow{e_{15}}$	$\overrightarrow{e_{16}}$	$\overrightarrow{e_{17}}$	$\overrightarrow{e_{18}}$	$\overrightarrow{e_{19}}$
do vértice:	5	5	6	6	6	6	7	8	9
para o vértice:	6	9	7	8	9	10	8	10	10

As Figs. 13.4a e 13.4b representam duas árvores X_1 e X_2, possíveis em G, que foram geradas aleatoriamente. Os pesos dos vértices nessas redes foram calculados de forma inversamente proporcional à soma dos custos das arestas no caminho entre o vértice e raiz, conforme definido em (13.13).

$$s_x^N = 1 - \frac{{d_{x,root}}^N}{\max_j \left({d_{j,root}}^N\right)} \tag{13.13}$$

Onde:
${d_{x,root}}^N$ é o custo somado das arestas no caminho entre a aresta x e a raiz *root* na árvore N.

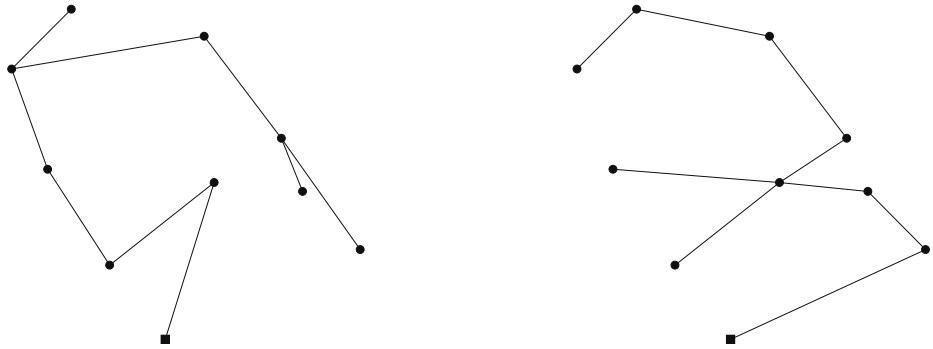
Com base nessa equação é possível encontrar o peso de todas os vértices em X_1 e X_2:

$$s^{X_1} = [\,1,00 \;\; 0,00 \;\; 0,08 \;\; 0,69 \;\; 0,14 \;\; 0,28 \;\; 0,37 \;\; 0,45 \;\; 0,82 \;\; 0,57\,]$$

$$s^{X_2} = [\,1,00 \;\; 0,74 \;\; 0,64 \;\; 0,37 \;\; 0,44 \;\; 0,27 \;\; 0,11 \;\; 0,00 \;\; 0,54 \;\; 0,35\,]$$

O peso das arestas é calculado com base nos pesos dos nós, como mostra a Eq. (13.14):

$$w_i^N = \frac{s_a^N + s_b^N}{2} \tag{13.14}$$



(a) X_1 (b) X_2

Figura 13.4: Exemplo 5: Árvores aleatórias.

O pesos dos nós s^{X_1} e s^{X_2} são inseridos em (13.14), o que leva ao peso topológico das arestas em X_1 e X_2:

$$\overrightarrow{w^{X_1}} = 0,91\overrightarrow{e_4} + 0,07\overrightarrow{e_6} + 0,11\overrightarrow{e_7} + 0,75\overrightarrow{e_9} + 0,63\overrightarrow{e_{10}} + 0,21\overrightarrow{e_{11}} + 0,36\overrightarrow{e_{14}} + 0,41\overrightarrow{e_{17}} + 0,51\overrightarrow{e_{18}}$$

$$\overrightarrow{w^{X_2}} = 0,87\overrightarrow{e_1} + 0,69\overrightarrow{e_5} + 0,59\overrightarrow{e_8} + 0,45\overrightarrow{e_9} + 0,35\overrightarrow{e_{11}} + 0,49\overrightarrow{e_{12}} + 0,19\overrightarrow{e_{13}} + 0,05\overrightarrow{e_{17}} + 0,44\overrightarrow{e_{19}}$$

Como o caso considerado é *mono-branch*, estes pesos são então utilizados na Eq. (13.8) para encontrar os vetores $\overrightarrow{X_1}$ e $\overrightarrow{X_2}$, que representam as redes X_1 e X_2 respectivamente:

$$\overrightarrow{X_1} = 2,73\overrightarrow{e_4} + 0,21\overrightarrow{e_6} + 0,34\overrightarrow{e_7} + 2,26\overrightarrow{e_9} + 1,88\overrightarrow{e_{10}} + l0,63\overrightarrow{e_{11}} + 1,09\overrightarrow{e_{14}} + 1,22\overrightarrow{e_{17}} + 1,53\overrightarrow{e_{18}}$$

$$\overrightarrow{X_2} = 2,62\overrightarrow{e_1} + 2,07\overrightarrow{e_5} + 1,76\overrightarrow{e_8} + 1,36\overrightarrow{e_9} + 1,05\overrightarrow{e_{11}} + 1,46\overrightarrow{e_{12}} + 0,56\overrightarrow{e_{13}} + 0,16\overrightarrow{e_{17}} + 1,32\overrightarrow{e_{19}}$$

Com os vetores $\overrightarrow{X_1}$ e $\overrightarrow{X_2}$ é possível calcular a posição relativa da rede X_1 em relação à X_2, usando (13.9):

$$\begin{aligned}\overrightarrow{(X_2, X_1)} \;=\; & 2,62\overrightarrow{e_1} + 2,73\overrightarrow{e_4} - 2,07\overrightarrow{e_5} + 0,21\overrightarrow{e_6} + 0,34\overrightarrow{e_7} - \\ & - 1,76\overrightarrow{e_8} + 0,90\overrightarrow{e_9} + 1,88\overrightarrow{e_{10}} - 0,42\overrightarrow{e_{11}} - 1,46\overrightarrow{e_{12}} - \\ & - 0,56\overrightarrow{e_{13}} + 1,09\overrightarrow{e_{14}} + 1,06\overrightarrow{e_{17}} + 1,53\overrightarrow{e_{18}} - 1,32\overrightarrow{e_{19}}\end{aligned}$$

Finalmente, o número escalar que define a distância entre X_1 e X_2 pode ser estimado calculado a norma Euclideana de $\overrightarrow{(X_2, X_1)}$, conforme definido em (13.12):[4]

$$D_{TN}(A,B) = \sqrt{\sum_{i=1}^{19} \left(\overrightarrow{(X_2, X_1)}_i\right)^2} = 5.93$$

Em (Carrano, 2007; Carrano et al., 2010) é demonstrado que o produto escalar e a norma definidos obedecem todos os axiomas de produtos escalares e métricas de distância respectivamente. Também é mostrado que o processo de transformação de redes proposto permite a extensão de entidades inerentes a espaços contínuos, como por exemplo, direções, distâncias, vizinhanças, projeções, etc., que ficam definidas para redes em geral. Mais detalhes sobre como essa metodologia é explorada para construção de operadores evolucionários são apresentados na Sec. 4.

[4] Exemplo adaptado de (Carrano et al., 2010).

3. Operadores Topológicos / Geométricos

A referência (Moraglio e Poli, 2004) constitui o marco inicial de um estudo que objetiva esclarecer as relações entre representações, operadores genéticos, estruturas de vizinhança e métricas de distância em algoritmos evolutivos. Com base nesse estudo é proposta uma nova classe de operadores, chamados de operadores topológicos e re-batizados de operadores geométricos em referências subsequentes. São propostos dois operadores, ε-mutação topológica uniforme e cruzamento topológico uniforme, que se enquadram nesta classe. Os pontos mais relevantes deste trabalho são discutidos na sequência dessa seção.

Operadores Topológicos

Antes de apresentar os operadores topológicos é importante esclarecer algumas definições de operadores evolucionários:[5]

Um operador evolucionário g-ário[6] OP utiliza g soluções pais, $p_1, p_2, \ldots, p_g$ para construir uma solução filha c, de acordo com alguma função de distribuição de probabilidade condicional dada:

$$Pr\{OP(p_1, \ldots, p_g) = c\} = f_{OP}(c|p_1, \ldots, p_g) \tag{13.15}$$

onde:
$Pr\{x\}$ é a probabilidade de ocorrência do evento x.

Definio 13.2 *O conjunto imagem de um operador OP é o conjunto de todos os possíveis filhos que podem ser obtidos por OP, quando os pais são $p_1, p_2, \ldots, p_g$, com probabilidade não nula:*

$$Im[OP(p_1, \ldots, p_g)] = \{c \in \mathcal{S} | f_{OP}(c|p_1, \ldots, p_g) > 0\} \tag{13.16}$$

Definio 13.3 *Um operador unário M é uma ε-mutação topológicamutação!topológica se, e somente se, $Im[M(p)] \subseteq B(p, \varepsilon)$, onde ε é o menor real possível para o qual essa afirmação é verdadeira e $B(p, \varepsilon)$ é uma vizinhança de tamanho ε, definida por uma métrica de distância D considerada.*

Definio 13.4 *Um operador binário CX é um cruzamento topológico se, e somente se, $Im[CX(p_1, p_2)] \subseteq [p_1; p_2]$. Isto significa que em um cruzamento topológico a solução filha se encontra necessariamente entre os indivíduos pais, tendo em conta uma métrica de distância D considerada.*

Tendo em conta essas definições se torna possível estabelecer dois operadores topológicos:

Definio 13.5 (ε-mutação topológica uniforme) *A ε-mutação topológica uniforme $UM\varepsilon$ é uma ε-mutação topológica onde todo filho z que se encontra a uma distância ε do pai p tem a mesma probabilidade de ser encontrado:*

$$f_{UM\varepsilon}(z|p) = Pr\{UM\varepsilon = z|P = p\} = \frac{\zeta(z \in B(p, \varepsilon))}{|B(p, \varepsilon)|} \tag{13.17}$$

$$Im[UM\varepsilon(x)] = \{z \in \mathcal{S} | f_{UM\varepsilon}(z|p) > 0\} = B(p, \varepsilon) \tag{13.18}$$

onde:
ζ é um função que retorna 1 se o argumento é verdadeiro ou 0 caso contrário.

[5] Definições extraídas de (Moraglio e Poli, 2004).
[6] A mutação é sempre um operador unário, enquanto o cruzamento é geralmente um operador binário.

Definio 13.6 (Cruzamento topológico uniforme) *O cruzamento topológico uniforme UX é um cruzamento topológico onde todo filho z que se encontra entre os pais p_1 e p_2 tem a mesma probabilidade de ser encontrado:*

$$f_{UX}(z|p_1,p_2) = Pr\{UX = z|P1 = p_1, P2 = p_2\} = \frac{\zeta(z \in [p_1;p_2])}{|\,[p_1;p_2]\,|} \tag{13.19}$$

$$Im[UM(p_1,p_2)] = \{z \in \mathcal{S}|f_{UX}(z|p_1,p_2) > 0\} = [p_1;p_2] \tag{13.20}$$

Uma característica importante da mutação topológica uniforme e o cruzamento topológico uniforme é que os mesmos são operadores definidos sem qualquer referência à representação adotada, e portanto são extensíveis à qualquer tipo de representação.

(Moraglio e Poli, 2004) demonstram que o espaço de configurações $\mathcal{C}$ pode ser definido de forma equivalente pelo par formado por $\mathcal{G}$ com qualquer uma das seguintes entidades: *(a)* A função de vizinhança Nhd; *(b)* O grafo de vizinhança $W(V,E)$; *(c)* A função de distânciadistância!função de d; *(d)* A mutação topológica uniforme UM; e, *(e)* O cruzamento topológico uniforme UX.

Isso torna possível a definição de um espaço métrico a partir da definição apenas da representação, que define $\mathcal{G}$, e um operador de mutação a de P_2 ou remover uma aresta não existente em P_2 do grafo atual (inicialmente P_1). Este método descreve um caminho que conecta as duas soluções, e se assemelha ao procedimento de *path-relinking* (Glover et al., 2000).

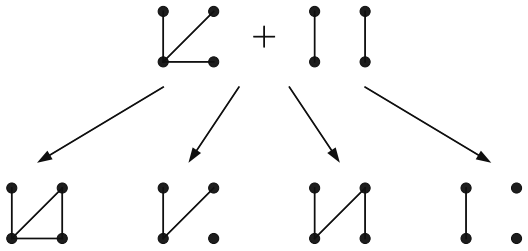


Figura 13.5: Exemplo de operador de cruzamento topológico, definido a partir de ε-mutações topológicas

A proposta dos operadores topológicos é extremamente útil na construção de algoritmos evolucionários para problemas discretos, uma vez que a mesma cria uma *framework* geral de desenvolvimento de algoritmos e permite a construção dos mesmos tendo como base apenas uma representação e um operador de mutação, ou uma representação e uma medida de distância. Segundo os autores, este *framework* traz avanços no estudo da unificação da teoria de operadores evolucionários:

Generalização: A mutação topológica e o cruzamento topológico estabelecem procedimentos gerais para construção de operadores de mutação e cruzamento, baseados em simples operações de modificação unitária.

Unificação: Estudos indicam que vários operadores propostos para diferentes representações obedecem as propriedades de operadores topológicos.

Independência de representação: Todas as definições apresentadas são independentes da representação evolucionária adotada. Isso constitui um avanço em relação à estudos anteriores, que geralmente desenvolvem operadores e ferramentas que são aplicados somente à representações específicas. O *framework* desenvolvido pode ser estendido para qualquer representação, desde que haja um operador de mutação, estrutura de vizinhança ou medida de distância definida.

Clarificação: As conexões entre representações, operadores, vizinhanças e distâncias se tornam mais claras com o uso de operadores topológicos.

Análise: Dado uma representação e um conjunto de operadores existentes, fica fácil analisar se este conjunto se enquadra nas definições topológicas. Caso sim, as propriedades conhecidas dos operadores topológicos são válidas para os operadores analisados.

É importante salientar que, mesmo com o uso de operadores topológicos, o desempenho de um algoritmo evolutivo continua estritamente relacionado com a escolha do conjunto representação / medida de distância adotada (Carrano, Fonseca, Takahashi, Pimenta e Neto, 2007). Isso faz com que, mesmo que sejam utilizados operadores topológicos, ainda seja necessário um estudo das representações e operadores mais adequados para o problema.

Aplicações de Operadores Topológicos / Geométricos

Vários estudos foram desenvolvidos utilizando os conceitos estabelecidos para os operadores topológicos (Moraglio e Poli, 2004). Nestes trabalhos foi priorizada a análise de operadores de cruzamento, que foram re-batizados de cruzamentos geométricos. A escolha pelo estudo dos operadores de cruzamento é justificada pelos autores pelo fato desses operadores constituírem um ponto crítico no desenvolvimento de AG's. Uma breve descrição das principais referências que utilizam essa classe de operadores é apresentada na sequência:

- (Moraglio e Poli, 2004) apresentam os primeiros avanços no estudo de operadores geométricos. São apresentadas as definições iniciais sobre as quais os operadores geométricos estão estabelecidos (vide o início dessa sessão). É apresentada uma análise sobre os operadores binários clássicos e é mostrado que os mesmos são operadores topológicos sob a distância de Hammingdistância!de Hamming.
- (Moraglio e Poli, 2005) utilizam três métricas específicas para permutações para construir três cruzamento geométricos aplicáveis à problemas de otimização de permutações.
- (Moraglio et al., 2006) propõem um cruzamento geométrico aplicável ao problema de comparação de cadeias de DNA. Este operador se baseia em uma medida de comparação de cadeias de caracteres para gerar a orientação do espaço métrico gerado para o problema.
- (Moraglio et al., 2007) apresentam três cruzamentos para o problema de particionamento de grafos. Estes operadores são baseados na distância *Labeling-Independent* e na distância de Hamming (vide Sec. 2).
- (Moraglio e Togelius, 2009) propõem um algoritmo *Particle Swarm Optimization* (PSO) com inércia utilizando os princípios de operadores geométricos. Este algoritmo foi comparado com três outros na solução de problemas clássicos de otimização.

4. Operações Contínuas Generalizadas para Otimização de Redes em Árvore

O procedimento de transformação de redes em vetores embutidos no espaço contínuo (Sec. 2) é utilizado em (Carrano, 2007; Carrano et al., 2010) para a construção de várias operações aplicadas à problemas de otimização de árvores. Estas operações são extensões de operações contínuas, que só são possíveis devido a metodologia proposta em (Carrano, 2007; Carrano et al., 2010). As mesmas são brevemente descritas ao longo dessa sessão.

Interpolação em Linha

O conceito de posição relativa apresentado na Eq. (13.9) pode ser usado para definir uma "interpolação em linha" para duas redes em árvore. Sejam S e D duas árvores definidas em um grafo de busca $G(\mathcal{V}, \mathcal{A}, \mathcal{B})$ (S é a rede de partida e D é a rede direção). É possível gerar redes correspondentes à pontos sobre uma "trajetória aproximada", que começa em S e termina em D, conforme descrito abaixo:

O problema é definido por:
Dadas as redes de partida e destino S e D, um escalar $\gamma \in [0,1]$ e uma tolerância $\epsilon > 0$ busca-se uma árvore R, associada ao vetor $\overrightarrow{R}$ que é aproximadamente situada na posição espacial $\overrightarrow{P} = \overrightarrow{S} + \gamma \cdot (\overrightarrow{S,D})$, tal que: $\left\| \overrightarrow{(R,P)} \right\| \leq \epsilon$.

Tal rede pode ser gerada através do seguinte procedimento:

1. Definir o contador $j = 1$, e a rede resultante atual $R_j = S$;
2. Encontrar uma nova rede R_{j+1} que é obtida a partir da remoção de uma aresta de S e a inserção de uma aresta de D em R_j, de forma que $\left\| \overrightarrow{(R_{j+1},P)} \right\| < \left\| \overrightarrow{(R_j,P)} \right\|$ e R_{j+1} mantenha factibilidade;
3. Se $\left\| \overrightarrow{(R_{j+1},P)} \right\| < \epsilon$, ir para o passo 4. Caso contrário, fazer $j = j + 1$ e ir para o passo 2;
4. Se o problema é *mono-branch*, fazer $R = R_{j+1}$ e parar; caso contrário, vá para o passo 5;
5. Se o problema é *multi-branch*, encontrar uma rede R_{j+2} que tem a mesma topologia de R_{j+1}, mas tem tipos de conexões tais que $\left\| \overrightarrow{(R_{j+2},P)} \right\|$ atinge o mínimo valor possível, com a topologia da rede fixa e os tipos de conexões variáveis. Fazer $R = R_{j+2}$ e parar.

A ideia por trás deste processo é fazer com que R caminhe ao longo do segmento $\overrightarrow{(S,D)}$, de forma que as componentes do vetor $\overrightarrow{(R,D)}$ decresçam progressivamente, enquanto que as componentes do vetor $\overrightarrow{(S,R)}$ cresçam progressivamente, e a soma das normas destes vetores seja próxima a norma de $\overrightarrow{(S,D)}$. Este processo iterativo é necessário para garantir que R_j caminhe sobre o conjunto de soluções factíveis entre S e D. Após a definição da topologia de R (após o passo 3), o ajuste dos tipos de conexões em problemas *multi-branch* é direto, e pode ser realizado de forma não-iterativa.

Este procedimento faz com que R possua apenas arestas do conjunto $S \cup D$. Isso faz desse procedimento um operador de cruzamento geométrico, conforme descrito na Sec. 3. No entanto, o mesmo ainda apresenta evoluções em relação ao cruzamento geométrico básico:

- O peso topológico da aresta w_i^N presente na *T-norma* faz com que diferentes perturbações unitárias[7] tenham diferentes impactos no valor calculado da distância. Com isso, diferentes escolhas de arestas a serem removidas/inseridas podem levar a soluções que distanciam ou aproximam de P, que é o vetor ideal que se encontra sobre o segmento $\overrightarrow{(S,D)}$ para o escalar γ. Da forma com que está proposta, a interpolação em linha busca encontrar soluções que se aproximem o máximo possível de P, o que faz que essa interpolação se aproxime muito da interpolação de vetores no espaço $\mathbb{R}^n$, cujas propriedades são conhecidas e favoráveis para a otimização.

- O fato dessa interpolação trabalhar de forma semelhante à interpolação contínua permite a extensão de outras ferramentas que dependem de interpolações, como por exemplo buscas unidimensionais, que serão apresentadas na próxima seção.

[7] Uma perturbação unitária pode ser compreendida como a remoção de uma aresta de S e inclusão de uma aresta de D em R_j

- Uma vez que a distância medida entre as soluções é continua, a interpolação permite uma definição mais adequada do parâmetro γ, que pode assumir uma conotação aproximadamente contínua (dois valores próximos de γ tendem a conduzir a duas árvores levemente diferentes). Isso não seria possível se o algoritmo se orientasse pela distância de Hamming, uma vez que a mesma tem natureza absolutamente discreta.

Busca Unidimensional

A operação de interpolação descrita torna possível a implementação de métodos otimização unidimensional. Esse procedimento pode ser implementado utilizando o algoritmo de seção áurea conforme descrito abaixo:

O problema é definido por:
Dadas as redes de partida de destino S e D e uma tolerância $\epsilon > 0$, busca-se uma árvore R, associada ao vetor $\overrightarrow{R}$ que é aproximadamente situada na posição espacial $\overrightarrow{P} = \overrightarrow{S} + \alpha^* \cdot (\overrightarrow{S,D})$, tal que: $\left\|(\overrightarrow{R,P})\right\| \leq \epsilon$. α^* é determinado otimizando a função $f(\cdot)$ a partir de S no segmento $(\overrightarrow{S,D})$: $\alpha^* = \arg\min_{\alpha} f\left(\left\lceil \overrightarrow{S} + \alpha \cdot (\overrightarrow{S,D}) \right\rceil\right)$, onde $\left\lceil \overrightarrow{X} \right\rceil$ é a árvore cujo vetor associado é o mais próximo possível do vetor $\overrightarrow{X}$.

Tal rede pode ser gerada através do seguinte procedimento:

1. Definir $a = 0$, $b = 1$, $x_a = 0,382$ e $x_b = 0,618$;
2. Encontrar uma árvore R_a que é aproximadamente situada na posição espacial $\overrightarrow{P} = \overrightarrow{S} + x_a \cdot (\overrightarrow{S,D})$;
3. Encontrar uma árvore R_b que é aproximadamente situada na posição espacial $\overrightarrow{P} = \overrightarrow{S} + x_b \cdot (\overrightarrow{S,D})$;
4. Encontrar $f_a = f(R_a)$ e $f_b = f(R_b)$;
5. Enquanto $(b - a) > \epsilon$:
   (a) Se $f_a < f_b$:
      i. Fazer $b = x_b$, $x_b = x_a$, $x_a = b - 0,618 \cdot (b - a)$, $f_b = f_a$;
      ii. Encontrar uma árvore R_a que é aproximadamente situada na posição espacial $\overrightarrow{P} = \overrightarrow{S} + x_a \cdot (\overrightarrow{S,D})$;
      iii. Encontrar $f_a = f(R_a)$.
   (b) Caso contrário:
      i. Fazer $a = x_a$, $x_a = x_b$, $x_b = a + 0,618 \cdot (b - a)$, $f_a = f_b$;
      ii. Encontrar uma árvore R_b que é aproximadamente situada na posição espacial $\overrightarrow{P} = \overrightarrow{S} + x_b \cdot (\overrightarrow{S,D})$;
      iii. Encontrar $f_b = f(R_b)$.
6. Se $f_a \leq f_b$, faça $R = R_a$; caso contrário, faça $R = R_b$.

Assim como no espaço de variáveis contínuas, esta otimização unidimensional garante encontrar a rede ótima R no segmento que une S à D, dado que a função objetivo $f(R)$ é unimodal neste segmento. No caso de comportamento não-unimodal, os limites unimodais máximo e mínimo definem os limites de erro na minimização.

Geração de Pontos Aleatórias a Distâncias Pré-definidas

A geração aleatória de um ponto que se encontra a uma distância pré-definida de um dado ponto é uma operação fundamental em vários algoritmos estocásticos. Esta operação pode ser realizada utilizando os conceitos propostos conforme descrito abaixo:

O problema é definido por:
Dada a rede inicial S e um escalar γ, que define a distância desejada, busca-se uma árvore R cuja distância em relação à S seja tão próximo quanto possível de γ: $\left\|\overrightarrow{(R,P)}\right\| \approx \gamma$.

Tal rede pode ser gerada através do seguinte procedimento:

1. Definir $\overrightarrow{R_x} = \overrightarrow{S}$;
2. Determinar $\mathcal{O}$, que é o conjunto de arestas que podem ser removidas de R_x tal que a rede resultante (após ser corrigida para manutenção da factibilidade) cumpra $\left\|\overrightarrow{(R_x,S)}\right\| \leq \gamma$;
3. Se $|\mathcal{O}| > 0$, ir para o passo 4; caso contrário ir para o passo 7;
4. Escolher aleatoriamente uma conexão $o \in \mathcal{O}$ e remover de R_x;
5. Determinar $\mathcal{I}$, que é o conjunto de conexões que re-estabelecem a factibilidade de R_x e cumprem $\left\|\overrightarrow{(R_x,S)}\right\| \leq \gamma$;
6. Escolher aleatoriamente uma conexão $i \in \mathcal{I}$ e inserir em R_x;
7. Fazer $R = R_x$;
8. Se o problema é *mono-branch*, parar; caso contrário, vá para o passo 9;
9. Determinar o conjunto de conexões de R que podem ter seus tipos mudados, de forma que a rede resultante tenha o menor valor possível para $\left|\left\|\overrightarrow{(R,S)}\right\| - \gamma\right|$. Realizar estas alterações.

Este procedimento permite a definição de um operador de mutação, que é capaz e realizar não só buscas locais, mas também buscas locais quando necessário (em algoritmos cujo raio de mutação é inversamente proporcional à sua aptidão por exemplo). A utilização da *T-norma* também permite relacionar de forma mais adequada a remoção de uma aresta com seu impacto na função objetivo.

Vizinhança e Busca Local

O conceito de distância proporcionado pela *T-norma* induz imediatamente uma estrutura de vizinhança, que pode ser definida como:

$$\mathcal{V}\left(\overrightarrow{X_0},\epsilon\right) = \left\{X \;\middle|\; \left\|\overrightarrow{(X_0,X)}\right\| < \epsilon\right\} \tag{13.21}$$

em torno do vetor $\overrightarrow{X_0}$ que representa a árvore X_0, com raio ϵ. Neste caso a busca local fica imediatamente definida como a busca dentro de uma dada vizinhança.

Vários métodos de busca local podem ser propostos para o espaço vetorial onde as redes estão embutidas. Uma possível estratégia seria:

O problema é definido por:
Dada a rede inicial S e um escalar γ, que define o raio de vizinhança, busca-se uma árvore R que representa o mínimo local na região em torno de S.

Tal rede pode ser gerada através do seguinte procedimento:

1. Fazer $R = S$ e $f(R) = f(S)$;

2. Determinar $\mathcal{O}$, que é o conjunto de arestas que podem ser removidas de R tal que a rede resultante R_x (após ser corrigida para manutenção da factibilidade) cumpra $\left\|\overrightarrow{(R_x, S)}\right\| \leq \gamma$;
3. Para cada $o \in \mathcal{O}$:
   (a) Fazer $R_x = R$;
   (b) Remover a aresta o de R_x;
   (c) Determinar $\mathcal{I}$, que é o conjunto de conexões que re-estabelecem a factibilidade de R_x e mantém $\left\|\overrightarrow{(R_x, S)}\right\| \leq \gamma$;
   (d) Para cada $i \in \mathcal{I}$:
      i. Inserir a aresta i em R_x e encontrar $f(R_x)$;
      ii. Se $f(R_x) < f(R)$:
         A. Fazer $R = R_x$ e $f(R_x) = f(R)$.
      iii. Remover i de $\mathcal{I}$.
   (e) Remover o de $\mathcal{O}$.
4. Se o problema é *mono-branch*, parar; caso contrário, ir para o passo 05;
5. Determinar $\mathcal{M}$, que é o conjunto de arestas que podem ter seus tipos modificados em R tal que a rede resultante R_x cumpra $\|\overrightarrow{(R_x, S)}\| \leq \gamma$;
6. Para cada $m \in \mathcal{M}$:
   (a) Para cada $b \in \mathcal{B}$:
      i. Fazer $R_x = R$;
      ii. Substituir a aresta m em R_x por uma aresta do tipo b e encontrar $f(R_x)$;
      iii. Se $f(R_x) < f(R)$:
         A. Fazer $R = R_x$ e $f(R_x) = f(R)$.
   (b) Remover m de $\mathcal{M}$.

Este procedimento, apesar de garantir a obtenção do ótimo local na vizinhança de S, é muito caro computacionalmente, e pode se tornar inviável mesmo em problemas com um número moderado de arestas. Este custo pode ser reduzido utilizando uma busca local aproximada, conforme descrito abaixo:

O problema é definido por:
Dada a rede inicial S, o raio de vizinhança γ e um número máximo de avaliações sem melhoria N_{EV}, busca-se uma árvore R que ao menos aproxima o mínimo local na região em torno de S.

Tal rede pode ser gerada através do seguinte procedimento:

1. Fazer $R = S$ e $f(R) = f(S)$;
2. Fazer $n = 0$;
3. Calcular $\rho = \mathrm{U}(0,1) \cdot \gamma$ onde $\mathrm{U}(0,1)$ é um número aleatório com distribuição uniforme no intervalo $[0,1]$;
4. Gerar uma rede R_x a partir de R, tal que $\left\|\overrightarrow{(R_x, S)}\right\| \approx \rho$ (esta rede é gerada utilizando o procedimento de geração de árvores aleatórias a distâncias pré-definidas) e encontrar $f(R_x)$;
5. Se $f(R_x) < f(R)$:
   (a) Fazer $R = R_x$ e $f(R_x) = f(R)$ e voltar ao passo 2.
6. Caso contrário, fazer $n = n + 1$;
7. Se $n \geq N_{EV}$, então parar; caso contrário, voltar ao passo 3.

Este método permite o controle do custo computacional do método de busca local, mesmo em problemas com muitos vértices. Ainda pode ser acrescentado um parâmetro que controle o número máximo de avaliações de função executado pelo método. No entanto, é importante salientar que este método não garante a obtenção do ótimo local exato, uma vez que o mesmo não percorre todos os pontos da vizinhança de S.

Como no caso contínuo, o conceito de busca local proposto tem como intenção encontrar o mínimo local da função. No espaço vetorial onde as redes estão definidas, o mínimo local fica definido como o ponto que apresenta valor mínimo de função objetivo dentro da vizinhança considerada, ao invés de uma vizinhança arbitrariamente pequena, que é geralmente considerada em problemas contínuos.

Algoritmo Genético Proposto

As operações apresentadas nesta seção foram utilizadas para criação de 5 operadores genéticos, sendo 3 de cruzamento e 2 mutação:

crv1: Dadas duas soluções pais p_1 e p_2, a operação de interpolação em linha é utilizada para gerar duas soluções filhas c_1 e c_2, para dois aleatórios uniformes $\gamma_1, \gamma_2 \in [0, 1]$.

crv2: Dadas duas soluções pais, p_1 e p_2, a operação de busca unidimensional é utilizada para gerar um filho c_1. O segundo filho c_2 é gerado utilizando a operação de interpolação em linha, considerado um aleatório uniforme $\gamma \in [0, 1]$.

crv3: Dada uma solução pai p e a melhor solução encontrada até o momento p^*, a operação de busca unidimensional é utilizada para gerar uma solução filha c.

mut1: Dada uma solução pai p e um raio de mutação γ, a operação de geração de pontos aleatórios a distância pré-definidas é utilizada para gerar uma solução filha c.

mut2: Dada uma solução pai p e um raio de vizinhança γ, a operação de busca local (com custo computacional controlado) é utilizada para gerar uma solução filha c.

Estes operadores foram divididos em duas classes: binários, que são operadores que geram duas soluções filhas (crv1 e crv2); e, unários, que são operadores que geram uma solução filha (crv3, mut1 e mut2). Os mesmos foram utilizados na construção de um algoritmo genético, denominado *GANet*, cujo esquema básico é apresentado na sequência:

inicialização:

- gerar uma população inicial factível;
- definir $p_{bin} = 0.80$ e $p_{un} = 0.45$;
- avaliar a função objetivo para cada objetivo da população inicial;
- encontrar o valor de aptidão para cada indivíduo, utilizado ranking linear.

enquanto não critério de parada

operações binárias:

- dividir a população em pares (aleatoriamente, com distribuição uniforme);
- **para** cada par de indivíduos:
  - gerar um número aleatório $0 \leq r_{bin} \leq 1$ com distribuição uniforme;
  - **se** $r_{bin} \leq p_{bin}$ **então**

    escolher aleatoriamente entre **crv1** e **crv2** e executar o operador.

operações unárias:

- **para** cada indivíduo:
  - gerar um número aleatório $0 \leq r_{un} \leq 1$ com distribuição uniforme;
  - **se** $r_{un} \leq p_{un}$ **então**
    escolher aleatoriamente entre **mut1**, **mut2** e **crv3** e executar o operador.

avaliação de função e seleção:

- avaliar a função objetivo para cada indivíduo novo gerado;
- encontrar o valor de aptidão para todos indivíduos, utilizando ranking linear;
- realizar a seleção utilizando a Amostragem Estocástica Universal (do inglês *Stochastic Universal Sampling* ou simplesmente SUS) (Baker, 1987).

end enquanto

Aplicações bem sucedidas deste algoritmo, ou variantes do mesmo, podem ser encontradas em:

- (Carrano, Guimaraes, Takahashi, Neto e Campelo, 2007), onde um Algoritmo Imunológico Artificial (que utiliza os operadores propostos) é utilizado para projetar sistemas de distribuição de energia elétrica considerado incertezas na evolução de carga;
- (Pereira et al., 2009), onde três variações do algoritmo proposto são aplicadas na solução do *Degree-Constrained Minimum Spanning Tree Problem* (DCMST);
- (Carrano et al., 2010), onde este algoritmo é aplicado na solução do *Optimal Communication Minimum Spanning Tree* (OCST) e do *Quadratic Minimum Spanning Tree* (QMST).

No presente trabalho, este algoritmo foi aplicado na solução de um problema clássicos de projeto de redes: o *Optimal Communication Spanning Tree*. O algoritmo *GANet* foi comparado com outros cinco GA's clássicos, desenvolvidos para otimização de problemas de árvores *mono-branch*. Os resultados observados são apresentados na Sec. 5.

5. Estudo de Caso - Optimal Communication Spanning Tree (OCST)

No problema *Optimal Communication Spanning Tree*, ou OCST (Hu, 1974), busca-se à árvore geradora de mínimo custo, onde este custo é baseado nos requisitos de comunicação entre os pares de nós do sistema (Soak et al., 2006). Este problema foi inicialmente provado como NP-difícil (NP-hard) (Garey e Johnson, 1979) e posteriormente como MAX SNP-difícil (MAX SNP-hard) (Papadimitriou e Yannakakis, 1991; Soak et al., 2006). A formulação do problema é apresentada abaixo.

A função objetivo é definida como:

$$\min \sum_{i,j \in \mathcal{V}} R_{i,j} \cdot C^X_{i,j} \tag{13.22}$$

onde:
$C^X_{i,j}$ é a soma dos custos das arestas no caminho $i - j$;
$R_{i,j}$ são os requisitos de comunicação entre i e j;
$\mathcal{V}$ é o conjunto de vértices.
A única restrição é a restrição topológica que exige que a rede seja uma árvore.

O algoritmo proposto na seção anterior foi utilizado para solução de instâncias euclideanas de 25 nós (300 variáveis) e 50 nós (1225 variáveis) do OCST. Essas instâncias foram geradas considerando

grafos completos, com os nós gerados aleatoriamente dentro de um quadrado 50×50. Os requisitos de comunicação entre os nós foram gerados aleatoriamente, com distribuição uniforme, no intervalo $[0, 200]$.

O *GANet* e outros cinco GA's, baseados em representações específicas para redes em árvores, foram aplicados na solução dessas instâncias:

$\mathcal{A}_1$: *GANet* (Carrano, 2007; Carrano et al., 2010);

$\mathcal{A}_2$: GA utilizando representação *Characteristic Vector* (Rothlauf, 2005);

$\mathcal{A}_3$: GA utilizando representação *Prüfer Numbers* (Prüfer, 1918);

$\mathcal{A}_4$: GA utilizando representação *Network Random Keys* (Routhlauf et al., 2002);

$\mathcal{A}_5$: GA utilizando representação *Edge Sets* (Raidl e Julstrom, 2003);

$\mathcal{A}_6$: GA utilizando representação *Node Biased* (Palmer e Kershenbaum, 1994a);

Detalhes sobre estes cinco últimos algoritmos podem ser encontrados em (Carrano, Fonseca, Takahashi, Pimenta e Neto, 2007).

Todos os algoritmos, incluindo o *GANet*, foram testados utilizando o mesmo conjunto de parâmetros:

- Número de execuções: 30 execuções por instância;
- Tamanho da população: 50 indivíduos;
- p_{crz} / p_{bin}: 0.80 por par;
- p_{mut} / p_{un}: 0.45 por indivíduo;
- Método de seleção: SUS com ranking linear ($s = 2$);
- Critério de parada: 1.500 avaliações de função sem melhoria para o melhor indivíduo da população.

Foram consideradas três critérios de mérito (C_{MER}) para avaliação dos algoritmos:

- *bst*: valor de função objetivo da melhor solução encontrada dentre 30 execuções do algoritmo;
- f^*: média ($\overline{X}$) e desvio padrão (s) da variável aleatória definida pelo valor de função objetivo da melhor solução encontrada em cada execução do algoritmo;
- n_{FE}: média ($\overline{X}$) e desvio padrão (s) da variável aleatória definida pelo número de avaliações de função gasto para atingir a melhor solução encontrada em cada execução do algoritmo.

Os resultados observados nos seis algoritmos, para as instâncias de 25 e 50 nós, são apresentados na Tab. 13.3.

Sob o ponto de vista do valor de função objetivo observado, é possível ver que $\mathcal{A}_1$ apresenta o melhor desempenho, com uma variação muito baixa entre a melhor solução obtida e a média observada. No caso de 50 nós, $\mathcal{A}_1$ foi o único a obter uma solução com valor de função objetivo abaixo de $1,115E+7$. Outro aspecto interessante nessa instância é que a média de $\mathcal{A}_1$ é melhor que a melhor execução de todos os outros algoritmos estudados.

Já sob o ponto de vista de avaliações de função é possível observar que $\mathcal{A}_1$ é mais rápido que todos os outros algoritmos para a instância de 25 nós, e ligeiramente mais lento que $\mathcal{A}_3$ e $\mathcal{A}_6$ para a instância de 50 nós. No entanto, essa menor velocidade na instância de 50 nós pode ser justificada pela maior capacidade de convergência do algoritmo: uma vez que $\mathcal{A}_1$ atinge melhores soluções que os outros algoritmos, é razoável que o mesmo gaste mais avaliações de função que eles, uma vez que melhores soluções são mais difíceis de ser alcançadas.[8]

[8] Descrição e resultados adaptados de (Carrano et al., 2010).

Tabela 13.3: OCST: Resultados obtidos

Inst.		OCST: 25 nós		OCST: 50 nós	
Alg.	C_{MER}	$\overline{X}$	s	$\overline{X}$	s
$\mathcal{A}_1$	bst	2,5620E+6	-	1,1087E+7	-
	f^*	2,5624E+6	0,0061	1,1099E+7	0,0040E+6
	n_{FE}	1,0326E+4	1,8556E+3	6,5959E+4	2,0396E+4
$\mathcal{A}_2$	bst	2,5620E+6	-	1,1415E+7	-
	f^*	2,5764E+6	0,1089	1,1926E+7	0,3523E+6
	n_{FE}	3,2167E+4	5,4320E+3	7,6265E+4	1,5854E+4
$\mathcal{A}_3$	bst	2,6360E+6	-	1,1809E+7	-
	f^*	2,8545E+6	1,9991	1,2971E+7	0,8964E+6
	n_{FE}	1,2711E+4	2,6864E+3	6,1056E+4	1,3665E+4
$\mathcal{A}_4$	bst	2,6039E+6	-	1,1782E+7	-
	f^*	2,8027E+6	1,5887	1,4197E+7	1,6379E+6
	n_{FE}	2,4872E+4	8,1407E+3	7,2492E+4	1,6789E+4
$\mathcal{A}_5$	bst	2,5620E+6	-	1,1182E+7	-
	f^*	2,5742E+6	0,0989	1,1640E+7	0,2277E+6
	n_{FE}	2,0735E+4	2,9372E+3	9,0303E+4	1,3878E+4
$\mathcal{A}_6$	bst	2,5620E+6	-	1,1322E+7	-
	f^*	2,6242E+6	0,3163	1,1653E+7	0,3070E+6
	n_{FE}	1,5632E+4	4,8127E+3	5,3659E+4	1,0203E+4

Testes de Panorama de Aptidão

O OCST foi utilizado para estudar como a *T-norma* afeta o ordenamento relativo das soluções no espaço de busca, e qual o impacto disso na função objetivo. O panorama de aptidão gerado pelo ordenamento relativo proporcionado pela *T-norma* é comparada com o panorama de aptidão observado quando as soluções são ordenadas conforme a distância de Hamming. Essa comparação é feita através do seguinte procedimento:

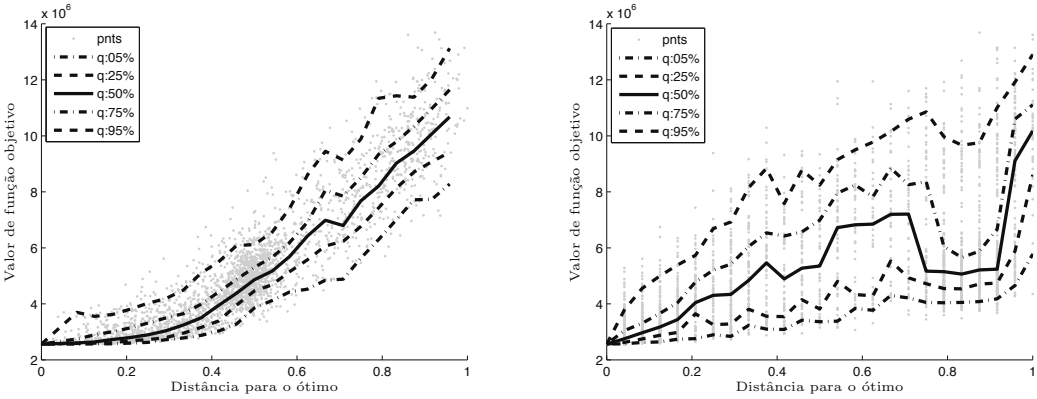
1. Gerar um conjunto de k redes factíveis aleatórias;
2. Encontrar o distância entre cada rede e a melhor solução conhecida para o problema usando a *T-norma* e a distância de Hamming;
3. Ordenar as redes em ordem crescente de distância para o ótimo;
4. Normalizar a distância para ambas as métricas;
5. Dividir o intervalo de distâncias, que estão normalizadas entre 0 e 1, em l sub-intervalos;
6. Para cada intervalo, encontrar os quantis $q_{0.05}$, $q_{0.25}$, $q_{0.50}$, $q_{0.75}$ e $q_{0.95}$ do valor de função objetivo para estimar a dispersão do conjunto de soluções.

As Figs. 13.6a e 13.6b mostram o ordenamento das soluções oferecido pela *T-norma* e a distância de Hamming respectivamente, para a instância de 25 nós do OCST. Já as Figs. 13.7a e 13.7b apresentam estes resultados para o OCST de 50 nós.

Deve-se notar que a amplitude coberta pelo valor de função objetivo em cada sub-intervalo é muito maior para a distância de Hamming que para a *T-norma*. Isso significa que, no espaço métrico induzido pela T-norma, as redes estão espalhadas em torno do ótimo seguindo um padrão próximo ao de bacias de atração circulares: o valor de função objetivo é aproximadamente o mesmo para redes que estão localizadas à uma dada distância da rede ótima em todas as direções espaciais. Por outro lado, a distância de Hamming define conjuntos de redes que são "equidistantes" para a rede ótima, e apresentação valores de função objetivo consideravelmente diferentes. Isto pode ser interpretado,

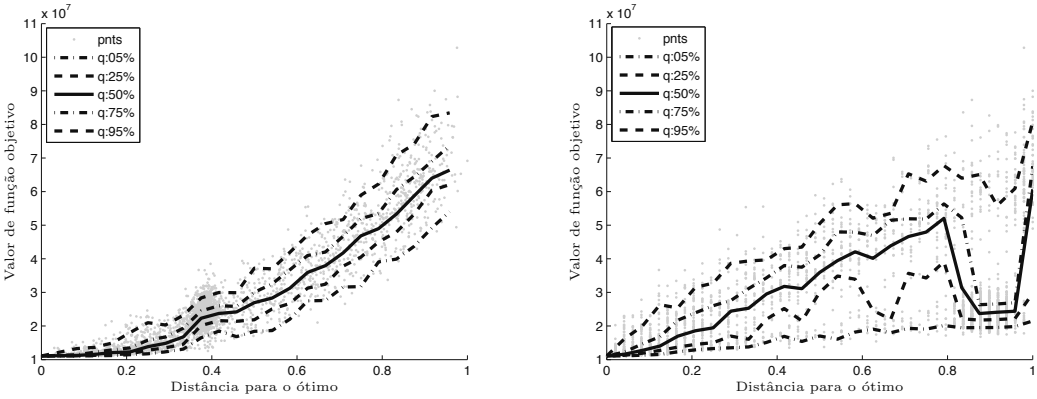
utilizando uma analogia do espaço contínuo, como "curvas de nível" da função objetivo que são consideravelmente diferentes de círculos, uma vez que um círculo cruza várias curvas de nível.

Estas observações suportam a interpretação de que a representação vetorial proposta gera coordenadas que são favoráveis para o processo de otimização.



(a) Distância *T-norma* (b) distância de Hamming

Figura 13.6: OCST (25 nós): Teste de panorama de aptidão.



(a) Distância *T-norma* (b) distância de Hamming

Figura 13.7: OCST (50 nós): Teste de panorama de aptidão.

(Página deixada propositadamente em branco)

CAPÍTULO 14

Tratamento de Restrições em Algoritmos Evolutivos

Elizabeth F. Wanner

Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Os algoritmos evolutivos (AEs) são classificados como métodos de otimização global e são considerados métodos robustos e efetivos quando se deseja encontrar um mínimo global aproximado. A natureza exploratória dos AEs permite uma identificação rápida de regiões promissoras no espaço de busca, mas, por outro lado, prejudica a convergência e precisão nos estágios finais do processo de busca. Nesses estágios finais pouca informação nova é obtida através de seus mecanismos de busca local, enquanto seus mecanismos de busca global introduzem perturbações que são muito grandes para permitir uma convergênca com alta precisão ((De Jong, 1993; Mitchell, 1996)). Os AEs não utilizam informações locais do espaço de soluções, tais como derivadas, durante o processo de busca, o que faz com que esses algoritmos apresentem uma taxa de convergência mais lenta se comparados às técnicas clássicas de busca local (veja (Jonhson e Rahmat-Samii, 1997)). Além disso, os AEs, por serem algoritmos irrestritos por natureza, possuem dificuldade de localizar o ponto de ótimo em problemas com restrições, principalmente de igualdade. Esta dificuldade é devida à inabilidade do AE em realizar uma busca em regiões de volume zero, regiões que possuem dimensão menor que o espaço de busca original.

Os AES, originalmente criados para lidar com problemas irrestritos, necessitam de mecanismos específicos para incorporar restrições à função objetivo. Inicialmente em (Michalewicz, 1995a) e posteriormente em (Michalewicz e Schoenauer, 1996a), os autores discutiram os diversos métodos de

tratamento de restrições em uma classe especial dos AES, os algoritmos genéticos. A maioria das técnicas de tratamento de restrições foi classificada em cinco categorias:

1. Métodos baseados em funções de penalidade;
2. Métodos baseados na preservação da factibilidade das soluções;
3. Métodos baseados na distinção entre soluções factíveis e infactíveis;
4. Métodos baseados em representações e operadores;
5. Métodos híbridos.

A primeira classe de métodos utiliza vários tipos de funções de penalidade para penalizar soluções que violem alguma restrição. A segunda classe de métodos usa explicitamente o conhecimento da estrutura das restrições e utiliza vários operadores de busca para manter a factibilidade das soluções. A terceira classe de métodos utiliza operadores de busca distintos para lidar com as soluções factíveis e infactíveis, enquanto a quarta classe usa um esquema de representação indireta que contêm instruções de como construir uma solução factível. Finalmente, na quinta classe, os AEs são combinados com outras heurísticas ou com outros métodos clássicos de tratamento de restrições.

Neste capítulo, algumas técnicas especializadas no tratamento de restrições, baseadas nessa classificação, serão apresentadas e discutidas. Os métodos baseados em otimização multiobjetivo também serão abordados. Para obter maiores detalhes destas e de outras técnicas especializadas no tratamento de restrição, recomenda-se a leitura dos trabalhos (Michalewicz, 1995b; Coello Coello, 2002; Venkatraman e Yen, 2005).

Em todos os métodos que serão discutidos nesse capítulo, considere o seguinte problema de otimização mono-objetivo:

$$\begin{array}{l} x^* = \min_x f(x) \\ \text{sujeito a:} \left\{ \begin{array}{ll} g_i(x) \leq 0; & i = 1, 2, \cdots, r \\ h_j(x) = 0; & j = 1, 2, \cdots, p \end{array} \right. \end{array} \tag{14.1}$$

sendo que $x \in \mathbb{R}^n$, $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$, $g(\cdot) : \mathbb{R}^n \to \mathbb{R}^r$, e $h(\cdot) : \mathbb{R}^n \to \mathbb{R}^p$. As funções g_i e h_j representam, respectivamente, funções de restrição de desigualdade e de igualdade.

Usualmente, cada restrição de igualdade do problema (14.1) deve ser relaxada e transformada em duas restrições de desigualdade da forma:

$$\begin{aligned} h_j(x) - \epsilon \leq 0 \\ -h_j(x) - \epsilon \leq 0 \end{aligned}$$

onde ϵ é um número pequeno. Desta forma, o problema (14.1) é transformado no seguinte problema mono-objetivo com apenas restrições de desigualdade:

$$\begin{array}{l} x^* = \min_x f(x) \\ \text{sujeito a:} \left\{ \begin{array}{ll} g_i(x) \leq 0; & i = 1, 2, \cdots, r \\ g_k(x) = h_j(x) - \epsilon \leq 0 & k = r+1, \cdots, r+p \\ g_j(x) = -h_j(x) - \epsilon \leq 0 & j = r+p+1, \cdots, r+2p \end{array} \right. \end{array} \tag{14.2}$$

Observe que o problema considerado é o de minimizar uma função objetivo. Entretanto, a maximização de uma função objetivo pode ser convertida em um problema de minimização através do princípio de dualidade, no qual

$$\max f(x) = \min \ -f(x)$$

.

Este capítulo está dividido da seguinte forma:

Seção 1 descreve a maneira usual de tratamento de restrições, o método de penalidade, e algumas variações deste;

Seção 2 descreve os métodos que se baseiam na premissa de que soluções factíveis são preferíveis à soluções infactíveis;

Seção 3 apresenta alguns métodos que fazem uso de representações e operadores especiais, responsáveis pela preservação da factibilidade das soluções;

Seção 4 descreve os métodos que transformam o problema mono-objetivo em um problema multi-objetivo ao tratar as restrições como funções objetivo adicionais;

Seção 5 apresenta metodologias que utilizam alguma técnica de otimização, geralmente numérica ou heurística, combinada com um algoritmo evolutivo;

Seção 6 descreve algumas técnicas de tratamento de restrições em algoritmos multiobjetivo;

Seção 7 conclui o capítulo.

1. Métodos de Penalidade

A maneira tradicional de incorporar restrições nos algoritmos evolutivos é através do método de penalidade, que transforma o problema restrito (14.1) em um problema irrestrito aproximadamente equivalente. No método de penalidade, um termo é adicionado à função objetivo penalizando qualquer violação das restrições. Este método gera uma sequência de pontos infactíveis cujo limite corresponde à solução ótima do problema original.

As restrições de desigualdade e de igualdade são incorporadas à função de aptidão $F(x)$, que passa ser definida como a soma da função-objetivo $f(x)$ e o termo de penalidade, o qual depende da violação das restrições de desigualdade $g_i(x)$ e de igualdade $h_j(x)$:

$$F(x) = f(x) + \sum_{i=1}^{r} \alpha_i [\max\{0, g_i(x)\}]^{\gamma} + \sum_{j=1}^{p} \alpha_j |h_j(x)|^p \tag{14.3}$$

onde, usualmente γ e p são iguais a 1 ou 2 O parâmetro α_i corresponde ao parâmetro de penalidade para a i-ésima restrição de desigualdade e o parâmetro α_j corresponde ao parâmetro de penalidade para a j-ésima restrição de igualdade.

Ao se utilizar o método de penalidade, o valor total das violações das restrições é usado para penalizar uma solução infactível. Assim, soluções factíveis são favorecidas durante o processo de seleção. Apesar da popularidade, o método de penalidade apresenta alguns aspectos negativos, como por exemplo a falta de regras bem definidas para se determinar os parâmetros de penalidade. Esses parâmetros são responsáveis pelo grau de penalização a ser aplicado e, para guiar a busca de modo eficiente para a região factível, devem ser bem escolhidos. Assim, existem $r + 2p$ parâmetros distintos que precisam ser escolhidos de modo eficiente nesta abordagem. Com o objetivo de se reduzir o número de parâmetros, as funções de restrições são normalizadas e apenas um valor do parâmetro α pode ser escolhido para todas as restrições do problema.

Independentemente do número de parâmetros envolvidos na formulação, a solução ótima para a função (14.3) depende da escolha desses parâmetros. A solução do problema penalizado pode estar arbitrariamente perto da solução original se valores grandes de α forem escolhidos.

Este fato pode ser observado no seguinte exemplo:

$$\begin{array}{l} x^* = \min_x x_1^2 + x_2^2 \\ \text{sujeito a: } \left\{ \; x_1 + x_2 - 1 = 0 \right. \end{array}$$

A solução ótima deste problema é o ponto $x^* = (0.5; 0.5)$ e o valor ótimo é $f(x^*) = 0.5$. Considere agora o problema penalizado irrestrito:

$$x^* = \min_x x_1^2 + x_2^2 + \alpha(x_1 + x_2 - 1)^2$$

Observe que, para qualquer valor de α, a função objetivo deste novo problema é convexa. Assim, a condição necessária e suficiente de otimalidade é dada por $\nabla(xx_1^2 + x_2^2 + \alpha(x_1 + x_2 - 1)^2) = 0$, e portanto temos que

$$x_1 + \alpha(x_1 + x_2 - 1) = 0$$
$$x_2 + \alpha(x_1 + x_2 - 1) = 0$$

Resolvendo-se essa duas equações simultaneamente obtemos

$$x_1 = x_2 = \frac{\alpha}{2\alpha + 1}$$

. Portanto, se $\alpha \to \infty$ temos que $x_i = x_2 \to 0.5$

Entretanto, escolhendo-se um valor muito grande para α a factibilidade será enfatizada e, a maior parte dos algoritmos irá convergir rapidamente para um ponto factível qualquer. E mesmo que esse ponto factível esteja longe da solução ótima, uma convergência prematura poderá ocorrer. Além disso, se a solução ótima estiver localizada na fronteira da região factível, o algoritmo será atraído para o interior da região factível e não será capaz de se mover para a fronteira entre as regiões factível e infactível. Portanto, com valores elevados para α, o algoritmo não será capaz de explorar regiões infactíveis desde o início do processo de busca. E, se existirem regiões factíveis disjuntas, o algoritmo irá mover-se para uma dessas regiões e não será capaz de explorar as outras regiões factíveis, a menos que elas estejam muito próximas uma da outra.

Por outro lado, escolhendo-se um valor muito pequeno pequeno para α, o algoritmo passará muito tempo explorando regiões infactíveis e convergirá para pontos fortemente infactíveis ((Bazaraa et al., 1979)). Este é um aspecto muito importante em AEs, já que vários problemas possuem a solução ótima na fronteira da região factível.

Existem pelo menos três escolhas que definem uma relação entre um indivíduo infactível e a região factível do espaço de busca ((Dasgupta e Michalewicz, 1997)):

1. um indivíduo deve ser penalizado apenas pela infactibilidade, sem levar-se em conta a proximidade do mesmo com a região factível;

2. o valor total das violações das restrições pode ser calculado e ser utilizado para definir um fator de penalidade;

3. o custo de tornar uma solução infactível em uma factível deve ser levado em consideração.

Com o objetivo de ajustar os parãmetros de penalidade ou mesmo de reduzir os efeitos desses, várias heurísticas para o desenvolvimento de novas funções de penalidade foram propostos. Estas heurísticas se baseiam nessas três definições dadas acima. Iremos, brevemente, fazer uma revisão de alguns desses métodos.

Penalidades Estáticas

Nessa classe de funções de penalidade, os parâmetros de penalidade não dependem da geração atual do algoritmo evolutivo e, portanto, permanecem constantes durante todo o processo evolutivo.

Em (Homaifar et al., 1994), os autores propuseram uma metodologia na qual o usuário pré-definia níveis de violação das restrições. Para cada nível, o parâmetro de penalidade era escolhido de modo

que este aumentava sempre que o nível aumentava. Considerando um problema de forma (14.2), a população inicial era composta de indivíduos factíveis e infactíveis, sendo a função de aptidão dada por:

$$F(x) = f(x) + \sum_{i=1}^{r+2p} R_{k,i} \times \max\left[0,\ g_i(x)^2\right]$$

sendo $R_{i,k}$ os parâmetros de penalidade, e $k = 1, 2, \cdots, l$ onde l representava o número de níveis de violação das restrições pré-definido pelo usuário. Essa metodologia tinha como objetivo balancear cada restrição separadamente definindo um conjunto de parâmetros de penalidade para cada restrição. Esses parâmetros eram determinados através de certas regras determinísticas pré-definidas.

Uma desvantagem desta metodologia se deve ao número elevado de parâmetros envolvidos: com $r + 2p$ restrições e l níveis de violação, são necessários $(r + 2p)(2l + 1)$ parâmetros. Além disso, é necessário se conhecer os graus de violação das restrições em cada problema, os quais não são conhecidos em qualquer problema.

Penalidades Dinâmicas

Nessa classe de métodos, a geração na qual o AE se encontra influencia o parâmetro de penalidade que será aplicado. Usualmente, o parâmetro de penalidade aumenta à medida que o número de gerações aumenta.

Em (Joines e Houck, 1994), os autores propuseram uma metodologia na qual a função de aptidão, a cada geração t, era dada por:

$$F(x) = f(x) + (C \times t)^{\alpha} \times SVC(\beta, x)$$

sendo C, α e β constantes definidas pelo usuário,

$$SVC(\beta, x) = \sum_{i=1}^{r} D_i^{\beta}(x) + \sum_{j=1}^{p} D_j(x)$$

e

$$D_i(x) = \begin{cases} 0 & g_i(x) \leq 0 \\ |g_i(x)| & g_i(x) > 0 \end{cases}$$

$$D_j(x) = \begin{cases} 0 & -\epsilon \leq h_j(x) \leq \epsilon \\ |h_j(x)| & \text{nos outros casos} \end{cases}$$

Alguns autores acreditam que os métodos de penalidade dinâmica funcionam melhor que os métodos de penalidade estática. Entretanto, na prática, é tão difícil desenvolver um método de penalidade dinâmica quanto encontrar parâmetros de penalidade que funcionem bem em métodos de penalidade estática.

Por exemplo, neste método descito acima verificou-se que a qualidade da solução final obtida era extremamente sensível à escolha dos parâmetros α e β. Além disso, ou o método convergia para soluções infactíveis, ou para uma solução factível longe da solução ótima ((Dasgupta e Michalewicz, 1997)).

Penalidades de Recozimento

Em (Michalewicz e Attia, 1994), os autores desenvolveram um método baseado na idéia do algoritmo de Recozimento Simulado ((Kirkpatrick Jr. et al., 1983)) de forma que os parâmetros de penalidade eram modificados apenas quando o algoritmo ficasse preso em um ponto de mínimo local. Apenas as restrições ativas eram consideradas a cada iteração e o parâmetro de penalidade aumentava ao longo das gerações, de modo que as soluções infactíveis eram bastante penalizadas na última geração. As restrições eram divididas em quatro grupos: restrições de igualdade lineares, restrições de desigualdade lineares, restrições de igualdade não lineares e restrições de desigualdade não lineares. Um conjunto de restrições ativa A era construído e todas as restrições não lineares de igualdade, juntamente com todas as restrições de desigualdade não lineares ativas, eram incluídas nesse conjunto.

A função de aptidão era dada por:

$$F(x) = f(x) + \frac{1}{2\tau}\sum_{i\in A}\phi_i^2(x)$$

sendo τ um valor de *temperatura* e

$$\phi_i(x) = \begin{cases} \max[o,\ g_i(x)] & 1 \le i \le r \\ |h_j(x)| & 1 \le j \le p \end{cases}$$

Um aspecto interessante dessa técnica é que a população inicial era composta de múltiplas cópias de uma única solução satisfazendo todas as restrições. A cada geração, a temperatura τ diminuia e uma nova população era criada usando a melhor solução obtida na geração anterior. Essa solução era usada como ponto inicial para gerar as cópias para a próxima população. O processo terminava quando uma temperatura pré-definida τ_f era atingida.

Este algoritmo era capaz de preservar a factibilidade de todas as restrições lineares através de operadores que levavam indivíduos factíveis em indivíduos factíveis. A cada geração o algoritmo considerava apenas as restrições ativas, o que fazia com que a pressão exercida sobre as soluções infactíveis diminuísse devido ao decréscimo da temperatura τ. Este método era bastante sensível a escolha dos valores de τ, problema similar ao que occore no algoritmo de Recozimento Simulado. Essa metodologia de lidar separadamente com as restrições lineares pode ser considerada eficiente, porém necessita de uma solução inicial factível e de operadores especiais que preservem a factibilidade.

Penalidades Adaptativas

Aqui os parâmetros de penalidade são alterados utilizando informações adquiridas durante o processo evolutivo.

Este método foi inicialmente proposto em (Bean e Alouane, 1992) e a função de aptidão era dada por:

$$F(x) = f(x) + \lambda(t)\left[\sum_{i=1}^{r} g_i^2(x) + \sum_{j=1}^{p}|h_j(x)|\right]$$

O parâmetro $\lambda(t)$ era atualizado a cada geração t, de acordo com a seguinte regra

$$\lambda(t+1) = \begin{cases} \left(\frac{1}{\beta_1}\right)\lambda(t) & b_i \in \mathcal{F} \quad \forall \ \ t-g+1 \le i \le t \\ \beta_2\lambda(t) & b_i \notin \mathcal{F} \quad \forall \ \ t-g+1 \le i \le t \\ \lambda(t) & \text{nos outros casos} \end{cases}$$

sendo b_i o melhor indivíduo da geração i, $\mathcal{F}$ o conjunto factível, $\beta_1 \neq \beta_2$ e $\beta_1,\ \beta_2 > 1$.

Neste método, o parâmetro de penalidade da próxima geração $\lambda(t+1)$ diminuia quando todos os melhores elementos nas últimas g gerações fossem factíveis, aumentava se todos os melhores elementos fossem infactíveis e permanecia o mesmo da geração anterior nos outros casos. Os valores de g, β_1 e β_2 precisam ser bem escolhidos para um bom resultado do método.

Em (Lemonge e Barbosa, 2004), os autores propuseram um método de penalidade adaptativa sem a necessidade de parâmetros externos. A adaptação utilizava informações da população corrente, tais como a média dos valores das funções objetivo e da violação de cada restrição. A função de aptidão era dada por:

$$F(x) = \begin{cases} f(x) & \text{se x for factível} \\ \\ \bar{f}(x) + \displaystyle\sum_{i=1}^{r+2p} k_i v_i(x) & \text{se x não for factível} \end{cases}$$

sendo que

$$\bar{f}(x) = \begin{cases} f(x) & \text{se } f(x) > \langle f(x) \rangle \\ \\ \langle f(x) \rangle & \text{se } f(x) \leq \langle f(x) \rangle \end{cases}$$

O valor $\langle f(x) \rangle$ representava a média dos valores das funções objetivo na população corrente. O parâmetro de penalidade k_i era definido a cada geração por

$$k_i = |\langle f(x) \rangle| \frac{\langle v_i(x) \rangle}{\displaystyle\sum_{l=1}^{r+2p} \langle v_l(x) \rangle^2}$$

sendo $\langle v_l(x) \rangle$ a média da violação da l-ésima restrição na população corrente. A idéia básica era que os valores dos parâmetros de penalidade fossem distribuídos de modo que aquelas restrições que eram mais difícieis de serem atingidas teriam um valor mais alto para o parãmetro de penalidade.

Penalidade por Morte

O método de penalidade por morte é possivelmente o método mais simples de tratar restrições. Neste método, os indivíduos infactíveis são simplesmente rejeitados. No caso de um indivíduo infactível, a função de aptidão recebe o valor zero, não sendo mais necessário o cálculo do grau de violação das restrições.

Este método funciona razoavelmente bem quando a região factível é convexa e representa uma parte considerável da região de busca total[1]. Nas outras situações este método apresenta algumas limitações. Em problemas onde a razão entre a região factível e a região de busca é pequena, e quando a população inicial só contenha soluções infactíveis, é necessário que as soluções possam ser melhoradas e não simplesmente eliminadas. Além disso, é frequente situações nas quais o algoritmo chega ao ótimo do problema mais facilmente quando este consegue cruzar a fronteira entre as regiões infactível e factível, especialmente quando a região factível for não convexa, não sendo portanto aconselhável uma eliminação de indivíduos infactíveis.

[1] Define-se a *região de busca* como a parcela do espaço de busca cujos pontos são representados pela codificação empregada no algoritmo evolutivo.

2. Métodos de Preservação da Factibilidade

Nesta classe de métodos, o algoritmo evolutivo utiliza, explicitamente, o conhecimento da estrutura das restrições e utiliza vários operadores de busca para manter a factibilidade das soluções. Esses métodos se baseiam na premissa de que soluções factíveis são preferíveis à soluções infactíveis.

O método proposto em (Michalewicz e Janikow, 1991) considerava o problema mono-objetivo (14.1), com apenas restrições lineares, e iniciava-se com uma solução factivel ou com uma população de soluções factíveis. Os operadores genéticos eram responsáveis pela preservação da factibilidade das novas soluções. No caso da mutação, quando uma determinada componente x_i do vetor x era mutada, o domínio desta componente era calculado. Este domínio era determinado levando-se em consideração as restrições lineares e as outras componente do vetor x. Assim, o novo valor de x_i era escolhido neste domínio determinado e, portanto, o novo vetor x seria sempre factível. O operador de cruzamento entre duas soluções x e y era dado por

$$ax + (1 - a)y \quad 0 \leq a \leq 1$$

Observe que, como o problema só possuia restrições lineares, era possível garantir a convexidade da região de busca e, portanto, esse operador de cruzamento sempre gerava soluções factíveis.

Este método não necessita de nenhum parâmetro adicional além dos parâmetros usuais dos AEs e pode ser generalizado para lidar com restrições não lineares desde que a região de busca seja convexa. Entretanto, não é capaz de lidar com regiões de busca não convexas, que representam a situação geral em problemas com restrições não lineares.

O método proposto em (Powell e Skolnick, 1989) incorporava uma regra heurística sugerida em (Richardson et al., 1993) que sugeria um mapeamento de soluções factíveis no intervalo $(-\infty,\ 1)$ e de soluções infactíveis no intervalo $(1,\ \infty)$, considerando sempre as soluções factíveis melhores que as infactíveis. Considerando o problema mono-objetivo (14.1), os indivíduos da população eram avaliados usando a seguinte função de aptidão:

$$F(x) = \begin{cases} f(x) & \text{se factível} \\ 1 + r\left(\sum_{i=1}^{r} g_i(x) + \sum_{j=1}^{p} h_j(x)\right) & \text{se infactível} \end{cases}$$

A função $f(x)$ era normalizada no intervalo $(-\infty,\ 1)$, as funções $g_i(x)$ e $h_j(x)$ eram normalizadas no intervalo $(1,\ \infty)$ e r era uma constante real. Observe que, nesse método, a função objetivo e o valor total das violações das restrições não eram combinados quando o indivíduo era infactível, como ocorre nos métodos de penalidade. Soluções infactíveis era comparadas baseando-se apenas na violação das restrições, enquanto que soluções factíveis eram comparadas baseando-se apenas no valor da função objetivo.

Este método apresentava algumas características importantes: enquanto nenhuma solução factível fosse obtida, o valor da função objetivo não influenciaria na função de aptidão; tendo soluções factíveis e infactíveis na população as soluções factíveis teriam um valor melhor da função de aptidão. A desvantagem desse método deve-se à falta de diversidade no mecanismo de seleção. Com o objetivo de evitar uma convergência prematura, os autores usaram um processo de seleção com classificação linear, de modo que nas gerações iniciais a convergência fosse lenta e nas gerações finais a convergência fosse forçada devido ao número de cópias presentes do melhor indivíduo.

O método proposto em (Deb, 2000) utilizava um operador de seleção, baseado em torneio, no qual duas soluções eram comparadas a cada vez, seguindo os seguintes critérios:

1. Qualquer solução factível é preferida em relação à uma solução infactível.

2. Entre duas soluções factíveis, aquela que tiver melhor valor de função objetivo será preferida.

3. Entre duas soluções infactíveis, aquela que tiver menor violação das restrições será preferida.

Usualmente os parâmetros de penalidade são necessários para definir uma violação nas restrições relacionando-as ao valor da função objetivo. Entretanto, nesse método não existia a necessidade de definir um fator de penalidade, uma vez que nas três situações descritas acima as soluções nunca eram comparadas em termos dos valores da função objetivo e das restrições simultaneamente. Na primeira situação, nem o valor da função objetivo e nem o valor das restrições eram utilizados, uma vez que uma solução factível era sempre preferida. No segundo caso, soluções eram comparadas usando apenas os valores das funções objetivo, enquanto que no terceiro caso, as soluções eram comparadas usando apenas os valores das restrições violadas.

Considerando o problema mono-objetivo (14.2) e motivado por essas observações, a seguinte função de aptidão foi proposta, na qual as soluções infactíveis eram comparadas apenas em termos das restrições violadas:

$$F(x) = \begin{cases} f(x) & \text{se } g_i(x) \leq 0, \quad i = 1, \cdots, r+2p \\ f_{pior} + \sum_{i=1}^{r+2p} g_i(x) & \text{se } g_i(x) > 0, \quad i = 1, \cdots, r+2p \end{cases}$$

sendo que f_{pior} correspondia ao valor da pior solução factível na população. Se na população não houvesse nenhuma solução factível então $f_{pior} = 0$. Observe que o valor de aptidão de uma solução infactível dependia do valor da violação das restrições e da população de soluções. Entretanto, o valor de aptidão de uma solução factível era fixo e igual ao valor da função objetivo.

Os resultados obtidos e apresentados em (Deb, 2000) podem ser considerados promissores, porém fica evidente que o método tem dificuldade em manter a diversidade na população. Para evitar uma convergência prematura, algumas técnicas de nicho combinadas com taxas maiores de mutação se fazem necessárias. Isso significa que o processo de busca foi direcionado inicialmente para procurar soluções factíveis e depois utilizou técnicas que mantinham a diversidade para se aproximar da solução ótima. As funções de partilha, envolvidas nas técnicas de nicho, são computacionalmente caras e introduzem um parãmetro extra ao algoritmo.

O conceito por trás destes dois métodos se baseia na hipótese de que soluções factíveis são melhores que soluções infactíveis, e supondo que esta hipótese seja válida, espera-se que esta técnica funcione bem. A diferença entre os dois métodos é que o segundo não necessita de um parâmetro de penalidade r. Entretanto estes métodos terão dificuldade em problemas nos quais a solução ótima está localizada na fronteira entre as regiões factível e infactível.

Em (Mezura-Montes e Coello Coello, 2005), os autores utilizaram um mecanismo de mutação auto-adaptativa em uma estratégia evolutiva do tipo $(\mu + \lambda)$ para explorar efetivamente a região de busca do problema restrito. Este algoritmo foi acoplado a um mecanismo de comparação baseado nas três regras de factibilidade descritas em (Deb, 2000) e a um mecanismo simples de diversidade. A idéia era permitir que indivíduos com baixo valor de violação de restrições e com o melhor valor de função objetivo fossem selecionados para a próxima geração. Este método, inicialmente, pretendia alcançar a região factível e posteriormente, pretendia mover pontos dentro da região factível na direção do ótimo do problema. Este método apresentou bons resultados em diversos problemas analíticos, sendo superior a alguns métodos de tratamento de restrições presentes na literatura.

3. Métodos Baseados em Representações e Decodificadores

Alguns tipos de representações especiais têm sido propostas para lidar com certos problemas nos quais uma representação genérica, por exemplo a representação binária usada tradicionalmente, não

é apropriada. Devido à mudança na representação, é necessário que os novos operadores genéticos desenvolvidos funcionem de modo similar ao operadores genéticos usados na representação tradicional.

A mudança na representação tem como objetivo simplificar a forma da região de busca e os novos operadores genéticos são usualmente utilizados para preservar a factibilidade das soluções. A principal aplicação desses métodos é em problemas nos quais a localização de uma solução factível seja extremamente difícil. A referência (Davis, 1991) apresenta vários exemplos de AEs que utilizam representações e operadores especiais para resolver problemas reais que vão desde a problemas de horários, síntese de arquitetura de redes neurais a análise de DNA.

Um exemplo desse tipo de método é conhecido como *GEnetic algorithm for Numerical Optimization for COnstrained Problems* (GENOCOP). GENOCOP, proposto por (Michalewicz, 1992), eliminava as restrições de igualdade removendo parte da região de busca, e simplificando o problema para o AE. As restrições restantes eram restrições lineares de desigualdade, que formavam um conjunto convexo de busca para o algoritmo. O método buscava encontrar, inicialmente, uma solução factível através de uma amostragem da região factível. Se após um certo número de tentativas o método falhava em encontrar uma solução factível, o usuário devia escolher um ponto inicial aleatório. A população inicial do algoritmo era composta por cópias dessa solução inicial. Os operadores genéticos de recombinação usavam combinação linear entre os indivíduos garantindo, através das propriedades de conjuntos convexos, a factibilidade dos descendentes.

Uma vez que o GENOCOP iniciava-se com um indivíduo factível, era necessário que o algoritmo (ou mesmo o usuário) fosse capaz de gerar uma solução factível em um tempo razoável. O fato de permitir apenas restrições lineares, limita a aplicação do método a regiões de busca convexas.

Uma outra alternativa pertencente a essa classe de métodos é a localização da fronteira da região factível. A idéia principal consistia em realizar a busca em áreas próximas à fronteira da região factível. Uma vez que problemas com restrições não-lineares possuem pelo menos uma restrição ativa na solução ótima, é justificável realizar uma busca na fronteira entre as regiões factível e infactível. Os componentes básicos desses tipos de métodos são

(a) mecanismo de geração de soluções factíveis iniciais e

(b) operadores genéticos especiais que sejam capazes de explorar a região factível.

Além disso, os operadores genéticos devem satisfazer mais duas condições ((Michalewicz, 1992)): os operadores de cruzamento precisam ser capazes de gerar todos os filhos entre os pais e pequenas mutações devem resultar em pequenas mudanças nos valor da função de aptidão.

Em (Michalewicz e Schoenauer, 1996b), operadores genéticos especializados foram desenvolvidos e baseados nas expressões das restrições. Supondo que a superfície de busca fosse uma superfície Riemaniana de dimensão $n-1$ no espaço $\mathbb{R}^n$ e regular [2], a distância euclidiana usual poderia ser usada como uma medida de distância nesta superfície. Os operadores, que levam pontos sobre a superfície em pontos sobre a superfície, foram baseados em curvas geodésicas (curvas geradas pela interseção da superfície com planos).

Observe que, nesta abordagem, os operadores eram dependentes da parametrização escolhida, fazendo que os operadores ficassem limitados a um certo tipo de problema. Entretanto nas situações em que os operadores foram aplicados, os resultados foram precisos e eficientes.

Nos métodos baseados em decodificadores, um cromossomo é responsável por fornecer instruções de como gerar uma solução factível. Cada decodificador impõe uma relação T entre a solução factível e a solução decodificada. Algumas condições devem ser satisfeitas ao lidar com decodificadores ((Palmer e Kershenbaum, 1994b)):

1. para cada solução factível s deve existir uma solução decodificada,

[2] uma superfície é dita regular quando o vetor gradiente, ortogonal à superfície, está definido em todos os pontos da superfície

2. cada solução decodificada S deve corresponder a uma solução factível s,

3. todas as soluções factíveis devem ser representadas pelo mesmo número de decodificadores d,

4. a transformação T deve ser rápida, e

5. a transformação T deve possuir uma característica local de que pequenas mudanças na solução decodificada S produzam pequenas mudanças na solução factível s.

Em (Koziel e Michalewicz, 1998, 1999), os autores propuseram um homomorfismo entre um cubo n-dimensional $[-1,\ 1]^n$ e uma região de busca, convexa ou não. A idéia principal consistia na transformação do problema mono-objetivo original (14.2) em um outro problema topologicamente equivalente, mas de fácil solução. Este mapeamento era responsável pela geração de soluções factíveis, não sendo necessária a aplicação de nenhum operador que mantivesse a factibilidade das soluções.

Esta metodologia mostrou-se eficiente porém alguns problemas ficaram evidentes: alto custo computacional e o homomorfismo T não possuia a característica local de levar pequenas mudanças na solução decodificada em pequenas mudanças na solução factível quando a região de busca é disjunta.

4. Métodos Baseados em Otimização Multiobjetivo

A principal idéia por trás desses métodos resume-se em re-escrever o problema mono-objetivo restrito como um problema multiobjetivo. Em primeiro lugar, cada restrição de igualdade do problema (14.1) deve ser relaxada e transformada em duas restrições de desigualdade. Desta forma, o problema mono-objetivo passa a ter $r + 2p$ restrições de desigualdade e o novo problema multiobjetivo terá $r + 2p + 1$ funções objetivo. Qualquer técnica de otimização evolutionária multiobjetivo ((Fonseca e Fleming, 1995)) pode ser aplicada ao vetor $\vec{v} = (f(x),\ f_1(x),\ f_2(x),\ \cdots,\ f_{r+2p}(x))$, no qual $f(x)$ corresponde a função objetivo original do problema (14.1) e $f_k(x),\ \ k = 1, 2, \cdots, r + 2p$ representam as restrições do problema mono-objetivo. Para uma solução ser considerada ótima x^*, é necessário que $f_k(x^*) = 0$, $k = 1, 2, \cdots, r + 2p$ e $f(x^*) \leq f(y)$ para qualquer solução factível y.

As restrições em um problema mono-objetivo restrito podem ser vistas como objetivos que precisam ser fortemente atingidos antes que a função objetivo seja minimizada. Atender um conjunto de restrições violadas é claramente um problema multiobjetivo de minimizar novas funções até que um certo valor (meta) seja atingido. O conceito de não dominância é portanto aplicável.

Em (Coello Coello e Mezura-Montes, 2002), os autores usaram um algoritmo evolutivo multiobjetivo, com seleção baseada em torneio, e seguindo as regras básicas de factibilidade descritas em (Deb, 2000) e os critérios de dominância na população do algoritmo. Nesse método, um parâmetro externo S_r foi usado para controlar a diversidade da população. Apesar de apresentar bons resultados nos problemas testados, o método apresentou dificuldades ao lidar com um aumento na dimensão do problema.

Uma heurística genérica para tratamento de restrições em algoritmos genético foi mostrada em (Venkatraman e Yen, 2005). A heurística proposta era dividida em duas fases. Na primeira fase, a função objetivo era desprezada e o problema de otimização restrito era tratado como um problema de satisfação das restrições. O algoritmo procurava minimizar a violação das restrições e, eventualente, encontrar uma solução factível. A solução com menor violação das restrições era arquivada e tratada como uma solução de elite na população. Na segunda fase, considerava-se um problema biobjetivo que minimizava a violação das restrições e o valor da função objetivo. Esta metodologia mostrou-se competitiva ao ser comparada com outras técnicas de tratamento de restrições.

Uma abordagem baseada em técnicas de otimização multiobjetivo foi proposta em (Camponogara e Talukdar, 1997). A metodologia consistia em transformar o problema mono-objetivo da forma (14.2)

em um problema biobjetivo, minimizando a função objetivo e minimizando a seguinte função:

$$\phi(x) = \sum_{i=1}^{r+2p} \max\left[0,\ g_i(x)\right]$$

Redefinindo-se o problema como um problema biobjetivo, era possível gerar soluções não dominadas em relação aos dois objetivos. Cada duas soluções encontradas, x_a e x_b, definiam uma direção de busca. Uma busca linear era feita nesta direção obtendo-se uma nova solução x que dominava x_a e x_b. Essa busca linear substituía o operador de cruzamento. Uma eliminação de metade da população era aplicada em intervalos regulares da seguinte forma: as soluções com menor valor da função de aptidão eram trocadas por soluções geradas aleatoriamente.

Esta metodologia apresenta problemas em manter a diversidade na população, uma vez que os piores indivíduos são descartados a cada geração. A utilização da busca linear provoca um aumento no custo computacional do algoritmo.

5. Métodos Híbridos

Nesta classe, são considerados as metodologias que utilizam alguma técnica de otimização, geralmente numérica ou heurística, para lidar com as restrições dentro de um algoritmo evolutivo.

Um algoritmo evolutivo híbrido que combina funções de penalidade com um método primal-dual foi proposto em (Adeli e Cheng, 1994). Este método considerava um problema mono-objetivo da forma (14.2) e baseava-se na minimização, em sequência, de métodos lagrangeanos, utilizando a seguinte função de aptidão:

$$F(x) = f(x) + \frac{1}{2}\sum_{i=1}^{r+2p} \gamma_i \left\{\left[g_i(x) + \mu_j\right]^+\right\}^2$$

sendo $\gamma_i > 0$, μ_i um parâmetro associado a i-ésima restrição e

$$\left[g_i(x) + \mu_j\right]^+ = \max\left[0, g_i(x) + \mu_j\right]$$

Os valores de μ_i eram definidos em termos da máxima violação da restrição correspondente. Um outro parâmetro adicional $\beta > 1$, definido pelo usuário, era multiplicado pelos valores antigos de μ_i e γ_i, fazendo com que a penalidade fosse aumentada com o aumento da geração. Apesar dos bons resultados apresentados por esse método em certos problemas, o método é dependente da escolhas destes parâmetros.

Em (Bernardino et al., 2009), um algoritmo genético (AG) foi hibridizado com um algoritmo baseado em Sistemas Imunes Artificiais (SIA) para lidar com restrições em problemas reais. O SIA, inspirado no príncípio de seleção clonal, foi acoplado ao AG para ajudar o algoritmo a guiar a população para a região factível. Este algoritmo híbrido foi testado em vários problemas com restrições de desigualdade e seu desempenho foi comparado com outras técnicas de tratamento de restrições, mostrando-se eficiente nos problemas testados.

Em (Araujo et al., 2009), um mecanismo de busca local foi acoplado a um algoritmo genético para resolver problemas mono-objetivo com restrições (lineares e não lineares) de desigualdade. Esse mecanismo de busca local utilizava aproximações quadráticas e lineares para as funções objetivo e de restrição. Na fase de busca local, estas aproximações definiam um problema associado no qual a função objetivo era quadrática e as funções de restrições eram quadráticas ou lineares. Esse problema associado era resolvido via uma formulação baseada em Desigualdades Lineares Matriciais ((Boyd et al., 1997)) e a solução deste problema associado era, então, introduzida na população do algoritmo

genético. O algoritmo genético acoplado a esse mecanismo de busca local foi testado com vários problemas analíticos e os resultados mostraram que a metodologia é eficiente nesta classe de problemas.

Na maior parte dos trabalhos descritos nesse capítulo, as restrições de igualdade são relaxadas e transformadas em duas restrições de desigualdade. Técnicas apropriadas para tratamento de restrições de igualdade em algoritmos evolutivos não são frequentes. Restrições de igualdade constituem um problema para os AEs, uma vez que esse tipo de restrição define um conjunto factível de dimensão menor que a região de busca do problema.

Os AEs, heurísticas cuja busca é essencialmente baseada na amostragem de toda a região de busca, têm baixa probabilidade de encontrar soluções factíveis. Sabemos que a natureza aleatória dos operadores dos AEs permite a busca por regiões ótimas (bacias de atração) no espaço global. Esses operadores fazem com que a busca torne-se *espalhada* pelo espaço, maximizando a chance de encontrar outras bacias de atração. Podemos chamar essa propriedade de *busca por volume*. Entretanto, a aleatoriedade de tais operadores é conflitante com a necessidade de uma busca em objetos bem definidos de *volume zero*, tais como os conjuntos factíveis definidos por restrições de igualdade. A propriedade de *busca por volume* irá produzir um tipo de *movimento de afastamento aleatório* em relação ao objeto factível, o que acarretará na convergência a taxas mais lentas.

Tendo em mente essas dificuldades, uma metodologia especializada no tratamento de restrição de igualdade em algoritmos genéticos foi apresentada em (Wanner et al., 2005). Considerando um problema mono-objetivo com apenas uma restrição de igualdade, é possível construir uma aproximação quadrática para a função-objetivo e para a restrição, obtendo desta forma um problema quadrático associado. A solução analítica para esse problema associado pode ser obtida, porém, de uma maneira geral, essa solução não corresponde à solução do problema original. Entretanto, se as aproximações quadráticas forem precisas, a solução do problema quadrático será uma boa aproximação para a solução do problema original.

A idéia principal desse trabalho consistia em restringir o algoritmo a encontrar soluções que estivessem no interior de um objeto que possuia a dimensão igual à do conjunto factível. Este objeto consiste em uma aproximação de segunda ordem da superfície factível. Essa solução obtida deveria então ser introduzida na população do algoritmo. Desta forma, o operador de busca local proposto pode ser dividido em duas etapas:

- a aproximação de segunda ordem deve ser iterativamente atualizada, e
- o processo evolucionário de busca deve ser feito de forma que as soluções estejam na fronteira da superfície aproximada.

Um operador que segue essas etapas era, então, incluído no algoritmo genético e desta maneira, irá se opor ao *movimento de afastamento aleatório* descrito acima. Esse operador pode ser interpretado também como um operador de *elitismo* especializado em melhorar, iteração após iteração, a estimativa do conjunto factível. O operador de elitismo convencional, que mantém um único ponto (ou um conjunto de pontos) em cada geração, não promove uma busca pelo conjunto factível, uma vez que esse conjunto é uma superfície m-dimensional e não um conjunto discreto de pontos. O operador proposto garantia uma aproximação para esse objeto no seguinte sentido: a melhor superfície de segunda ordem que aproxima o conjunto factível era mantida e melhorada à medida que o algoritmo prosseguia.

Este operador pode ser descrito da seguinte forma:

Passo 1. Construir as aproximações quadráticas para a função-objetivo e para a restrição de igualdade

Passo 2. Construir o problema quadrático associado

Passo 3. Encontrar a solução analítica do problema quadrático associado

Passo 4. Introduzir essa solução na população atual do algoritmo genético

Este operador foi acoplado a um algoritmo genético e foi testado em problemas com restrições de igualdade. Em todos os casos, o algoritmo híbrido mostrou-se eficaz, encontrando a solução do problema original de modo preciso e eficiente.

Em (Peconick et al., 2007), os autores estenderam a metodologia descrita acima para tratar várias restrições não-lineares de igualdade com algoritmos genéticos. O novo operador era composto das seguintes etapas:

Passo 1. Construir as aproximações quadráticas para a função-objetivo e para as restrições de igualdade

Passo 2. Construir o problema quadrático associado

Passo 3. Determinar as projeções sobre as superfícies quadradas aproximadas via um processo iterativo, obtendo um ponto situado sobre todas as restrições

Passo 3. Introduzir este ponto na população atual do algoritmo genético

Esta metodologia mostrou-se eficaz nos problemas testados, atingindo a solução com mais precisão e eficiência. Observe que o operador proposto apenas resolve o problema de factibilidade. Ao obter soluções factíveis, o algoritmo genético irá guiar essas soluções na direção da solução ótima.

6. Tratamento de Restrições em Problemas Multiobjetivo

Considere o seguinte problema de otimização multiobjetivo:

$$x^* = \min_x \begin{cases} f_i(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{cases} \tag{14.4}$$

$$\text{sujeito a: } \begin{cases} g_i(x) \leq 0; & i = 1, 2, \cdots, r \\ h_j(x) = 0; & j = 1, 2, \cdots, p \end{cases}$$

sendo que $x \in \mathbb{R}^n$, $f(\cdot) : \mathbb{R}^n \to \mathbb{R}^m$, $g(\cdot) : \mathbb{R}^n \to \mathbb{R}^r$, e $h(\cdot) : \mathbb{R}^n \to \mathbb{R}^p$. As funções g_i e h_j representam, respectivamente,as funções de restrição de desigualdade e de igualdade.

A maneira tradicional de tratar as restrições em problemas multiobjetivo é usar os métodos de penalidade descritos na seção 1. As restrições de desigualdade e de igualdade são incorporadas a $F_k(x)$, $k = 1, 2, \cdots, m$, e passa a ser definida como a soma da função-objetivo $f_k(x)$, $k = 1, 2, \cdots, m$ e do termo de penalidade, o qual depende da violação das restrições de desigualdade $g_i(x)$ e de igualdade $h_j(x)$:

$$F_k(x) = f_k(x) + \sum_{i=1}^{r} \alpha_i [\max\{0, g_i(x)\}]^\gamma + \sum_{j=1}^{p} \alpha_j |h_j(x)|^p \tag{14.5}$$

Em princípio, todas as estratégias de atualização dos parâmetros de penalidade descritas na seção 1 podem ser utilizados na formulação multiobjetivo. Entretanto, a maior parte dos trabalhos envolvendo problemas multiobjetivo com restrições utiliza a abordagem do método de penalidade estática ((Srinivas e Deb, 1994)). Uma vez que os termos de penalidade são adicionados a cada uma das funções objetivo, qualquer algoritmo evolutivo multiobjetivo pode ser utilizado. Espera-se que os algoritmos evolutivos sejam capazes de mover a população para o interior da região factível e encontrar boas estimativas para o conjunto Pareto-ótimo.

Em (Jiménez et al., 1999), os autores apresentaram um procedimento para tratamento de restrições de desigualdade em problemas multiobjetivo. Restrições de igualdade deviam ser relaxadas e transformadas em duas restrições de desigualdade. Este procedimento sugeria uma classificação cuidadosa das soluções factíveis e infactíveis e utilizava uma técnica de nicho para manter a diversidade. O algoritmo proposto utilizava um torneio binário baseado nas regras propostas em (Deb, 2000) e sugeria estratégias específicas para cada caso.

Este procedimento é mais sistemático do que a simples utilização de um método de penalidade, já que leva em consideração a factibilidade e a dominância das soluções. Entretanto, quando há um empate em relação à factibilidade ou à dominância, o algoritmo proposto tende a priorizar a dominância.

Em (Ray et al., 2000), uma técnica mais elaborada de tratamento de restrições foi proposta. Neste método, para construir uma nova população, eram realizadas três classificações baseadas nos critérios de não dominância: com relação aos valores das funções objetivo, com relação aos valores de violação de restrições, e com relação aos valores das funções objetivo e das violações das restrições. Uma vantagem deste método é a forma de preservar a diversidade na população: quando as soluções eram infactíveis, as soluções seriam escolhidas de acordo com o nível de dominância em relação à violação das restrições, dando uma ênfase maior às soluções que não satisfaziam o maior número de restrições. Porém, à medida que a população passava a conter apenas soluções factíveis, o algoritmo tenderia à estagnar.

Em (Deb et al., 2000), os autores modificam o princípio de dominância: antes de comparar duas soluções com relação à dominância, a factibilidade das soluções era verificada. Se uma solução era factível e a outra não, a solução factível dominava a infactível. Se as duas soluções eram infactíveis, a solução com menor violação de restrições dominava a outras. Se ambas as soluções eram factíveis, aplicava-se o princípio de dominância usual. Esse procedimento proposto não necessita de nenhum outro método de tratamento de restrições e pode ser utilizado em qualquer algoritmo evolutivo multiobjetivo.

Esta abordagem, utilizada com o NSGA-II, foi testada em um conjunto de problemas de teste ((Deb et al., 2001)), juntamente com as outras duas abordagens descritas acima ((Jiménez et al., 1999; Ray et al., 2000)) . Os resultados mostraram uma superioridade do primeiro método em relação aos outros dois.

Seguindo a mesma linha de operadores de busca local baseados em aproximações quadráticas de funções, em (Wanner et al., 2008), os autores propuseram um operador adicional capaz de encontrar estimativas Pareto-ótimas mais precisas. O novo operador usava aproximações quadráticas para as funções objetivo e de restrições, que eram construídas usando as avaliações de funções usuais do algoritmo evolutivo. A fase de busca local consistia em resolver um problema multiobjetivo quadrático, via uma escalarização usando uma formulação conhecida como *goal attainment* ((Ehrgott, 2000)). A solução desse problema era introduzida na população do algoritmo. Esta metodologia foi testada em problemas analíticos e em problemas onde a avaliação da função objetivo impunha um alto custo computacional e, em ambos os casos, os resultados indicaram que a metodologia era capaz de encontrar conjunto Pareto-ótimo de qualidade superior se comparados com um algoritmo multiobjetivo tradicional.

Apesar do grande número de trabalhos em otimização multiobjetivo, existem poucos trabalhos que lidam com tratamento de restrições nesse tipo de problema ((Coello Coello, 1999)). A maior parte dos trabalhos lidam com problemas irrestritos, usam penalidade estática ou tratam as restrições como funções objetivo adicionais.

7. Conclusões

Neste capítulo apresentamos diversas técnicas de tratamento de restrições em algoritmos evolutivos, mono e multiobjetivo. As técnicas apresentadas tratam desde os métodos de penalidade e suas variações à técnicas híbridas, combinando um algoritmo evolutivo com uma heurística ou um método numérico.

Todos os trabalhos nessa direção são, por assim dizer, inconclusivos. Mesmo que uma determinada metodologia funcione bem em uma certa classe de problemas, esta tenderá a ser inferior em outra. Dependendo do tipo de problema a ser abordado, novas técnicas precisaram ser desenvolvidas. Este é um campo vasto de pesquisa e há muito ainda a ser desenvolvido.

De um modo geral, as técnicas de tratamento de restrições devem levar em consideração os seguintes aspectos:

- serem genéricas, possibilitando pequenas modificações para lidar com o maior número de problemas;
- não necessitar de ajustes dos parâmetros, ou pelo menos minimizar esses ajustes, facilitando a utilização para o usuário;
- eficiência, não acarretando ao algoritmo um alto custo computacional na sua aplicação;
- incorporar o maior conhecimento possível sobre o domínio, melhorando assim o desempenho do algoritmo.

Uma técnica de tratamento de restrições ideal deveria incorporar todos esses apectos, entretanto sabe-se que, na prática, isso é impossível. Por exemplo, ao incorporar conhecimento sobre o domínio haverá naturalmente uma redução da generalidade da técnica ((Goldberg, 1989)). Apesar desses aspectos serem conflitantes, é necessário tê-los em mente ao propor uma técnica de tratamento de restrições. É ainda recomendável que as limitações de cada técnica proposta conhecida, facilitando a escolha frente a um determinado problema de otimização.

CAPÍTULO 15

Otimização em Ambientes Incertos

Fabrício O. de França [*] *Fernando J. Von Zuben* [**]

[*] *Centro de Matemática, Computação e Cognição*
Universidade Federal do ABC

[**] *Departamento de Engenharia de Computação e Automação Industrial*
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

A incerteza de um parâmetro, por definição, indica a faixa de dispersão de valores aceitáveis que podem ser atribuídos ao parâmetro. Logo, a presença de incerteza em processos de tomada de decisão implica na necessidade de agir sob variação dinâmica, conhecimento aproximado ou incompleto, e ausência de controle sob certas variáveis que influenciam na decisão. Este capítulo trata de algoritmos computacionais para otimização iterativa em ambientes incertos, de modo que a incerteza do ambiente implica na imprevisibilidade do resultado da otimização, caso os efeitos das incertezas sejam negligenciados. O resultado se torna imprevisível no sentido de que, quando um mesmo algoritmo é aplicado diversas vezes a um dado problema de otimização com incertezas, a qualidade da solução encontrada em cada caso pode ser muito mais variável do que ocorre naturalmente em algoritmos estocásticos. Um fator agravante aqui é que todo problema real de otimização e de interesse prático vai expressar algum grau de incerteza. Sendo assim, o propósito deste capítulo é estudar algoritmos voltados para otimização e que apresentem uma maior robustez na presença de incertezas, sendo que a computação evolutiva fornece soluções bastante competitivas para este fim. Ao longo deste capítulo, será mostrado

como identificar as incertezas em processos de otimização e como a computação evolutiva pode ser empregada para tratar desse tipo de problema.

1. Introdução

Ao longo dos demais capítulos desta obra, foi feito um esforço coordenado visando mostrar como, quando e onde aplicar computação evolutiva e também outras meta-heurísticas voltadas para a solução de problemas de interesse prático, que desafiam profissionais das mais diversas áreas. Imagina-se que o leitor deve ter se surpreendido com a diversidade de cenários de aplicação e com a flexibilidade dos algoritmos evolutivos e seus derivados e afins. No entanto, este capítulo vem mostrar que ainda há espaço para novos cenários, embora voltados apenas para processos de otimização, e apresenta desafios adicionais para esta flexibilidade dos algoritmos evolutivos.

São tratados aqui problemas de otimização em ambientes incertos, os quais são formalizados matematicamente de acordo com a origem da incerteza. As incertezas podem ter diversas origens: (*i*) ausência de acuidade em sensores de medida; (*ii*) variação temporal de parâmetros e de objetivos; e (*iii*) erros, simplificações e aproximações no processo de modelagem matemática do problema de otimização, incluindo aqui a própria definição da função-objetivo.

Negligenciar essas incertezas pode ser desastroso em otimização de processos, pois decisões equivocadas podem ser tomadas, causadas pela diferença entre a situação idealizada que guia a tomada de decisão e a situação real do processo. Com isso, os efeitos cumulativos desses potenciais equívocos, na qualidade do resultado final, tornam-se imprevisíveis.

E os algoritmos de otimização, os de computação evolutiva inclusos, mesmo em versões mais elaboradas, normalmente não se mostram capazes de lidar de forma apropriada com incertezas durante o processo de otimização. Visando maior robustez de desempenho, é necessário monitorar o ambiente, propor operadores mais robustos às incertezas e empregar recursos computacionais adicionais visando minorar os efeitos potencialmente danosos das incertezas sobre o desempenho dos algoritmos de otimização, particularmente dos algoritmos evolutivos. Em linhas gerais, a ideia é fazer o melhor uso possível da informação disponível, ciente de que o que se dispõe de informação pode não ser suficiente para auxiliar na escolha da melhor decisão a cada passo.

Portanto, este capítulo trata de ferramentas capazes de tornar os algoritmos evolutivos mais efetivos no tratamento de otimização em ambientes incertos. Para tanto, optou-se por dividir o capítulo em quatro seções principais, o que se justifica pelo fato de que os problemas incertos são geralmente classificados em quatro tipos: *problemas ruidosos*, em que há incerteza na medição do valor da função-objetivo; *problemas robustos*, em que as decisões são tomadas admitindo-se uma faixa de tolerância nos valores do vetor-solução; *problemas de aproximação de função*, quando não existem meios de se definir precisamente uma função-objetivo, ou o cálculo dela é muito custoso; *problemas variantes no tempo*, quando a superfície de otimização muda sua conformação em função do tempo, provocando deslocamentos dos pontos de ótimo. Mas antes dessas quatro seções principais, existe uma seção dedicada a descrever em linhas gerais problemas relevantes que conduzem a processos de otimização em ambientes incertos, e que são bons candidatos a serem abordados via computação evolutiva.

2. A incerteza presente em problemas reais de otimização

Em lugar de representar uma exceção, a presença de algum grau de incerteza em qualquer atividade humana é regra. Fazem parte desta regra os problemas de otimização, com incertezas que vão desde a definição do que se quer ver presente na solução até no que vai suportar a decisão a cada passo de um processo iterativo de solução.

Já foi discutido ao longo dos demais capítulos desta obra que, dentre todos os problemas que

existem para serem resolvidos pela humanidade, parte deles é computável, ou seja, admite solução empregando recursos computacionais. Será tratada aqui a computação digital e não serão consideradas outras formas de computação. Dentre os problemas computáveis, parte deles é de otimização. E dentre os problemas de otimização, parte deles é intratável computacionalmente, no sentido de que a garantia de obtenção da solução ótima via algoritmos exatos de resolução requer recursos computacionais, mais especificamente tempo de processamento e memória, além do razoável. Há problemas computáveis de otimização, presentes no dia-a-dia das pessoas, que requerem bilhões de anos de processamento computacional para serem resolvidos por algoritmos exatos (que garantem obter a solução ótima), isso empregando os computadores digitais mais poderosos já concebidos. Certamente trata-se de um tempo de processamento além do razoável.

E o que já foi visto também ao longo dos demais capítulos desta obra é que as meta-heurísticas representam alternativas de solução para esses tipos de problemas de otimização computáveis, mas intratáveis. Elas são capazes de produzir propostas de solução empregando recursos computacionais razoáveis, mas ao preço de não mais garantir três condições ao longo da otimização: (*i*) não garantem obter a solução ótima; (*ii*) não permitem antecipar o custo até a convergência; e (*iii*) não garantem sequer a convergência. Na ausência de alternativas mais poderosas, as meta-heurísticas reinam soberanas, apesar deste preço a pagar. E dentre as meta-heurísticas, encontram-se todas as metodologias apresentadas nesta obra, ao longo de seus capítulos, incluindo obviamente as técnicas de computação evolutiva.

A computação evolutiva, então, é uma meta-heurística que deve ser empregada junto a certos problemas de otimização que não admitem solução a partir de algoritmos exatos, mesmo que estes algoritmos exatos existam. Este é o cenário de onde se parte neste capítulo.

Repare que não se falou ainda, nos parágrafos iniciais desta seção, de incerteza. É hora de afirmar então que a presença de incerteza amplia ainda mais os desafios do processo de otimização junto a esses problemas computáveis, mas intratáveis. Os desafios para se chegar a uma solução já eram significativos, e com incertezas ficam ainda mais expressivos.

Mas será que problemas de otimização em ambientes incertos, com as características levantadas acima, são comuns no dia-a-dia das pessoas e das instituições? Nos parágrafos que se seguem, será apresentada uma resposta afirmativa para esta questão, a partir de exemplos descritos de forma genérica (não-matemática).

Na formulação de problemas de otimização que envolvem diversas entidades e condições a serem atendidas, como no caso de escalonamento de recursos ou tarefas, é sabido que muitos esforços devem ser investidos na definição do modelo matemático que descreve o problema. Por vezes, devido à complexidade dos processos reais envolvidos, algumas simplificações e aproximações são adotadas. Em outras circunstâncias, aspectos relevantes do processo podem ser negligenciados involuntariamente ao longo da formulação. Quando o modelo matemático não consegue retratar aspectos decisivos da realidade do processo que se pretende modelar, é possível que se obtenha a solução certa para o problema errado. Essas simplificações, aproximações e erros de modelagem podem, por sua vez, ser interpretados como fontes de incertezas, a serem devidamente incorporadas à formulação matemática do problema de otimização. Um exemplo aqui pode ser a decisão de onde posicionar, considerando todo o Território Nacional ou dentro de algum Estado da Federação, uma nova unidade de uma grande rede de hipermercados, visando maximizar o retorno do investimento no menor tempo possível. Como considerar matematicamente todos os principais aspectos de cada localidade candidata, visando estimar adequadamente o retorno de investimento? Outro exemplo vinculado ao anterior seria como definir a capacidade de atendimento deste hipermercado (tamanho das instalações, número de funcionários, etc.). Como estimar apropriadamente a demanda esperada de cada localidade candidata?

Em outros problemas de otimização, tanto os objetivos quanto os parâmetros do processo podem variar ao longo do tempo, durante a etapa de resolução. Um exemplo aqui é o roteamento de veículos em grandes cidades, visando minimizar o tempo de percurso, sendo que as ruas e avenidas podem estar

sujeitas a níveis diferentes de congestionamento ao longo do trajeto dos veículos. Os mecanismos de otimização devem então ser adaptados às diversas fontes de variação que ocorrem durante a operação, que também caracterizam ambientes incertos.

Custos e ausência de recursos tecnológicos também podem impactar na ocorrência de incerteza em ambientes de otimização. É muito comum a necessidade de tomar decisões ignorando-se o comportamento de variáveis que sabidamente influenciam no andamento do processo, mas que não puderam ser monitoradas pelo custo dos sensores ou por ainda não existir tecnologia disponível para o projeto e implementação desses sensores. Custos e ausência de recursos tecnológicos também podem impactar na qualidade da atuação, de modo que mesmo que a decisão ótima seja conhecida a atuação não vai corresponder exatamente à decisão ótima que deveria ser tomada. Como exemplo para ambos os problemas (sensoriamento e atuação), pode-se citar toda e qualquer empresa prestadora de serviço e sua interação com os clientes. Como atender otimamente à demanda dos clientes e como perceber (sensoriar) qual é essa demanda? Como atrair novos clientes tendo em vista que a melhor maneira de atraí-los não é completamente viável? Qual o impacto de fazer isso de forma simplificada?

Por último, cabe mencionar a dificuldade de avaliação das soluções candidatas em problemas de otimização. Os algoritmos evolutivos e muitas outras meta-heurísticas requerem a atribuição de graus de qualidade às soluções, e isso pode ser muito custoso, em certos casos até impedindo uma avaliação precisa. Tem-se, assim, uma fonte importante de incertezas. Como exemplo, pode-se citar o processo de avaliação de potencial de novos produtos a serem lançados no mercado. Em qual ordem os produtos devem ser lançados de forma a causar maior impacto nos consumidores, dado um leque de opções? Pode-se até supor que o custo de lançamento de cada novo produto já seja conhecido, ou então esta passa a ser outra fonte de incerteza que também impacta na avaliação do potencial do produto.

Espera-se que os exemplos apresentados até aqui já permitam ao leitor identificar o quão comuns e relevantes são os problemas de otimização em ambientes incertos que precisam ser resolvidos no dia-a-dia das pessoas e das instituições. No entanto, para concluir esta seção serão contrastados dois problemas de otimização em ambientes incertos: um tratável e o outro intratável computacionalmente (já exemplificado acima, inclusive).

Considere o uso simples de um Navegador GPS, onde o usuário quer apenas traçar a menor rota entre sua residência e seu trabalho. Para isso, o sistema encontra a solução do problema do caminho mínimo, da Teoria de Grafos, para determinar a melhor rota entre dois pontos. Esse é um problema bem conhecido e para o qual existem diversas soluções de complexidade polinomial para resolvê-lo. Logo, é possível obter a solução de ótimo global em tempo razoável. Imaginem agora que, ao seguir o caminho indicado pelo GPS, que é a solução ótima, o usuário acaba parando em um congestionamento. Esse congestionamento pode ser encarado como um agente externo que mudou parâmetros do problema de otimização. A rota atual não é mais o ótimo global e, portanto, é preciso encontrar outra solução. Para esse exemplo, o procedimento é simples: basta atribuir um custo alto para os trechos que contêm congestionamento e pedir para o Navegador GPS encontrar uma nova solução.

Agora, considere um exemplo um pouco diferente: imagine que o usuário do Navegador GPS é uma empresa de distribuição de mercadorias. Essa empresa contém ao seu dispor diversos caminhões-baús para transportar e entregar lotes de mercadorias em n pontos de entrega. Cada caminhão, equipado com um navegador GPS, agora não quer apenas saber a menor rota entre os diversos pontos de entrega, considerados par-a-par. O objetivo a ser cumprido aqui é encontrar a sequência de n pontos de entrega que minimize o caminho total percorrido ou o tempo de entrega total. Esse problema é conhecido como o problema do caixeiro viajante (PCV) e, ao contrário do problema do caminho mínimo, não existe um algoritmo exato que retorne o ótimo global em um tempo razoável, isso quando n é grande. Esse é um problema em que o recomendado é o uso de meta-heurísticas que, embora não garantam produzir a solução ótima, fornecem uma solução de boa qualidade e que pode ser obtida antes dos caminhões partirem para as entregas. Se, no entanto, ao executar uma rota determinada pelo sistema o caminhão enfrentar o mesmo problema de congestionamento do exemplo anterior, resulta um novo

problema que já não é tão simples de se resolver. De fato, com essa perturbação de parâmetros do problema, não só a rota atual deixa de ser ótima mas, possivelmente, a sequência de pontos de entrega pode se alterar. Dado o custo para se resolver esse problema, não é sempre viável mandar o Navegador GPS recalcular a melhor solução. Este tipo de situação será tratado ao longo das próximas seções deste capítulo.

Nas próximas seções, quatro tipos de ambientes incertos em problemas de otimização serão explorados individualmente e de forma mais detalhada, com exemplos de casos reais. Será dada ênfase aos procedimentos geralmente adotados para adaptar algoritmos de computação evolutiva a cada contexto, buscando evitar degradação de desempenho no tratamento dos problemas de otimização resultantes. Adicionalmente, recomendações de leituras serão dadas em cada seção, visando complementar esta abordagem introdutória.

3. Problemas Ruidosos

Problemas ruidosos são aqueles em que existe uma incerteza no valor da função-objetivo durante a medição. Isso pode ocorrer, por exemplo, devido à ausência de acuidade nos sensores, à interferência de fatores externos e à presença de propriedades estocásticas do próprio problema de otimização. Usualmente, a função-objetivo para esse tipo de problema é representada conforme a Eq. 15.1:

$$F(x) = f(x) + \delta, \tag{15.1}$$

onde a função-objetivo original é acrescida de uma perturbação, geralmente de natureza gaussiana. Pode-se interpretar, então, $f(x)$ como a função-objetivo ideal, $F(x)$ como a função-objetivo medida e δ como uma distribuição normal com média zero e variância σ^2 definida experimentalmente.

Exemplos Ilustrativos

Para ilustrar esse tipo de ambiente incerto, pode-se considerar o problema de escalonamento de processos, muito comum em Ciência da Computação, o qual deve ser resolvido por sistemas operacionais que gerenciam ambientes multiprocessados. Esse problema consiste em, dados P processos a serem executados por M processadores de igual eficiência (tratar processadores homogêneos representa uma simplificação do problema), é requisitado que esses processos sejam distribuídos e enfileirados de forma a minimizar o tempo total de execução. Nesse problema, cada processo tem um dado tempo de execução nos processadores, conhecido a priori. Embora esse tempo de execução seja considerado determinístico, na prática ele pode apresentar graus de incerteza, sofrendo pequenas flutuações em torno do tempo esperado. É fácil constatar que cada perturbação causa uma diferença pequena no valor da função-objetivo, mas que, em conjunto, podem ser suficientes para impedir que a proposta de escalonamento considerada ótima (solução ideal) não corresponda à melhor solução possível para o problema real.

Outro exemplo pertinente é a otimização de parâmetros em algoritmos de Computação Evolutiva (Lobo et al., 2007). Nesses algoritmos, conforme o leitor deve ter observado em capítulos anteriores, existem diversos parâmetros que devem ser ajustados visando bom desempenho em aplicações práticas, com o agravante de que o melhor ajuste para uma certa aplicação pode ser bem diferente do melhor ajuste para uma outra aplicação do mesmo algoritmo. Além dos diversos parâmetros, existe uma grande diversidade de operadores de mutação e recombinação, os quais inclusive podem ser aplicados em conjunto, em diversas configurações e a diferentes taxas. Diante disso, fica caracterizado o problema da escolha da combinação ótima de parâmetros e operadores que conduzam ao melhor resultado em cada aplicação de um algoritmo evolutivo. Devido ao número de parâmetros que alguns algoritmos têm, sendo alguns deles com valores contínuos, e a diversidade de operadores, esse processo de escolha se configura como uma tarefa complicada de se resolver. Uma proposta para tornar esta tarefa automática

é a incorporação da própria combinação de parâmetros e operadores ao problema de otimização. Logo, como os algoritmos evolutivos são estocásticos por natureza, a avaliação das propostas de combinação de parâmetros e operadores torna-se incerta, em algum grau. O efeito é um ruído na função-objetivo do problema, visto que ela incorpora a avaliação de parâmetros do próprio algoritmo de otimização.

Nas próximas seções, o leitor poderá constatar que esses dois exemplos podem também ser interpretados como fontes de outros tipos de incerteza junto ao processo de otimização.

Reduzindo o Ruído

Média Explícita

Lidar com ruído na função-objetivo de algoritmos evolutivos pode ser relativamente simples, dadas algumas características dessa meta-heurística. A maneira mais direta para se aproximar do valor real da função-objetivo é a chamada *média explícita*, que consiste em calcular várias amostras do valor da função-objetivo em um dado ponto e, em seguida, extrair a média desse valor, conforme ilustrado na Eq. 15.2:

$$\hat{F}(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x), \tag{15.2}$$

onde $f_i(x)$ é o i-ésimo valor medido da função-objetivo, $\hat{F}(x)$ é a média amostral, ou seja, o valor estimado para a função-objetivo, e N é o número de amostras utilizadas. Conceitualmente, a existência de N amostras da função-objetivo num dado ponto leva à redução do desvio-padrão da média amostral correspondente por um fator de $\sqrt{N}$. Em outras palavras, o valor médio da função-objetivo torna-se mais confiável naquele ponto. Uma vez que tende a ocorrer um aumento da qualidade da avaliação de função ao aumentar o número de amostras, intuitivamente aumentar o número N de amostras, com a consequente redução do desvio-padrão da média, pode ser o caminho a seguir.

Mas, voltando ao exemplo de escalonamento citado acima, imagine que o problema é de larga escala, com milhões de processos a serem executados por milhares de processadores. O custo para computar a função-objetivo uma única vez já é alto e, como se trata de um algoritmo populacional, esse custo aumenta proporcionalmente ao tamanho da população (número de soluções candidatas). Assim, os custos inerentes dessa classe de algoritmos podem se ampliar ainda mais com a reamostragem da função-objetivo, trazendo problemas de escalabilidade aos algoritmos evolutivos.

Uma primeira alternativa, capaz de reduzir custos, seria calcular a média durante as iterações, ou seja, a cada iteração é feito um novo recálculo da função-objetivo para indivíduos remanescentes de gerações anteriores.

Em alguns artigos na literatura, encontram-se outras estratégias para se ter um número de amostras suficientes sem aumentar consideravelmente o custo do algoritmo. A solução mais simples, proposta por Aizawa e Wah (1994, 1993), consiste em definir o tamanho da amostragem adaptativamente durante a execução do algoritmo. Para tanto, eles adotaram duas estratégias: aumentar o número de amostras a cada geração ou usar um número de amostras maior para as soluções candidatas (indivíduos da população) com maior desvio amostral.

Uma outra maneira foi explorada em (Stagge, 1998), em que a média do valor da função-objetivo é calculada apenas quando dois indivíduos, i e j, são comparados entre si durante o processo de seleção. Como a avaliação dos dois indivíduos sofre o efeito de ruído, a comparação relativa entre ambos deve se dar por meio de testes estatísticos. Considerando a maximização da função-objetivo, o autor trabalhou com a hipótese nula

$$H_0 : \hat{F}(x_i) - \hat{F}(x_j) \geq 0, \tag{15.3}$$

que é a hipótese do indivíduo i ser melhor que o indivíduo j e que pode ser testada contra a hipótese alternativa

$$H_1 : \hat{F}(x_i) - \hat{F}(x_j) < 0. \tag{15.4}$$

Definidas as hipóteses, e dado um valor N_0 inicial de amostras (avaliações da função-objetivo), é realizado o teste-t de hipóteses (Student, 1908) que indica se a hipótese H_0 pode ser rejeitada com um erro probabilístico $\leq \alpha \in [0, 1]$, ou alternativamente, com grau de confiança $p = 1 - \alpha$. Se esse grau de confiança for baixo, então mais avaliações serão necessárias para i, j ou ambos. As amostragens continuam a ser realizadas até que o grau de confiança seja maior ou igual ao limiar ou um número N_{max} de amostragens seja atingido. Dessa forma, é possível deixar o número de avaliações por indivíduo no mínimo necessário para termos um grau de confiança acima de um limiar acerca da aceitação ou rejeição da hipótese. Outros testes de hipóteses podem ser adotados em substituição ao teste-t, dependendo do comportamento estatístico do ruído detectado no problema.

Uma estratégia parecida com essa foi adotada por Branke e Schmidt (2003, 2004); Cantú-Paz (2004), os quais fizeram o uso da técnica de amostragem sequencial. Esta técnica consiste em aumentar incrementalmente o número de amostras até que seja possível tomar uma decisão baseada na diferença de valores médios e na variância entre os dois conjuntos de amostras. Um exemplo é a amostragem sequencial baseada no valor observado da função-objetivo. Nesta técnica, a diferença entre $\hat{F}(x_i)$ e $\hat{F}(x_j)$ é calculada conforme a Eq. 15.5:

$$d^*_{ij} = \frac{\hat{F}(x_i) - \hat{F}(x_j)}{\sqrt{s_i^2 + s_j^2}}, \tag{15.5}$$

onde s_k é o desvio-padrão observado para o indivíduo k. Este procedimento está descrito na forma de pseudocódigo no Algoritmo 1.

Algoritmo 1 Pseudocódigo para o método de amostragem sequencial

```
Gere N_0 amostras para cada indivíduo (i e j)
Determine a diferença d*_ij entre os indivíduos i e j, conforme Eq. 15.5.
para k = 1 até K − 1 faça
    se |d*_ij| < ε_k então
        Retorne o melhor indivíduo
    fim se
    Gere uma amostra adicional para cada indivíduo
    Atualize d*_ij
fim para
Gere N_k amostras adicionais para cada indivíduo
Retorne o melhor indivíduo
```

O algoritmo começa gerando a amostragem inicial para cada indivíduo, calculando em seguida a diferença entre valores dessas duas amostragens. A partir daí, o algoritmo entra em um ciclo em que é verificado se a diferença é menor que um dado ϵ_k e, em caso afirmativo, retorna o melhor entre os dois indivíduos. Caso contrário, gera-se uma nova amostra para cada um. Após completarem um total de K amostras, caso a diferença ainda não seja suficiente para afirmar qual dos dois indivíduos é o melhor, são geradas mais N_k amostras e, então, o melhor indivíduo é retornado, mesmo sem informações suficientes. Pode-se verificar que, no pior caso, o algoritmo gera $K + N_k$ amostras para cada indivíduo.

Em lugar de realizar diversas amostragens da função-objetivo em um único ponto, uma proposta diferente é procurar estimar o valor da função-objetivo em um ponto a partir do valor obtido para ela

em outros pontos na vizinhança. Passa-se então de uma abordagem de média no tempo para outra de média no espaço, através de um critério de máxima verossimilhança (Kay, 1993). Neste caso, leva-se em conta um histórico de pontos previamente amostrados e sua distância em relação ao ponto a ser estimado. Essa técnica foi aplicada em (Sano e Kita, 2000; Sano et al., 2000; Sano e Kita, 2002) e tem como pré-requisito que o ruído tenha as mesmas características na vizinhança do ponto sendo avaliado e que a função-objetivo seja localmente suave.

Já Jaskowski e Kotlowski (2008) idealizaram 4 métodos diferentes de média explícita para selecionar o melhor indivíduo da população, dado um limite k máximo de amostras. O primeiro deles, chamado ingênuo (do inglês *naïve procedure*), distribui as k amostras igualmente para cada indivíduo da população, ou seja, se existem n indivíduos, cada um terá $\frac{k}{n}$ amostras para o cálculo da média. Após as amostragens serem feitas, o indivíduo que tem a média com maior valor (no caso de maximização) é retornado.

O segundo método, um pouco mais elaborado, é chamado de torneio (do inglês *tournament procedure*), e começa de maneira parecida com o procedimento anterior, amostrando uma quantidade igual para cada indivíduo. Só que, dessa vez, o número de amostras é metade do caso anterior, $\frac{k}{2n}$. Após a amostragem, os $n/2$ melhores indivíduos são selecionados para a fase seguinte, em que eles ganharão novas amostragens e o processo é repetido até que sobre apenas um único indivíduo. A ideia por trás desse método é que os melhores indivíduos tenham um número maior de amostras do que os piores indivíduos, já que estes últimos, mesmo com o efeito do ruído, não poderiam atingir o desempenho dos melhores indivíduos.

O terceiro método é chamado de seleção de candidatos (do inglês *candidate selection*) e consiste em amostrar os indivíduos com menor grau de confiança no teste de hipóteses, em comparação com o melhor indivíduo atual. O seu pseudocódigo está descrito em maiores detalhes no Algoritmo 2.

Algoritmo 2 Pseudocódigo para o método de seleção de candidatos

```
Gere 1 amostra para cada indivíduo
k ← k − n
enquanto k > 0 faça
    a_max ← elemento com maior valor médio
    a_prox ← elemento com segundo maior valor médio
    a_min ← elemento que minimiza o grau de confiança em relação a a_max
    se conf(a_min, a_max) < conf(a_max, a_prox) então
        Gere nova amostra de a_min
    senão
        Gere nova amostra de a_max
    fim se
    k ← k − 1
fim enquanto
Retorne o melhor indivíduo
```

Inicialmente, como já mencionado, é gerada uma amostra para cada indivíduo, restando apenas $k - n$ amostras a serem distribuídas. Em seguida, o algoritmo entra em seu laço principal onde escolhe três indivíduos: aquele com o maior valor médio (a_{max}, melhor candidato), aquele com o segundo maior valor médio (a_{prox}, concorrente direto) e aquele que tem um menor grau de confiança no teste de hipóteses em relação ao melhor indivíduo (a_{min}). Caso o grau de confiança entre esse último e o melhor indivíduo seja menor do que o grau de confiança entre o melhor e o segundo melhor indivíduo, uma nova amostra é gerada para a_{min}. Caso contrário, uma nova amostra é gerada para a_{max}. O processo se repete até que o número máximo k de amostragens seja alcançado, quando então é retornado o indivíduo com melhor média. A ideia por trás desse procedimento é distribuir as amostras

de tal forma a maximizar o grau de confiança de que o indivíduo a_{max} é o melhor da população. Para facilitar os cálculos, o grau de confiança é aproximado seguindo a Eq. 15.6:

$$conf(x, y) = (media(x) - media(y))^2 \cdot N_x, \tag{15.6}$$

onde N_x é o número de amostras de x.

Finalmente, o último procedimento, denominado bayesiano de um estágio à frente (do inglês *Bayesian one-stage ahead procedure*) e baseado numa técnica de amostragem denominada *Bayesian look ahead one-stage sampling*, requer o conhecimento dos parâmetros reais do ruído (média e variância) e de propriedades estatísticas da função-objetivo no espaço de busca. Em (Jaskowski e Kotlowski, 2008), esse algoritmo foi utilizado apenas como base de comparação com os outros três métodos. Ele dificilmente tem utilidade prática, uma vez que geralmente não se conhecem os parâmetros e propriedades requeridos. Com isso, e dada a complexidade desse procedimento, que foge do escopo introdutório desse texto, os autores remetem o leitor para a literatura específica.

Outras Abordagens

Uma alternativa para a média explícita, abordada na seção anterior, é o uso da *média implícita*, a qual consiste simplesmente em aumentar o tamanho da população. Em algoritmos populacionais, as áreas mais promissoras são mais exploradas e, nessas áreas, muitos indivíduos similares são gerados. Quando tem-se uma população muito grande, o efeito do ruído em um indivíduo é compensado pela avaliação da função-objetivo de todos os indivíduos na mesma região. Como consequência disso, em (Miller et al., 1996) foi demonstrado que, para uma população infinita, a seleção proporcional não é afetada pelo ruído.

Surge então a questão de qual abordagem é a melhor: aumentar o número de amostras por indivíduo (média explícita) ou aumentar o tamanho da população (média implícita). As investigações feitas até o momento mostram que cada técnica é melhor sob determinadas condições. Por exemplo, a média explícita mostrou-se melhor nos casos de uma estratégia evolutiva $(1, \lambda)$ (Beyer, 1993). Já a média implícita teve vantagens ao ser aplicada em uma determinada versão de algoritmo genético (Fitzpatrick e Grefenstette, 1988). Em (Miller et al., 1996; Miller, 1997) foram determinadas metodologias para calcular o valor ideal do tamanho da população e número de amostras para reduzir otimamente o efeito do ruído. Para tanto, Aizawa e Wah (1994) empregaram uma formulação de problema de escalonamento. Nessa formulação, o número de processos se torna o número de avaliações ou amostragens por indivíduo, o número de processadores é o tamanho da população e o tempo de execução é a duração de cada geração do algoritmo genético.

Outra abordagem que pode ser feita para atenuar o efeito do ruído é modificar o processo de seleção. A forma mais simples, e utilizada por Markon et al. (2001), é de simplesmente aceitar os filhos de certos indivíduos caso o valor da função-objetivo seja melhor que os dos pais acrescidos de um limiar. Em (Branke e Schmidt, 2003), os autores adotam o uso do cálculo de probabilidade de diferentes maneiras, para compensar o efeito do ruído no momento da seleção proporcional.

Casos Reais

Casos reais de problemas com função-objetivo ruidosa já foram estudados por Pietro et al. (2002), onde os autores levaram em conta a física ruidosa dos sensores de movimento do futebol de robôs, por Darwen (2000); Barone e While (2000), onde foi estudado o efeito do ruído em jogos de azar. O efeito do ruído também foi estudado em outros algoritmos bio-inspirados, por exemplo em otimização por colônia de formigas (Gutjahr, 2003) e otimização por enxame de partículas (Fernandez-Marquez et al., 2009), e também em algumas outras meta-heurísticas não-populacionais, como recozimento simulado (Gutjahr et al., 1996; Prudius e Andradóttir, 2005; Branke et al., 2008) e busca tabu (Costa e Silver, 1998).

4. Problemas Robustos

Problemas robustos são aqueles em que não se busca necessariamente a solução ótima, mas sim uma boa solução que apresente o mínimo de variações quando parâmetros e outros atributos do problema variam. Essa situação ocorre quando não há precisão na determinação dos valores dos parâmetros ou existe algum ruído na determinação dos mesmos, tolerâncias de fabricação, alterações sob diferentes condições (temperatura, umidade, armazenamento). A função-objetivo para esse tipo de problema é representada conforme a Eq. 15.7:

$$F(x) = f(x + \vec{\delta}), \tag{15.7}$$

onde $f(x)$ é a função-objetivo ideal, $F(x)$ é a função-objetivo medida e δ é a perturbação ou ruído aditivo, geralmente considerado ser de distribuição normal, com média zero e variância σ^2 definida experimentalmente. Repare que agora a perturbação está no argumento da função-objetivo e não no valor calculado a cada ponto.

Pode-se pensar nesse problema como um problema ruidoso, só que com o ruído agindo sobre o vetor de parâmetros que caracteriza o argumento da função-objetivo. Apesar disso, vale lembrar que o objetivo aqui não é atenuar o efeito do ruído na determinação de valores mais confiáveis para a função-objetivo, mas sim encontrar um ponto do espaço de busca que representa uma solução de boa qualidade e para a qual o efeito do ruído sobre a função-objetivo é mínimo.

Exemplos Ilustrativos

A aplicação mais óbvia para esse tipo de problema é na fabricação de produtos (Parmee et al., 1994; Thompson, 1998; Wiesmann et al., 1998; Kumar et al., 2006; Loyer e Jézéquel, 2009), onde as peças ou produtos projetados sofrem alterações durante ou após a fabricação devido a fatores como imprecisão na manufatura, temperatura e umidade do ar. O objetivo do fabricante é então encontrar bons parâmetros e que sofram uma menor perda de qualidade diante desses efeitos.

Um outro exemplo, que foi apresentado na seção anterior como um problema ruidoso, é o de escalonamento de processos, também visto por alguns autores como um problema robusto (Leon et al., 1994; Sevaux e Sörensen, 2002a,b). A razão disso é que, além de termos ruídos externos que influenciam o tempo gasto na alocação dos processos, existem também ruídos que afetam os parâmetros internos da função-objetivo, como diferenças na finalização prevista de um determinado passo ou disponibilidade de um componente. Logo, para este tipo de problema, pode ser indicado não buscar a solução ótima (considerando ausência de ruído), mas sim uma boa solução que tenha o mínimo de degradação de desempenho ao sofrer essas alterações.

Outros tipos de problemas tratados como problemas robustos são aqueles de natureza econômica e financeira (Schied, 2006; Pictet et al., 1996), pois eles envolvem taxas de juros e de câmbio que flutuam em certa faixa de valores, assim como outros índices que tendem a sofrer variações dentro de certos intervalos, quando imprevistos de grandes consequências não ocorrem. Cabe antecipar aqui que, quando as variações de parâmetros e outros atributos são mais acentuadas, resulta um outro tipo de problema de otimização a ser tratado mais adiante: problema dinâmico.

Encontrando uma Solução Robusta

Médias Explícitas e Implícitas

Visando obter soluções robustas para os tipos de problemas que acabam de ser descritos, geralmente são utilizadas duas abordagens diferentes. A primeira delas é o uso de técnicas de média explícita e implícita, da mesma forma que nos problemas ruidosos. A diferença aqui é que o ruído é introduzido de forma controlada no vetor de parâmetros, de forma a cobrir a área de tolerância estipulada. As

mesmas técnicas mostradas na seção anterior podem ser imediatamente aplicadas aqui para encontrar uma solução robusta, como o uso da média da aptidão com várias amostras geradas a partir de perturbações aleatórias (Thompson, 1998; Wiesmann et al., 1998). Em (Loughlin e Ranjithan, 1999; Markon et al., 2001) é utilizada a amostragem denominada *Amostragem por Hipercubo Latino* (do inglês *Latin Hypercube Sampling* - LHS) (McKay et al., 2000) para gerar as amostras de perturbação dos parâmetros. Considerando em termos estatísticos, uma matriz $n \times n$ é um *Quadrado Latino* (do inglês *Latin Square*) se existir apenas um único valor não nulo em cada linha e em cada coluna (ver figura 15.1). O Hipercubo Latino é uma generalização do Quadrado Latino para dimensões maiores e pode ser representado como uma matriz $n \times d$, onde n é o número de amostras e d é a dimensão do vetor de parâmetros, e cada coluna apresenta uma permutação de valores, dentre os valores possíveis de cada variável, de forma a se ter n amostras diferentes. Isso traz a vantagem de não repetir amostras e ainda assim explorar toda a área de distribuição do ruído.

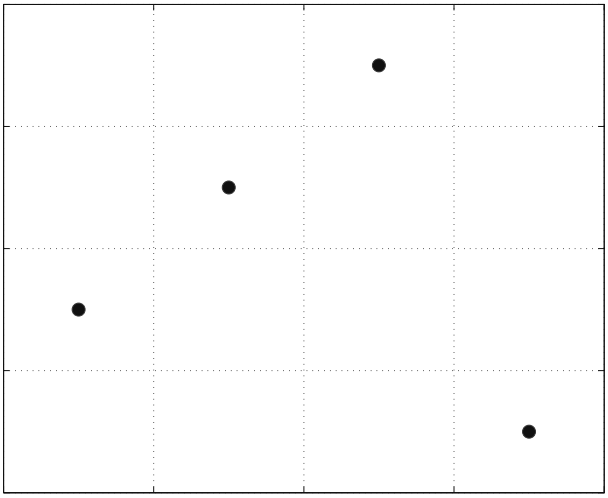


Figura 15.1: Exemplo de um Quadrado Latino, repare que cada linha e cada coluna contêm apenas uma amostra (representada por um círculo fechado).

Em (Branke, 1998), foi utilizada uma ideia similar à abordagem de média no espaço (Kay, 1993), já descrita no caso de problemas ruidosos. O valor da função-objetivo de um indivíduo é estimado através de uma média ponderada das soluções já calculadas, conforme Eq. 15.8:

$$f_{mod}(x_j) = \frac{\sum_i w_i.f(x_i)}{\sum_i w_i}, \tag{15.8}$$

onde x_j é a solução que se quer estimar, x_i são as soluções armazenadas no histórico e w_i é o peso que pondera o valor da função-objetivo para x_i. O peso é calculado em relação à distância do indivíduo que se quer estimar seguindo a Eq. 15.9:

$$w_i = max\{0, 1 - d \times b\}, \tag{15.9}$$

onde d é a distância entre a solução i e a solução j e b é um parâmetro definido pelo usuário. Essas soluções já calculadas podem ser tanto da população atual quanto de um histórico armazenado

referente a diversas iterações passadas.

Seguindo as mesmas soluções para Problemas Ruidosos, também foram feitos testes com média implícita (Tsutsui et al., 1996; Tsutsui e Ghosh, 1997; Branke, 1998), onde o aumento da população e a introdução de ruído nas variáveis de cada solução realizam uma amostragem equivalente à média explícita. De qualquer forma, novamente foi percebido que, dependendo da situação, uma proposta de tratamento pode ser melhor que a outra.

Formulação Multiobjetivo

Uma desvantagem em usar apenas a média, explícita ou implícita, do valor da função-objetivo de cada indivíduo é desprezar a sua variância (Das, 2000; Jin e Sendhoff, 2003). Uma alternativa é considerar dois critérios: o valor esperado para a função-objetivo e a sua variância. Mas para tanto, converte-se um problema de otimização mono-objetivo em um problema multiobjetivo. O leitor é convidado a consultar o capítulo que trata de Otimização Multiobjetivo para se familiarizar com os conceitos envolvidos, particularmente com os conceitos de soluções não-dominadas e de fronteira de Pareto e com a forma com que algoritmos evolutivos podem ser empregados no seu tratamento. Note que esses dois critérios não são necessariamente conflitantes, dependendo da natureza do problema e do ponto em que se encontram as soluções candidatas no espaço de busca. Outros critérios também podem ser adicionados ao problema multiobjetivo para permitir uma melhor escolha de soluções. Em (Ray, 2002), o próprio valor real de aptidão foi adicionado como critério. Outra opção é otimizar a função-objetivo e uma medida de *robustez* calculada como a razão entre o desvio padrão do valor da função-objetivo e o desvio padrão do ruído gerado no argumento da função-objetivo.

Com o uso de formulações multiobjetivo para resolver o problema de robustez em otimização em ambientes incertos, surgiu também o interesse em encontrar soluções para problemas multiobjetivos em que a fronteira de Pareto é robusta. Em (Deb e Gupta, 2005), foram examinados quatro casos diferentes:

- **A fronteira Pareto-ótima original é única e já é robusta:** o caso mais simples em que nada precisa ser feito;
- **Apenas parte da fronteira Pareto-ótima é robusta:** quando apenas parte da fronteira Pareto-ótima faz parte de uma região sensível a perturbações no espaço de variáveis. Neste caso, é necessário tratar apenas essa parte da fronteira de Pareto, buscando soluções mais robustas;
- **Uma fronteira Pareto-ótima local é robusta enquanto a global não é:** no caso em que a média das perturbações na fronteira local domina a média das perturbações na fronteira global, é preferível retornar a fronteira local, mais robusta.
- **Parte da fronteira Pareto-ótima global é robusta:** quando parte da fronteira global é robusta e na região sensível existe uma fronteira local que é robusta, o mais indicado seria combinar essas duas fronteiras de forma a constituir uma fronteira mais robusta.

Para lidar com esses problemas, foi utilizado o algoritmo genético multiobjetivo NSGA-II (Deb et al., 2002), substituindo as funções-objetivo pela média das amostragens de perturbações feitas, de forma similar ao procedimento do hipercubo latino. Outra forma utilizada é manter as funções-objetivo originais e adicionar uma restrição. Com esta restrição, são permitidas apenas soluções que tenham uma variância mínima no valor de cada função-objetivo, quando as variáveis são perturbadas. A formulação é apresentada a seguir:

$$\left.\begin{array}{ll} Minimizar & (f_1(x), f_2(x), \ldots, f_M(x)), \\ s.a. & \frac{\left\| f_i^{aprox.} - f_i(x) \right\|}{\|f_i(x)\|} \leq \eta, \qquad \forall i = 1..M, \\ & x \in S. \end{array}\right\}, \tag{15.10}$$

onde $f_i^{aprox.}$ é a média de n amostras de perturbação na solução para a função-objetivo i, M é o número de objetivos, S é o espaço de busca e $\|.\|$ pode ser qualquer medida de norma.

Em (Luo e Zheng, 2008), é explorado um algoritmo denominado *Eff-MOEA – Effective objective function based multi-objective evolutionary algorithm* – substituindo as funções-objetivo originais pela média dessas funções com perturbações no espaço de variáveis, amostradas através da técnica de *Monte Carlo*.

Uma outra forma de considerar o problema robusto é recorrendo à teoria de aritmética intervalar (Moore, 1966), geralmente utilizada para lidar com imprecisões numéricas computacionais. Nessa abordagem, o intervalo de tolerância que cada variável pode ter é codificado diretamente nos vetores de soluções. Dessa forma, um vetor de soluções passa a ser representado por:

$$[x] = [\underline{x}_1, \overline{x}_1], [\underline{x}_2, \overline{x}_2], ..., [\underline{x}_n, \overline{x}_n] \text{ para } x \in \Re^n, \tag{15.11}$$

onde $\underline{x}_i$ e $\overline{x}_i$ são, respectivamente, o limitante inferior e o limitante superior do intervalo da variável i.

Em (Soares et al., 2009), essa abordagem foi explorada em um algoritmo evolutivo multi-objetivo denominado [I]RMOEA – *Interval Robust Multi-Objective Optimization Evolutionary Algorithm* – onde o tamanho de cada intervalo é definido antes da otimização, de acordo com a precisão real do sistema. A avaliação das soluções é feita de acordo com um parâmetro de incerteza que define as amostras que serão geradas dentro do intervalo determinado junto a cada variável e, então, são retornados os piores valores obtidos dentre essas amostras. Nesse método, os operadores de dominância também foram alterados para lidar com a aritmética intervalar, de tal forma que uma solução deve dominar a outra dentro de todo o conjunto de parâmetros de incerteza atribuídos a eles.

De forma diferente, em (Forouraghi, 2000) foi feita a otimização não só dos parâmetros mas também do intervalo de tolerância. Dessa forma, é possível avaliar a melhor solução de um problema para diferentes tolerâncias. Outra novidade nessa abordagem foi o uso de medição de sinal-ruído como função de aptidão. Nessa metodologia, a aptidão de uma solução é calculada de acordo com a medição da avaliação do ruído, ou seja, da distribuição de amostras dentro de cada intervalo. Essas amostras são definidas através de um método de vetores ortogonais. A equação da aptidão é dada por:

$$\eta = -10 \log \left(\frac{\sum_{i=1}^{n} y_i^2}{n}\right), \tag{15.12}$$

onde y_i é o experimento i dentro do intervalo definido e n é o número de amostras.

Finalmente, em (Sanseverino, 2005), foi otimizado o "hipercubo" que envolve a tolerância de uma solução, ou seja, o quanto cada variável de uma solução pode ser alterada de forma a manter a diferença da função-objetivo dentro de um determinado limiar.

Uma forma alternativa de tratar as incertezas é aplicando lógica nebulosa (Pedrycz e Gomide, 2007) nos parâmetros. Dessa forma, cada parâmetro assume uma faixa variável de valores com diferentes graus de pertinência. O trabalho desenvolvido por Wang (2004) focou especificamente no problema de escalonamento, onde as incertezas foram modeladas na duração de cada processo, e o tempo entre o término de um processo e início de outro. Transformando esses valores em números nebulosos (Pedrycz e Gomide, 2007) é possível definir um cálculo de aptidão que reflete a possibilidade de uma solução não ser tão robusta, tendo em vista as imprecisões e incertezas.

Outras Aplicações e Técnicas

Outra situação em que Problemas Robustos podem ser estudados é quando as restrições são tratadas de forma robusta, ao invés da função-objetivo. A ideia aqui, ao contrário de soluções robustas em que se quer minimizar o prejuízo na aptidão quando há perturbações na solução, é garantir que a solução continuará viável mesmo na presença de perturbações. As mesmas técnicas descritas acima podem ser aplicadas aqui, mas com o cuidado em definir quando uma solução será considerada infactível (por

exemplo, quando o valor amostrado da restrição for infactível ou quando pelo menos uma amostra de perturbação for infactível). Esse problema é abordado por Deb et al. (2009) e denominado *Otimização Baseada em Confiabilidade*.

Aplicações recentes de otimização robusta podem ser encontradas em (Deb, 2008), onde foi feito um estudo de caso no problema de distribuição de energia hidrotérmica; Roy et al. (2009) estudaram o problema de projetar um sistema de resfriamento por bobinas; enquanto Lee et al. (2008) resolveram o problema de projetar a aerodinâmica em veículos de combate aéreo não-tripulados.

Outras técnicas bio-inspiradas já utilizadas para resolver problemas robustos são a *Otimização por Sistemas de Partículas Multiobjetivo* (Ono e Nakayama, 2009), que formula o problema como um problema bi-objetivo, conforme descrito anteriormente, e o *Sistema Hormonal Artificial* (Moioli et al., 2009) utilizado para controle autônomo de robôs móveis, onde é necessário estar imune a efeitos de ruídos em suas variáveis de decisão.

5. Aproximação de Função

Quando se modelam matematicamente problemas reais, nem sempre é possível modelá-los de forma completa ou precisa. Algumas vezes a razão está na complexidade intrínseca do problema, outras vezes está no custo do conhecimento e em outras circunstâncias está na ausência de recursos tecnológicos. Também podem ser inseridos neste contexto os erros involuntários ao longo do processo de modelagem. Em muitos casos, na ausência de uma formulação matemática para a função-objetivo, a atribuição de um valor de adaptação para as soluções candidatas é feita por operadores humanos, o que representa uma tarefa exaustiva, sujeita a erros e à incorporação de aspectos subjetivos na avaliação. A função-objetivo, nesses casos, é dada pela Eq. 15.13:

$$F(x) \approx E(x), \tag{15.13}$$

onde $E(x)$ é a função-objetivo estimada ou aproximada.

Exemplos Ilustrativos

Para os casos em que o custo de avaliação é alto, um problema muito estudado é a Otimização de Projetos Aerodinâmicos, que envolvem o cálculo da fluidodinâmica computacional. Há a necessidade de avaliar equações de Navier-Stokes em três dimensões, sendo que um computador moderno pode levar horas de cálculos para tratar uma única solução. Como, geralmente, em algoritmos de Computação Evolutiva é necessário avaliar milhares ou até milhões de soluções para obtermos uma solução de boa qualidade, esse problema pode se tornar inviável. Uma saída é utilizar aproximações dessa função original, o que acaba gerando um problema aproximado a um custo menor. Essa aproximação pode ser feita avaliando algumas amostras da função-objetivo real e, então, aproximando esta função por meio de métodos como modelo Kriging (Jeong e Murayama, 2004) e metamodelos (El-Beltagy e Keane, 1999) e por meio de técnicas de aprendizado de máquina como redes neurais artificiais (Rai, 2002; Hüsken et al., 2005) e máquinas de vetores-suporte (do inglês *Support Vector Machines*) (SVM) (Fan et al., 2005).

Já quando se fala em funções-objetivo que não se pode quantificar apropriadamente em termos determinísticos e não-subjetivos, citam-se como exemplos composição musical, síntese sonora, painéis e esculturas. No caso de composição musical, o objetivo do algoritmo é encontrar uma sequência de tons e timbres que gerem uma composição musical qualitativamente agradável para o ser humano ouvinte. Embora esse objetivo possa parecer bem claro, não existe uma função-objetivo bem definida para avaliar cada solução candidata. Intuitivamente, esse processo pode ser feito através da interação humana (Biles, 2002), onde existe um ou mais avaliadores que escutarão a a sequência sonora gerada pelo algoritmo e, em seguida, atribuirão um valor de acordo com a qualidade da solução gerada.

Um problema com essa abordagem é novamente a quantidade de avaliações que os algoritmos de Computação Evolutiva requerem e, portanto, a interação do avaliador pode se tornar cansativa, tender a erros e variações de critério. Outro problema é a subjetividade e arbitrariedade com que o avaliador atribuirá as notas. Uma maneira de reduzir esse problema é utilizar algum modelo de aprendizado para tentar entender a forma com que o avaliador atribui notas às composições. Um exemplo disso foi utilizado em (Johanson e Poli, 1998), onde os autores aplicaram uma rede neural para modelar as preferências do avaliador. Outras formas para resolver essa questão é a utilização de metamodelos criados a partir de certas estruturas de teoria musical (Wiggins et al., 1998). Uma linha de pesquisa bem estabelecida e que envolve operadores humanos na avaliação de soluções candidatas propostas por computador, particularmente em projetos artísticos, é a de algoritmos evolutivos interativos (Bentley, 1999; Takagi et al., 2001; Moroni et al., 2002; Machwe e Parmee, 2007).

Aproximando as Funções

Os métodos utilizados para aproximar funções podem ser divididos em três frentes, segundo Jin (2005). Duas dessas frentes são tradicionais da área de otimização e uma é específica da área de computação evolutiva:

- *Aproximação do Problema*: quando um método é dito aproximar um problema, significa que ele procura encontrar um problema simplificado aproximado do original mas com um custo menor. Uma maneira de implementar esse método é negligenciando certos aspectos da formulação do problema, como restrições e efeitos externos (Engquist e Runborg, 2002; Jaisankar e Raghurama Rao, 2007). Outra forma seria através de simulações numéricas contando com diferentes níveis de acuidade, impactando na qualidade da avaliação.

- *Aproximação Funcional e Meta-modelos*: a aproximação funcional refere-se à construção de um modelo matemático explícito e equivalente ao original. Tal modelo descreve um espaço de busca aproximado enquanto economiza em custo computacional. Quando esse tipo de aproximação é feita através de um modelo matemático, sua implementação se torna específica para um determinado problema e muitas vezes sua criação é difícil e custosa. Uma forma mais comum de aproximação funcional são os *meta-modelos*, que são sintetizados a partir de abordagem bottom-up, geralmente empregando aprendizado a partir de dados amostrados.

- *Aproximação Evolucionária*: esse tipo de aproximação foi concebida para algoritmos evolutivos e explora a abordagem populacional para gerar conhecimento sobre o espaço de busca. Esse tipo de aproximação leva em conta a distribuição espacial entre os indivíduos de uma geração e sua prole, de forma que seja possível utilizar aproximações com maior ou menor acuidade, dependendo da quantidade de informação existente na região do novo indivíduo. Métodos comuns a esse tipo de aproximação são: herança de aptidão (do inglês *fitness inheritance*), imitação de aptidão (do inglês *fitness imitation*) e atribuição de aptidão (do inglês *fitness assignment*).

Visando manter o foco nos algoritmos evolutivos, apenas os métodos dos dois últimos itens serão descritos a seguir.

Construção de meta-modelos por regressão não-paramétrica

Existem, na literatura, diversos métodos para construção de meta-modelos baseados em regressão não-paramétrica, ou seja, sem conhecimento a priori da forma da função a ser estimada. Esses modelos se diferenciam de acordo com o nível de precisão obtido em relação à função original e à quantidade de amostras que cada um requer. A seguir, são apresentados os métodos mais comuns da literatura.

Modelo Linear

O modelo linear de regressão gera uma combinação linear de diversas funções (lineares ou não) de forma a aproximar o máximo possível a curva gerada aos pontos de amostragem da função-objetivo real $f(x)$. Esse modelo assume a seguinte forma:

$$F(x) = \sum_{j=1}^{m} w_j h_j(x), \tag{15.14}$$

onde x pertence a uma região compacta (fechada e limitada), $F(x)$ é a função aproximada, h_j é a j-ésima função-base, w_j é o peso atribuído à j-ésima função-base e m é o número de funções-base.

A Eq. 15.14 representa uma combinação linear de m funções-base, onde sua estrutura e parâmetros são pré-definidos. Mesmo que as funções-base sejam não-lineares, o modelo ainda é considerado linear nos parâmetros, pois a flexibilidade de obtenção de $F(x)$ está relacionada apenas à escolha dos coeficientes da combinação linear $(w_j, j = 1, ..., m)$. Funções-base comumente usadas são polinomiais, wavelets, senoides e sigmoides. Os coeficientes da combinação linear são geralmente definidos com base em métodos de quadrados mínimos e restrições de suavidade podem ser adicionadas ao modelo.

Modelo Kriging

O modelo Kriging é um modelo generalizado do modelo linear de interpolação que considera a existência de correlação espacial entre amostras próximas. A equação básica de um modelo Kriging apresenta poucas diferenças em relação ao modelo linear, assumindo a forma:

$$F(x) = \sum_{j=1}^{m} w_j(x) h_j(x), \tag{15.15}$$

onde agora os coeficientes da combinação linear são dependentes de x e as funções-base dependem dos dados amostrados.

Os pesos são calculados de forma que a média dos erros entre os valores reais e os estimados seja nula e a variância seja mínima. Na verdade, Kriging se refere a um conjunto de métodos, sendo que um dos mais utilizados é aquele que obtém os pesos na forma:

$$w(x) = \begin{pmatrix} c(x_1, x_1), & \dots & c(x_1, x_n) \\ \vdots & \ddots & \vdots \\ c(x_n, x_1), & \dots & c(x_n, x_n) \end{pmatrix}^{-1} \begin{pmatrix} c(x_1, x) \\ \vdots \\ c(x_n, x) \end{pmatrix}, \tag{15.16}$$

onde $c(x_i, x_j)$ é um índice que estima a covariância entre duas amostras.

Redes Neurais Artificiais

Redes Neurais Artificiais (Haykin, 1999) (RNAs) representam uma técnica de inteligência computacional aplicada basicamente a quatro tipos de problemas de aprendizado de máquina: memória endereçável por conteúdo, classificação de padrões, agrupamento de dados e síntese de mapeamentos não-lineares multidimensionais. Haverá ênfase aqui neste último tipo de problema de aplicação, em que as redes neurais operam como modelos de regressão não-paramétrica.

A capacidade de aprendizado e a estrutura conexionista de redes neurais resultam da modelagem matemática de neurônios e suas conexões sinápticas no cérebro. O princípio de operação das redes neurais artificiais se fundamenta na interação de unidades de processamento simples e similares, denominadas neurônios artificiais. Os vários padrões já propostos para conexão entre neurônios artificiais

definem estruturas de processamento de informação com grande poder de síntese de mapeamentos não-lineares multidimensionais, a partir do ajuste dos pesos associados às conexões sinápticas. Se existirem conexões recorrentes, o mapeamento exibe propriedades dinâmicas, senão ele é estático. Serão considerados aqui mapeamentos estáticos e aprendizado supervisionado, em que estão disponíveis amostras do mapeamento a ser aproximado.

Apenas duas versões de RNAs voltadas para aprendizado a partir de dados amostrados serão consideradas a seguir, devido à relevância e ampla aplicação de ambos os modelos, além do que eles admitem uma representação similar àquela já adotada para descrever os modelos lineares. O primeiro deles é o perceptron de múltiplas camadas (MLP, do inglês *multilayer perceptron*) (Lippmann, 1987), o qual apresenta capacidade de aproximação universal (Cybenko, 1989), ou seja, aproxima com grau arbitrário de precisão qualquer mapeamento contínuo em uma região compacta do espaço de aproximação. No entanto, por não se conhecer a priori a forma do mapeamento que se deseja aproximar, é geralmente um desafio especificar o número de neurônios e o quanto a rede neural deve ser treinada para produzir uma boa capacidade de generalização, ou seja, para responder bem a amostras não empregadas durante a fase de treinamento supervisionado (ajuste dos pesos sinápticos). Como critério de parada para o treinamento supervisionado, pode-se separar parte do conjunto de amostras disponível para treinamento, criando um conjunto de dados de validação. O mapeamento realizado por uma rede neural do tipo perceptron com uma camada intermediária de neurônios é dado na forma, sendo $F_k(.)$ a saída produzida pelo k-ésimo neurônio da camada de saída:

$$F_k(x) = \sum_{j=1}^{n} w_{kj} f(\sum_{i=1}^{m} v_{ij} x_i + v_{0j}) + w_{k0}, \tag{15.17}$$

onde m é o número de entradas, n é o número de neurônios na camada intermediária, $f(.)$ é a função de ativação desses neurônios, k é o índice da saída, w_{kj} é o peso sináptico que conecta a saída do j-ésimo neurônio da camada intermediária ao k-ésimo neurônio da camada de saída e v_{ij} é o peso sináptico conectando a i-ésima entrada ao j-ésimo neurônio da camada intermediária. A função de ativação dos neurônios da camada intermediária geralmente é tomada como sendo a função tangente hiperbólica. Os índices i e j se iniciam em zero porque é necessário considerar uma entrada constante, denominada entrada de polarização, para todos os neurônios. Se apenas os pesos w_{kj} fossem ajustáveis, resultaria um modelo linear nos parâmetros ajustáveis. No entanto, os pesos v_{ij} são também ajustáveis, resultando um modelo não-linear de aproximação.

Uma outra forma bem conhecida de arquitetura de RNA que pode ser usada para a síntese de mapeamentos não-lineares é a rede neural com funções de ativação de base radial (RBF, do inglês *radial basis function*) (Broomhead e Lowe, 1988). A principal diferença entre as redes neurais MLP e RBF está no formato das funções de ativação dos neurônios da camada intermediária ($f(.)$) e na forma como as entradas são processadas pelos neurônios da camada intermediária. Na MLP é feito um produto interno, enquanto que na RBF é calculada a distância euclidiana. O mapeamento realizado por uma rede neural com funções de ativação de base radial é dado na forma, sendo $F_k(.)$ a saída produzida pelo k-ésimo neurônio da camada de saída:

$$F_k(x) = \sum_{j=1}^{n} w_{kj} \phi(\|x - c_j\|) + w_{k0}, \tag{15.18}$$

onde $\phi(.)$ é a função de base radial com centro em c_j e $\|.\|$ é a norma euclidiana. A rede neural RBF também apresenta capacidade de aproximação universal (Park e Sandberg, 1991).

Máquinas de Vetores-Suporte

As máquinas de vetores-suporte (SVM, do inglês *support vector machines*) (Vapnik, 1998) compõem um método de aprendizado estatístico que procura superar a limitação apresentada pelos modelos com capacidade de aproximação universal, ou seja, a dificuldade de definir o grau de flexibilidade do modelo não-paramétrico. Ao mapear os dados do problema original para um espaço de maior dimensão, empregando métodos de kernel, o que se busca é um hiperplano que apresente margem máxima (no caso de problemas de classificação) ou que melhor aproxime os dados, no caso de problemas de regressão. Ao retornar ao espaço original, produz-se o mapeamento não-linear que resolve o problema, com grau de flexibilidade adequada. As máquinas de vetores-suporte foram originalmente concebidas para resolver problemas de classificação binária, sendo posteriormente estendidas para regressão e agrupamento de dados. Os vetores-suporte, neste caso de classificação binária, são as amostras que definem a posição do hiperplano de máxima margem de separação. Em qualquer caso, encontrar o hiperplano no espaço expandido implica em resolver um problema de otimização quadrática com restrições de igualdade e/ou desigualdade, o qual tem solução única. Além de ter solução única, a complexidade do modelo não depende do número de entradas, mas sim do número de vetores-suporte.

Extraindo mais da população a cada geração

A população gerada pelos algoritmos de computação evolutiva são geralmente ricas em informação do espaço de busca a ser otimizado, além de tender a amostrar mais regiões com maior potencial de produção de indivíduos com bons níveis de avaliação. Diante disso, algumas técnicas foram desenvolvidas para minimizar o número de avaliações reais da função-objetivo e, assim, reduzir o custo quando se depara com funções-objetivo muito custosas. Essas técnicas serão descritas brevemente a seguir.

Herança de aptidão

Herança de aptidão (do inglês *fitness inheritance*) (Smith et al., 1995) é uma técnica que estima o valor da função-objetivo (*fitness*) de parte dos novos indivíduos a partir do valor já atribuído às soluções-pais, geralmente se baseando nos seus blocos construtivos (do inglês *building blocks*) (Goldberg, 1989). Blocos construtivos correspondem a um conceito empregado para justificar a eficácia dos algoritmos evolutivos e remetem a pedaços de boas soluções, ou padrões dentro da codificação, que contribuem para a avaliação dos indivíduos que os contêm. Sendo assim, ao longo das gerações, esses blocos construtivos tendem a se espalhar pela população. Com isso, é possível inferir que a dimensão do problema original é reduzida para o número de blocos construtivos.

Para se avaliar os indivíduos a partir de blocos construtivos, deve-se inicialmente obter a média da aptidão de todos os indivíduos que contêm cada bloco construtivo, de acordo com a Eq. 15.19:

$$F(x) = \frac{f}{l} + (l-1)p, \tag{15.19}$$

onde f é a aptidão real do bloco construtivo, l é o tamanho do bloco construtivo e p é a proporção de blocos construtivos corretos dentro do indivíduo.

Em (Sastry et al., 2001), foi avaliado o ganho de desempenho ao usar herança de aptidão com a proporção ótima de indivíduos que herdam a aptidão e aqueles que têm sua aptidão calculada com a função original. Nesse estudo, concluiu-se que, com o uso de uma população fixa, o número de avaliações da função real pode ser reduzido em até 70%.

O desempenho da herança de aptidão foi também avaliado para problemas multiobjetivo (Chen et al., 2002; Ducheyne et al., 2008). Chen et al. (2002) obtiveram um ganho na redução das avaliações de um algoritmo multiobjetivo com compartilhamento de aptidão (explicado a seguir). Já Ducheyne et al. (2008) concluíram que o uso de herança de aptidão em problemas não-convexos e não-contínuos pode afetar negativamente o desempenho do algoritmo.

Uma outra estratégia possível, que não trabalha diretamente com os blocos construtivos, é utilizar uma média ponderada da aptidão dos pais em relação ao quanto de cada pai o indivíduo absorveu (Chen et al., 2002; Salami e Hendtlass, 2003).

Imitação de aptidão

Imitação de aptidão (do inglês *fitness imitation*) foi idealizada por Kim e Cho (2001) e consiste em agrupar indivíduos da população em vários grupos distintos, através de uma medida de dissimilaridade. Ao agrupar os indivíduos, um deles será selecionado para ser o centro do grupo, e terá sua aptidão avaliada pela função-objetivo real. Os outros indivíduos pertencentes ao grupo terão sua aptidão estimada com base na aptidão do centro (imitação) e proporcional à dissimilaridade entre eles. O algoritmo de agrupamento e a fórmula de dissimilaridade podem ser escolhidos de acordo com o problema tratado. Em (Kim e Cho, 2001), foram utilizados o método k-médias de agrupamento (Cover e Hart, 1967) e a medida de distância euclidiana.

Em (Bhattacharya e Lu, 2003), foi criado um algoritmo evolutivo que combina os métodos de imitação de aptidão com o modelo *Support Vector Machine* (SVM) de regressão. Ele inicia o procedimento amostrando 5 vezes mais soluções do que o tamanho total da população e calculando o valor real da função-objetivo desses pontos de forma a gerar um modelo de regressão utilizando o SVM. Após esse passo, as N melhores soluções são selecionadas para fazer parte da população inicial. A seguir, aplica-se o processo de seleção de pais, reprodução e mutação, gerando novos indivíduos. Esses indivíduos terão sua aptidão calculada através do modelo de regressão do SVM. Em seguida, a partir da nova população, são gerados m agrupamentos e a aptidão de cada novo indivíduo é recalculada a partir da seguinte fórmula:

$$F(x) = f_{SVM}(x) - \rho_1\sigma_i - \rho_2 d_i - \rho_3 s_i, \tag{15.20}$$

onde ρ_1, ρ_2, ρ_3 são pesos atribuídos a cada um dos fatores da equação, $f_{SVM}(x)$ é a aptidão calculada via SVM, σ_i é o desvio padrão do valor da função-objetivo do cluster i ao qual o indivíduo pertence, d_i é a distância euclidiana mínima normalizada entre a solução e os elementos do cluster i que tiveram sua aptidão avaliada pela função-objetivo original, e s_i é a dispersão dos elementos do cluster i, calculada pela Eq. 15.21:

$$s_i = \frac{no.\_de\_individuos\_no\_cluster\_i}{dimensão\_do\_individuo}. \tag{15.21}$$

Com as aptidões aproximadas calculadas, encontra-se o valor ótimo atual aproximado e calcula-se o seu valor real e também o de seus k-vizinhos mais próximos. Repete-se esse procedimento para cada um dos m centróides referentes aos agrupamentos construídos. Esses novos pontos calculados serão adicionados ao modelo do SVM para ser feita uma atualização do modelo de aproximação. Finalmente, o processo se repete até que o critério de convergência seja alcançado.

Em (Jin e Sendhoff, 2004), foi criado um algoritmo parecido com o descrito acima, mas utilizando um *ensemble* de redes neurais para aproximar a função, calculando a função real apenas dos centroides dos grupos.

Atribuição de Aptidão - Algoritmos coevolutivos

O conceito de atribuição de aptidão (do inglês *fitness assignment*) está diretamente ligado aos algoritmos coevolutivos. Esses algoritmos representam um sistema de competição entre duas populações que evoluem em paralelo, onde a aptidão de cada indivíduo de uma população vai depender dos indivíduos da outra população. Esses algoritmos são particularmente úteis quando não existe um objetivo definido, ou seja, quando ocorre uma situação em que se quer saber, dados dois problemas correlatos, quando a solução de um problema compete bem, ou se aplica melhor, na solução do outro problema.

Dessa forma a pressão seletiva impõe alterações na avaliação durante a evolução, tornando o processo dinâmico e competitivo.

Uma forma de utilizar esse conceito em avaliação de indivíduos é imaginar que uma população é formada pelas soluções candidatas do problema a ser resolvido e a outra por uma série de modelos de predição capazes de aproximar a aptidão. Em (Schmidt e Lipson, 2008), essa ideia é estendida adicionando uma terceira população que evolui o conjunto de amostras de treinamento dos preditores.

Esse algoritmo começa com a população de soluções candidatas, onde várias iterações de um algoritmo evolutivo são efetuadas utilizando o melhor preditor atual, até que o erro de aproximação da melhor solução obtida seja menor que um limiar. Em seguida, passa-se para a população de preditores, onde eles sofrem também um processo de evolução (o treinamento de uma rede neural do tipo MLP ou RBF pode ser feito através de algoritmos evolutivos também) e, periodicamente, evoluem também a população de amostras de treinamento, aumentando a variabilidade de soluções. Idealmente, esses processos de evolução são feitos em paralelo.

Falsos Ótimos

Muitas vezes os pontos amostrados não são suficientes para dar uma indicação de onde se encontra o ótimo global ou, ao menos, boas soluções em uma região do espaço de busca que está sendo explorada. Mesmo que o cálculo real do valor da função-objetivo (aptidão) seja feito para alguns indivíduos da população, a baixa amostragem em regiões em que a superfície de aptidão sofre variações mais intensas pode conduzir a aproximações equivocadas e, por consequência, a falsos ótimos.

Em outros casos, a taxa de amostragem pode estar adequada, mas a distribuição não-uniforme de amostras pela superfície de aptidão pode induzir uma aproximação equivocada, a qual novamente pode guiar o processo evolutivo a falsos ótimos.

Um exemplo ilustrativo considerando um espaço de busca de dimensão 1 é apresentado na Figura 15.2.

Outras Aplicações e Técnicas

É evidente que essas estratégias de aproximação de funções podem ser empregadas junto a outras meta-heurísticas populacionais que operam em espaços de busca. Um exemplo é a otimização por enxame de partículas (PSO, do inglês *particle swarm optimization*) (Hendtlass, 2007).

Aproximações da função-objetivo também podem ser empregadas na presença de ambientes ruidosos (Branke et al., 2001), visando reduzir o efeito do ruído, e em otimização robusta (Paenke et al., 2006).

Uma outra contribuição possível da aproximação de funções-objetivo pode ocorrer em cenários em que os ótimos locais impedem ou dificultam a localização do ótimo global. Nesses casos, é possível substituir a função-objetivo original por uma aproximação que procura eliminar ótimos locais ou ao menos reduzir a quantidade destes, favorecendo a busca pelo ótimo global (Yang e Flockton, 1995; Liang et al., 2000, 1999).

6. Problemas Dinâmicos

Até este ponto do capítulo, foram apresentados problemas de otimização em ambientes incertos em que: (*i*) existe uma variação no valor da função-objetivo, em torno de seu valor real (ruído com propriedades estatísticas invariantes); (*ii*) a otimização deve levar em conta intervalos de valores admissíveis para certas variáveis e parâmetros, buscando a robustez da solução; (*iii*) a otimização deve se dar em situações em que é inviável ou impossível obter o valor real da função-objetivo, requerendo assim a definição de valores aproximados. Agora, será apresentado o quarto tipo de incerteza que pode surgir

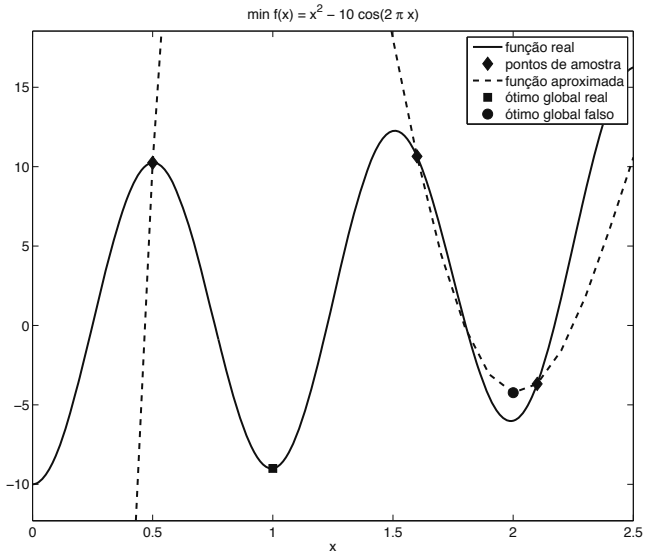


Figura 15.2: Exemplo da criação de um falso ótimo global por conta da aproximação de função com amostragens insuficientes em um problema de minimização. Note que a forma aproximada da função-objetivo (em linha tracejada) difere da forma real (em linha sólida) em algumas regiões onde faltaram pontos de amostra. Nessa situação, o ótimo global da função aproximada se torna o ponto localizado pelo círculo sólido, enquanto o ótimo global real, representado pelo quadrado sólido, está em um ponto diferente da função. Esse problema poderia ser solucionado com a obtenção de um número suficiente de amostras bem localizadas, o que geralmente não é fácil de conseguir.

quando se resolve um problema de otimização: a incerteza causada por variações dinâmicas ao longo do tempo. Esse tipo de problema difere bastante dos outros três, pois não só a intensidade de variação do valor da função-objetivo é maior, como também podem ocorrer mudanças estruturais no espaço de busca. Com uma maior intensidade de variação da função-objetivo, podem resultar fenômenos como mudanças na localização de um ou mais ótimos, surgimento de novos ótimos e desaparecimento de ótimos existentes. Já um exemplo de mudança estrutural está na variação da dimensão do problema de otimização.

Para representar esse tipo de problema, geralmente é utilizada a notação da Eq. 15.22, onde consta que a função é dependente, não só do vetor de variáveis, mas também do tempo:

$$F(x) = f(x, t), \tag{15.22}$$

onde $f(x, t)$ é a função-objetivo que varia em relação ao tempo t. Note que aqui o termo tempo não necessariamente representa o conceito de tempo físico, mas sim uma sequência de eventos discretos, de origem interna ou externa ao sistema.

Exemplos Ilustrativos

Um exemplo de problema de otimização que varia com o tempo, e que ocorre no dia-a-dia de boa parte das pessoas que vivem em grandes cidades, é o problema de definição de rotas na presença de congestionamentos do tráfego urbano, citado brevemente na introdução desse capítulo. Hoje em dia, muitos veículos já contam com um aparelho de GPS para facilitar a localização e navegação pela cidade. Esses aparelhos têm a capacidade de encontrar a rota que tem o menor custo (distância)

entre o ponto em que o veículo se encontra e o destino desejado. Esses algoritmos conseguem traçar essa rota quase instantaneamente, auxiliando o motorista no caminho a seguir. Mas os aparelhos de navegação não contêm informações sobre tráfego e possíveis congestionamentos, nem mesmo a localização e estado dos semáforos, uma vez que todos esses requisitos dependem do instante em que o motorista está passando pelos locais. Com isso, a solução dada pelo navegador, embora ótima quando se consideram condições de tráfego uniforme em todas as vias, pode vir a ter uma qualidade ruim por conta dos imprevistos e das condições não-uniformes de tráfego. Em uma situação ideal, o algoritmo seria capaz de perceber o mais breve possível a mudança no ambiente de busca e reagir traçando uma rota alternativa que evitasse imprevistos ou fatores que possam reduzir a qualidade da solução. Claro que, no caso de uma rota entre dois pontos, o navegador poderia facilmente recalcular uma nova rota partindo do ponto atual em que o motorista avistou o congestionamento, uma vez que a solução desse problema é relativamente barata. Mas quando se trata de rotas para múltiplos veículos, sob requisitos de otimização global (Ahn e Shin, 1991; Fleischmann et al., 2004), ou ciclos hamiltonianos (Guntsch e Middendorf, 2001; Zhou et al., 2003), o problema se torna $\mathcal{NP}$-difícil. Nessas circunstâncias, reiniciar o processo de otimização (supondo que o problema é estático) para cada novo estado do problema torna-se inviável, o que remete a técnicas de otimização capazes de operar sob condições e requisitos que variam dinamicamente.

Outro exemplo, já citado em seções anteriores, é o problema de escalonamento de processos computacionais (Savic et al., 1997; Dellaert et al., 2000; Ingolfsson et al., 2002). Como visto, esse problema trabalha com um número pré-fixado de processadores e processos, mas durante a execução da solução obtida pode ocorrer uma mudança em uma etapa de algum processo, ocasionando atraso na execução desse, um ou mais processadores podem sair de operação e processos novos e urgentes podem surgir de última hora. Todas essas mudanças causam uma alteração na dimensão do problema e, portanto, no espaço de busca, além de mover as boas soluções ao longo do espaço de busca. Adicionalmente, o custo de execução de cada processo pode se alterar com o passar do tempo, tornando algumas soluções infactíveis.

Lidando com Mudanças

Um ponto comum das meta-heurísticas populacionais, incluindo os algoritmos evolutivos, é a tendência de convergir para uma solução de ótimo local, que pode até ser o ótimo global. Em ambientes variantes no tempo, isso é algo indesejável, uma vez que o algoritmo perde a capacidade de reação ao se confrontar com uma mudança mais intensa de características do problema. Outro ponto importante é a detecção de que realmente houve uma mudança na função-objetivo e, portanto, deve-se reavaliar as soluções com as quais o algoritmo está trabalhando nesse momento. Note que simplesmente reavaliar todas as soluções a cada passo do algoritmo pode não ser desejável pelo custo computacional. Em revisão feita por Jin e Branke (2005), foram destacadas características incorporadas por algoritmos evolutivos para resolver essa situação, concluindo que o uso de uma ou mais dessas características, quando possível, são desejáveis para lidar com esse tipo de problema. Essas características são:

- *Geração de Diversidade*: um modo simples de evitar a convergência do algoritmo é simplesmente inserir diversidade de tempos em tempos, ou quando uma mudança no problema de otimização for detectada.

- *Manutenção de Diversidade*: de forma similar ao item anterior, mas um pouco mais elaborada, pode-se utilizar mecanismos que promovam a capacidade de exploração do espaço de busca, a todo momento. Isso obviamente implica em custos computacionais adicionais para o processo de otimização, visto que haverá um compromisso entre o investimento voltado para localizar o ótimo global e aquele voltado para manter soluções diversas na população.

- *Memória de Soluções*: o algoritmo pode também implementar uma memória de soluções passadas que pode ser utilizada na inserção de diversidade ou de forma a reinicializar a busca diante de mudanças no ambiente. Essa abordagem é particularmente útil em situações em que a mudança é cíclica, e o ótimo tende a passar sempre pelos mesmos lugares, embora nem sempre numa sequência previsível.

- *Abordagem Multipopulacional*: ao dividir a população original em várias sub-populações, pode-se incentivar as populações a se manterem afastadas entre si, explorando assim regiões distintas do espaço de busca. É evidente que cada sub-população pode ser composta por suas próprias sub-populações, e assim por diante, numa organização hierárquica e aninhada.

Uma outra característica, que inicialmente foi concebida com os Sistemas Imunológicos Artificiais (de Castro e Timmis, 2002), mais especificamente o algoritmo aiNet (de Castro e Von Zuben, 2001), e recentemente tem sido empregada em alguns algoritmos para otimização em ambientes variantes no tempo, é o controle dinâmico do tamanho da população (de França et al., 2005; Blackwell, 2007). Uma vez que é necessário manter diversidade sem perder o poder de exploração, os algoritmos tendem a necessitar de uma população com um tamanho maior ou menor que o inicialmente atribuído. Tendo um controle dinâmico desse tamanho, a partir de uma proposta inicial, pode-se minimizar o acréscimo do custo computacional gerado pelas técnicas apontadas anteriormente.

Além de incorporar tais características no algoritmo, pode ser desejável também alterar alguns mecanismos internos, como aqueles vinculados a operadores de mutação, reprodução e seleção, nos algoritmos evolutivos.

A seguir, será descrito, em linhas gerais, como cada uma dessas características desejáveis é implementada na literatura.

Introdução de Diversidade

Para gerar diversidade em algoritmos evolutivos, conta-se com algumas opções. Em (Cobb, 1990), por exemplo, toda vez que uma mudança é detectada, por reavaliação de indivíduos, a taxa de mutação é aumentada drasticamente de forma que a variabilidade das soluções aumente. Isso gera algumas questões, como o quanto essa taxa deve ser aumentada. Se aumentada demasiadamente, a busca se torna aleatória. Se for pequeno demais, pode não ter o efeito desejado.

Uma alternativa é aumentar gradativamente a taxa de mutação (Vavak et al., 1997), de forma controlada, e reduzi-la quando atingir níveis muito altos. De forma complementar, essas estratégias podem ser feitas também em relação à intensidade da mutação ou ao investimento de mais recursos na execução de etapas de busca local.

Outra estratégia é substituir indivíduos por novos amostrados em regiões distintas daquelas que estão sendo exploradas pelo algoritmo. Os indivíduos substituídos podem ser simplesmente os n piores ou os mais similares, por exemplo. Essa estratégia é conhecida como *imigração* (Grefenstette, 1992).

Manutenção de Diversidade

A ideia de manter diversidade não é exclusiva de ambientes variantes no tempo. Esse conceito já foi amplamente utilizado junto a meta-heurísticas populacionais para otimização, de forma a evitar a convergência prematura para ótimos locais. Em algoritmos evolutivos isso é feito geralmente por meio de compartilhamento de aptidão (do inglês *fitness sharing*) e métodos de nicho (Sareni e Krahenbuhl, 1998).

Compartilhamento de aptidão é uma metodologia aplicada principalmente em algoritmos genéticos em que a aptidão de cada indivíduo é influenciada não só pela função-objetivo mas, também, pelos indivíduos com aptidões parecidas. Nesse caso, como se quer manter diversidade, reduz-se o valor

dessa aptidão proporcional à distância em relação à aptidão dos indivíduos mais próximos. A fórmula de aptidão então se torna:

$$F(x) = \frac{f(x)}{m(x)}, \tag{15.23}$$

onde $F(x)$ é a aptidão calculada por meio do compartilhamento de aptidão, $f(x)$ é a aptidão original e $m(x)$ é a chamada contagem de nichos, a qual indica em quanto a aptidão será reduzida.

O cálculo da contagem de nichos é baseado em uma medida de distância capaz de apontar o quão próximos são dois valores de aptidão. Note aqui que não se está calculando a distância das soluções, mas sim do valor de suas funções-objetivo. A fórmula para contagem de nichos é dada por:

$$m(x) = \sum_{i=1}^{N} sh(dist(f(x), f(x_i)), \tag{15.24}$$

onde N é o tamanho da população, $dist(.,.)$ é a diferença entre dois valores de aptidão e $sh(.)$ é a medida de compartilhamento, a qual deve retornar 1 se os valores de aptidão dos dois indivíduos forem idênticos e 0 se a diferença entre eles for maior que um limiar. Uma forma comum de calcular essa medida é dada por:

$$sh(d) = \begin{cases} 1 - (d/\sigma_s)^{\alpha} & \text{se } d < \sigma_s \\ 0 & c.c. \end{cases}, \tag{15.25}$$

onde d é a distância entre os dois valores de aptidão, σ_s é o limiar de distância e α é um parâmetro que regula o formato da função de compartilhamento.

Apesar de inicialmente ser concebida para manter a diversidade de aptidão na população, essa técnica também pode ser utilizada para manter a diversidade de soluções, substituindo a medida de distância entre os valores de aptidão por uma medida de distância entre soluções, no espaço de busca.

Outra forma de nicho é conhecida como *crowding* e funciona de forma semelhante à substituição de indivíduos da população por novos indivíduos, descrito anteriormente no item de geração de diversidade. A diferença é que esse processo é feito constantemente e os indivíduos escolhidos para substituição são aqueles que têm uma similaridade alta com qualquer outro indivíduo. Essa escolha pode ser feita deterministicamente em toda a população ou em um subconjunto da população.

Finalmente, cabe mencionar também a técnica de *clearing*, similar ao compartilhamento de aptidão na formação dos nichos. Essa técnica preserva a aptidão original dos k melhores indivíduos de cada nicho, definidos pela distância entre eles em termos de valor de aptidão ou em termos de suas posições no espaço de busca, e reduz a aptidão de todos os outros. Dessa forma o algoritmo ainda mantém uma certa quantidade de bons indivíduos, mesmo que parecidos, para não perder o potencial de evolução, mas ao mesmo tempo desfavorece a manutenção de muitos indivíduos similares.

Um outro método que cabe mencionar é a chamada seleção termodinâmica (Mori et al., 1996, 1998) que procura manter a diversidade na etapa de seleção, ao invés de alterar a aptidão do indivíduo. Para tanto, esse método utiliza uma equação chamada de medida de energia livre, definida pela Eq. 15.26 para problemas de minimização. Os indivíduos da geração atual são então selecionados um a um para participarem da próxima geração de forma a minimizar o seguinte critério:

$$F = \langle E \rangle - TH, \tag{15.26}$$

onde F é o valor da energia livre da população, $\langle E \rangle$ é a média da aptidão da população, H é uma medida de diversidade e T, chamado de temperatura, é um controle da taxa de diversidade desejada para o problema.

O termo H, referente à diversidade, é calculado baseado na frequência com que cada indivíduo, ou parte dele, aparece na população. Para o caso de codificação binária, os autores definiram essa equação como sendo:

$$H = \sum_{i=1}^{D} H_i, \tag{15.27}$$

onde D é a dimensão do problema e H_i é dada na forma:

$$H_i = - \sum_{j \in \{0,1\}} p_j^i \log p_j^i, \tag{15.28}$$

com p_j^i sendo a frequência de ocorrência do valor j (0 ou 1, nesse caso) na posição i do vetor solução, calculada pela razão do número de ocorrências desse valor pelo tamanho atual da população já selecionada para a próxima geração.

Outras formas de manutenção de diversidade são propostas por Coelho et al. (2009) para o algoritmo evolutivo denominado evolução diferencial, onde a diversidade das soluções é monitorada e, dependendo do quanto cada indivíduo contribui para a diversidade, ele pode sofrer tipos diferentes de mutação.

Memória de Soluções

O uso de memória, em um primeiro momento, pode remeter apenas aos casos em que se tem um ambiente que muda de forma cíclica, ou seja, de tempos em tempos a solução ótima retorna a posições já visitadas. No entanto, o uso de memória pode ser útil também em casos em que o ótimo se aproxima de situações já experimentadas anteriormente, além de permitir manter a diversidade da população, ao preservar indivíduos que exerceram papéis relevantes num passado recente. O uso de memória pode ser classificado como explícita, quando se tem uma forma bem definida de como e quando armazenar e restaurar a memória, e implícita, usando, por exemplo, o conceito de dominância entre soluções alternativas.

Memória Explícita

Um método mais direto para fazer uso da memória, proposto por Ramsey e Grefenstette (1993), é utilizar uma base de soluções para armazenar os melhores indivíduos em diferentes situações. De tempos em tempos, o melhor indivíduo da população é armazenado juntamente com dados que permitam identificar o estado atual do espaço de busca. Toda vez que uma mudança no ambiente é percebida, o algoritmo é reinicializado com metade da população sendo obtida através da base de soluções, sendo que são selecionados aqueles indivíduos que foram melhores em situações mais parecidas com a atual. Esse tipo de metodologia implica que existe uma forma de comparar as várias configurações do ambiente dinâmico, o que não é viável na maioria dos problemas reais. Por exemplo, para saber se o ambiente retornou a uma configuração já experimentada no passado, podem ser exigidos mais recursos computacionais do que aqueles alocados para se buscar a solução ótima.

Uma abordagem semelhante foi adotada por Louis e Xu (1996), junto a problemas de escalonamento. O algoritmo armazena o melhor indivíduo de tempos em tempos e, ao detectar uma mudança no ambiente de otimização, o algoritmo é reinicializado com parte dos indivíduos pertencentes à memória. Essa metodologia tenta permitir que o algoritmo reinicializado contenha propostas de soluções que tiveram alta qualidade em instantes anteriores do processo de otimização. Espera-se que alguma dessas propostas de solução sirva como um bom ponto de partida para se encontrar o novo ótimo.

Vale citar também o modelo proposto por Trojanowski et al. (1997), que faz com que cada indivíduo tenha uma memória particular. Nessa memória, são armazenados os n últimos ancestrais do indivíduo

e, quando ocorre a mudança no ambiente, todos esses indivíduos e suas respectivas memórias são reavaliados. Após a reavaliação, o melhor entre o conjunto de ancestrais e o próprio indivíduo se torna o indivíduo atual.

Um outro tipo de memória explícita foi utilizado por de França et al. (2005) em um algoritmo de sistemas imunológicos artificiais. Esse algoritmo tem uma natureza multimodal, ou seja, localiza e mantém paralelamente múltiplos ótimos. Para esse algoritmo, toda vez que um indivíduo supostamente encontra algum ótimo local, ele é movido imediatamente para a memória e não passa mais por novas otimizações. Quando o ambiente apresenta uma mudança, os indivíduos da memória são reavaliados e copiados para a população principal, sofrendo novas otimizações em busca de novos ótimos locais. A ideia aqui é de que, ao armazenar múltiplos ótimos locais, o algoritmo está separando amostras distribuídas pelo espaço de busca e, quando ocorre uma mudança, mesmo que o novo ótimo global não esteja armazenado na memória, esse mecanismo irá inserir diversidade de forma distribuída, auxiliando no reinício do algoritmo.

Memória Implícita

Nos métodos de memória implícita, as técnicas mais comuns envolvem o uso de poliploides (Ng e Wong, 1995; Hadad e Eick, 1997) na codificação dos indivíduos. Poliploides são uma generalização do conceito de diploides em algoritmos genéticos (Goldberg e Smith, 1987). Esse tipo de representação conta com múltiplos cromossomos por indivíduo, cada qual associado a um diferente fenótipo. O fenótipo que será expresso é a combinação dos genes dominantes de cada um dos cromossomos do indivíduo. A determinação da dominância dos genes é feita através de um vetor que indica, para cada posição dos cromossomos, qual deles tem o gene dominante. Na hora da reprodução, é feita uma recombinação entre os múltiplos cromossomos de cada indivíduo, gerando indivíduos haploides. Esses indivíduos haploides são agregados através de um processo de seleção, gerando novos indivíduos poliploides.

O uso desse método para ambientes variantes no tempo permite que, ao considerar soluções compostas por múltiplos cromossomos em conjunto com um vetor de dominância, cada indivíduo acabe armazenando diversas soluções distintas, que podem representar memórias passadas de boas soluções. Essas soluções, ao mudar o ambiente, podem assim se tornar dominantes e ajudar na localização do novo ótimo.

Multipopulação

O conceito de multipopulação já vem sendo usado muito antes de ser empregado em propostas para ambientes variantes no tempo. O principal foco de abordagens multipopulacionais é tratar problemas multimodais – ou seja, executar a busca simultânea por múltiplos ótimos (Tsutsui et al., 1997). O uso desse conceito pode ser muito útil em ambientes dinâmicos, pois, se bem elaborado, vai conter implícita ou explicitamente todos os atributos considerados relevantes para algoritmos de otimização em ambientes dinâmicos, brevemente descritos nas seções anteriores.

Um exemplo de multipopulação para algoritmos genéticos em ambientes variantes no tempo foi proposto por Branke et al. (2000). Esta proposta recebeu a denominação de escoteiros auto-organizados (do inglês *self-organizing scouts*) e foi baseada num algoritmo genético de bifurcação (FGA, do inglês *forking genetic algorithm*) (Tsutsui et al., 1997). O processo do FGA funciona da mesma forma que um algoritmo genético comum, mas, quando a população converge para um ótimo local, ocorre uma bifurcação do espaço de busca, de modo que uma população fica responsável pela área onde o algoritmo convergiu e uma outra população por todo o resto do espaço. Esse processo gera dois espaços de busca disjuntos. Isso torna a busca pelo ótimo global mais ampla. No entanto, quando se trata de ambientes variantes no tempo, as divisões já realizadas no espaço de busca podem não refletir a atual configuração de regiões promissoras para a busca, fazendo com que algumas subpopulações fiquem

restritas a realizar uma busca onde já não existem boas soluções para serem localizadas. Outra consequência da frequente variação do ambiente é a possível ocorrência de muitas bifurcações do espaço de busca, criando uma dificuldade de alocação de recursos computacionais para um grande número de partições do espaço de busca.

No algoritmo de escoteiros auto-organizados, o autor tenta tratar esses aspectos do FGA com as seguintes alterações:

- O passo de mutação de cada sub-população é restrito à partição do espaço de busca em que ela se encontra.
- Todos os indivíduos são reavaliados a cada iteração.
- Cada sub-população é formada por um indivíduo central (o melhor indivíduo) e um raio de busca. Todos os indivíduos no raio de busca pertencerão a essa sub-população.
- Conforme o melhor indivíduo de cada população se altera, o centro da partição do espaço de busca o acompanha, promovendo partições dinâmicas do espaço de busca.
- É promovida uma redistribuição de indivíduos entre as sub-populações, de forma que as regiões mais promissoras tenham mais indivíduos e o tamanho total da população permaneça fixo.

Outra abordagem multipopulacional é o chamado algoritmo genético multinacional (do inglês *multinational genetic algorithm*) (Ursem, 2000). Nesse algoritmo, as populações são denominadas *nações* e cada uma delas tem um *governo* formado pelos indivíduos mais representativos (nesse caso, os indivíduos com melhor avaliação dentro da nação). Finalmente, esses governantes definem o sub-espaço de busca para a sua nação, conforme ilustrado na figura 15.3, onde são destacadas as curvas de nível da superfície de otimização. O agrupamento dos indivíduos é feito através de um procedimento de detecção de vales e montanhas ao longo da superfície de otimização. Dados dois pontos no espaço de busca, calcula-se a aptidão de algumas amostras situadas na linha formada entre eles. Um vale é detectado quando o *fitness* de uma dessas amostras é menor que aquele associado aos dois pontos. Esse procedimento é usado em três momentos nesse algoritmo: na migração dos indivíduos entre nações, na criação de novas nações e na união de duas nações.

De forma similar ao algoritmo de escoteiros auto-organizados, em que existe uma população-pai que controla a criação das outras populações, o algoritmo genético de balanço variável (SBGA, do inglês *shifting balance genetic algorithm*) (Wineberg e Oppacher, 2000) define uma população central que tem a responsabilidade de explorar as regiões promissoras, enquanto as sub-populações, denominadas *colônias*, têm a função de procurar novas regiões que ainda não foram exploradas. O grande diferencial desse algoritmo é em como ele mantém as sub-populações longe umas das outras. Para isso, ele usa uma medida de distância entre grupos, representada pela média da distância entre cada indivíduo de uma sub-população e todos os indivíduos pertencentes a uma outra sub-população. A partir desse cálculo, é determinada a porcentagem de sobreposição entre duas sub-populações. Essa informação é utilizada na fase de seleção de indivíduos para a próxima geração, a qual visa minimizar essa sobreposição.

População Auto-ajustável

Um problema encontrado com o uso de multipopulação está no tamanho total da população junto a cada problema de otimização. Se esse número for muito pequeno, existirão apenas poucos indivíduos por sub-população, o que pode fazer com que o algoritmo perca sua capacidade de exploração, a qual deveria ser o ponto forte da abordagem multipopulacional. Por outro lado, se esse número for muito grande, o custo de otimização por iteração tende a ser muito alto e, dessa forma, torna a busca pelo ótimo mais lenta, particularmente quando o ambiente varia a taxas elevadas.

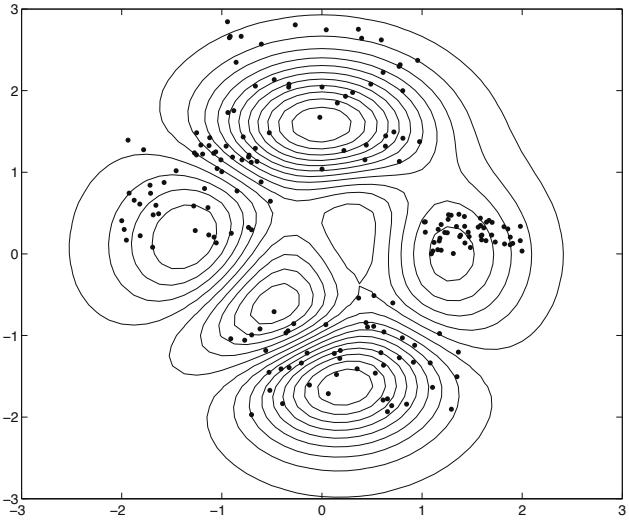


Figura 15.3: Ilustração do comportamento de um algoritmo genético multinacional em um espaço de busca bidimensional, onde são apresentadas as curvas de nível da superfície de otimização (informação esta não disponível para o algoritmo de busca). Cada vale detectado (ótimo local) contém uma população que se separa das demais. No caso da figura, são apresentadas quatro nações diferentes, sendo que a sub-população mais abaixo está identificando dois ótimos locais distintos.

A solução para isso é contar com um controle automático do tamanho da população, de forma que o algoritmo aumente o tamanho da população quando detectar a necessidade de explorar mais regiões em paralelo, ou diminua seu tamanho quando houver redundância ou necessidade de concentrar a busca em determinadas regiões. Um algoritmo que tem essa capacidade inerente em seu funcionamento é a já citada *aiNet* (de Castro e Von Zuben, 2001), que implicitamente conecta todos os seus indivíduos em uma rede, onde as ligações entre dois indivíduos podem ser identificadas pela distância entre eles. Note que, nesse algoritmo da área de sistemas imunológicos artificiais, cada indivíduo representa uma sub-população e, diante de uma situação controlada (ex.: convergência), os indivíduos que estão muito próximo de outros são eliminados e, em fases transitórias da busca, novos indivíduos são inseridos dentro da população. Dessa forma, cada região do espaço de busca tende a ser explorada por apenas uma sub-população de tamanho auto-ajustável. Para mais detalhes consulte o capítulo 6, que aborda os Algoritmos Imunoinspirados.

Outro ajuste automático do tamanho da população foi concebido em (Blackwell, 2007), onde um PSO multipopulacional define o número de populações em uma dada iteração t através da seguinte regra:

$$M(0) = 1, \tag{15.29}$$

$$M(t) = \begin{cases} M(t-1)+1 & \text{se } M_{livre} = 0 \\ M(t-1)-1 & \text{se } M_{livre} > n_{excesso} \end{cases}, \tag{15.30}$$

onde $M(.)$ é o número de populações na iteração corrente, M_{livre} é o número de populações livres, que não identificam ótimos ou são redundantes, e $n_{excesso}$ é um limiar para controlar o número desejado de populações livres.

Alteração da Mutação

Em algoritmos de computação evolutiva específicos para otimização em ambientes contínuos, a aplicação da mutação requer a definição de um tamanho de passo para cada iteração. Esse parâmetro, mesmo em ambientes estáticos, é difícil de ser determinado. Geralmente, é desejável um tamanho de passo grande quando ainda se está longe do ótimo, e que esse passo diminua com a proximidade do ótimo. No caso de ambientes variantes no tempo, essa questão é ainda mais desafiadora. Além da dificuldade de se definir uma medida de proximidade do ótimo, o ótimo pode se deslocar de forma arbitrária e repentina. Em (Rossi et al., 2007), foram propostas duas mutações auto-adaptativas para tratar desse problema, considerando que as mudanças seguem um padrão não-aleatório. Essas mutações utilizam uma técnica de detecção de movimento conhecida como filtro de Kalman (Kalman, 1960), utilizado geralmente para estimar o estado completo de um sistema dinâmico através de observações de um subconjunto de variáveis de estado e dispondo de um modelo do sistema. Uma dessas mutações gera um novo indivíduo através de uma distribuição gaussiana centrada na predição de onde o ótimo estará na próxima iteração, segundo o filtro de Kalman. A outra mutação modifica a mutação gaussiana, substituindo o valor estimado pela distribuição gaussiana existente por uma perturbação gaussiana de média 0 em torno da nova previsão do ótimo.

Estratégias para teste de desempenho

Diferente dos outros ambientes incertos estudados, o ambiente variante no tempo pode contar com alterações drásticas da superfície de otimização, incluindo até mudanças estruturais como a variação na dimensão do espaço de busca. Para tanto, diversas estratégias para teste de desempenho de um algoritmo em tais ambientes foram concebidas.

O ambiente de teste mais tradicional para problemas contínuos e, consequentemente, mais utilizado na maioria das aplicações de otimização em ambientes dinâmicos, está vinculado ao problema de picos móveis (Branke, 1999; Morrison e De Jong, 1999). Esse problema consiste em, dado um número pré-definido de picos (ótimos locais), encontrar e perseguir os picos enquanto eles mudam sua altura, largura e localização. Existem diversos parâmetros que podem ser definidos para tornar o problema mais desafiador, e um programa que cria instâncias desse ambiente de teste pode ser encontrado em *http://www.aifb.uni-karlsruhe.de/ jbr/MovPeaks* e *http://www.cs.uwyo.edu/ wspears/morrison/DF1-Overview.html*.

Para problemas multiobjetivos, foi proposta por Farina et al. (2004) uma sistemática bem completa para reutilizar problemas estáticos em ambientes variantes no tempo. Para tanto, esses ambientes foram divididos em quatro tipos:

- Os locais dos ótimos mudam, mas seus valores não.
- Tanto os locais dos ótimos quanto seus valores mudam com o tempo.
- Os valores mudam, mas os locais dos ótimos não.
- A nova superfície de otimização sofre mudanças de grande monta e arbitrárias, as quais impedem o estabelecimento de equivalências com a superfície de otimização anterior, como nos três casos anteriores.

Uma vez definidos esses tipos de ambientes dinâmicos, foram propostas as adaptações necessárias que devem ser feitas nas funções-objetivo do problema para que ele expresse o comportamento descrito.

Recentemente, no congresso *2009 IEEE Congress on Evolutionary Computation* (IEEE CEC 2009), uma nova metodologia para avaliação foi proposta com base em uma competição dentro do próprio congresso. Essa metodologia procurou ser mais completa e desafiadora que as outras já existentes, separando os casos de ambientes variantes no tempo em sete tipos:

- Deslocamentos pequenos nos ótimos.
- Deslocamentos grandes nos ótimos.
- Deslocamentos definidos por uma distribuição gaussiana.
- Mudanças provocadas por dinâmicas caóticas, logo imprevisíveis a médio e longo prazos.
- Mudanças periódicas e cíclicas.
- Mudanças periódicas e cíclicas com ruído.
- Mudanças estruturais, incluindo alteração na dimensão do problema.

Esses sete tipos foram incorporados em seis funções desafiadoras, originalmente estáticas. O desafio nessa competição é de não só o algoritmo estar preparado para reagir à mudança no ambiente, mas também ter um bom desempenho enquanto o ambiente não se altera. Esses problemas podem ser encontrados em *http://www3.ntu.edu.sg/home/EPNSugan/index_files/CEC-09-Dynamic-Opt/CEC09-Dyn-Opt.htm*.

No caso de ambientes discretos, existem também alguns exemplos de ambientes de teste. Em (Mori et al., 1996), foi apresentado, além do algoritmo com seleção termodinâmica, uma proposta de problema da mochila dinâmico, que consiste em diminuir gradativamente o limite de peso da mochila. Como a solução ótima para esse problema geralmente fica próxima a essa restrição de limite, isso faz com que a solução atual torne-se infactível ao ocorrer uma mudança, tornando o problema ainda mais desafiador.

Outra proposta em ambientes discretos é a descrita por de França et al. (2006) para o Problema do Caixeiro Viajante (PCV). Nesse artigo, instâncias estáticas amplamente conhecidas para o PCV são alteradas periodicamente, de forma que uma porcentagem da distância entre cidades é alterada de forma controlada, utilizando um procedimento de λ-opt (Glover e Kochenberger, 2003).

Outras Aplicações e Técnicas

Problemas variantes no tempo são de particular interesse em filtragem de sinais com canais variantes no tempo (Krusienski e Jenkins, 2004; Junqueira et al., 2005, 2006). Esses filtros são utilizados principalmente em transmissão e recepção de sinais sem fio, onde ocorrem interferências temporais das mais variadas formas, como mudança de local das antenas e ruídos externos.

Uma variação do algoritmo de otimização por enxame de partículas (PSO) foi proposta para o problema de sensoriamento de odores (Jatmiko et al., 2006) em robôs móveis. O sensoriamento é afetado por ruídos externos e depende de fatores temporais, como a localização do robô em relação à fonte de odor e a intensidade do odor. O PSO também foi adaptado de outras formas, incluindo a incorporação de reinícios controlados (Carlisle e Dozier, 2000), o emprego de multipopulação (Blackwell e Branke, 2004; Liang e Suganthan, 2005) e mecanismos de manutenção de diversidade ou anti-convergência (Li et al., 2006). Também foram utilizadas técnicas de agrupamento e busca local para aumentar a capacidade de reação do PSO (Li e Yang, 2009).

Outro algoritmo inspirado na natureza que também foi adaptado para casos variantes no tempo é a *dopt-aiNet* (de França et al., 2005; de França e Von Zuben, 2009), que corresponde a uma variação da *aiNet* citada anteriormente, e que incorporou diversos mecanismos para geração e manutenção da diversidade, memória, multipopulação e tamanho auto-ajustável da população, além da alteração de seus operadores de mutação para obter um desempenho melhor durante a otimização.

Algoritmos inspirados em colônia de formigas (*Ant Colony Optimization* - ACO) (Guntsch et al., 2001; Korosec e Silc, 2009) também foram adaptados para otimização em ambientes dinâmicos, tanto para casos contínuos como discretos. As principais alterações nesse algoritmo são feitas diretamente

na chamada matriz de *feromônio*, que é empregada na definição da distribuição de probabilidades de soluções parciais que o algoritmo utiliza para construir uma solução. Uma vez que é detectada uma mudança no ambiente, o algoritmo ativa alguma forma de alteração nessa matriz, visando favorecer novamente a exploração do espaço de busca.

Adicionalmente, conceitos de multipopulação e parâmetros auto-adaptativos foram incorporados em um algoritmo de evolução diferencial (do inglês *differential evolution*) (Brest et al., 2009), sendo este o que obteve o melhor desempenho na competição do *IEEE CEC 09* mencionada na sub-seção anterior.

7. Conclusões

O mundo de hoje é mais complexo e dinâmico e os problemas de otimização de interesse prático tendem a incorporar esta complexidade e esta dinâmica. Além disso, há uma demanda acentuada por automação de processos de tomada de decisão, visando o atendimento de critérios de desempenho mais exigentes. Logo, ao se aplicar algoritmos de otimização para tratar problemas reais, é necessário lidar com incertezas que podem degradar o desempenho de tais algoritmos, o que geralmente não ocorre no caso de simulações experimentais e problemas teóricos bem definidos.

Em face dessa situação, foram descritas neste capítulo abordagens via algoritmos evolutivos para tratar diversos tipos de incertezas comumente encontrados. As iniciativas tendem a tornar o algoritmo mais robusto para lidar com ambientes incertos, geralmente permitindo atenuar os efeitos da incerteza. Cabe mencionar que não se busca eliminar esses efeitos, visto que em muitas situações alguma degradação de desempenho é inevitável.

Primeiramente, foi apresentado o caso de incertezas geradas por ruídos. Quando o algoritmo requisita uma avaliação da função-objetivo, é retornado um valor deslocado em relação ao valor real. Esse tipo de incerteza é comum quando existe imprecisão na medição e quando se trabalha com equipamentos que sofrem interferências externas, por exemplo. Foram apresentadas diversas soluções da literatura, algumas bastante simples e outras mais elaboradas, de forma a minimizar o efeito desse ruído durante a avaliação dos indivíduos da população. Algumas dessas soluções têm como objetivo melhorar a acuidade da medição sem prejudicar o desempenho computacional do algoritmo.

Em seguida, tratou-se de problemas robustos que, diferente dos problemas ruidosos, apresentam um intervalo de valores admissíveis para parâmetros do problema de otimização. Esse tipo de situação surge, por exemplo, em especificações técnicas vinculadas à qualidade de produtos em plantas industriais. Logo, buscam-se soluções de boa qualidade e que têm sua avaliação pouco alterada quando se consideram variações de parâmetros dentro de intervalos admissíveis. Embora tenha sido mostrado que situações desse tipo podem ser abordadas de forma similar aos caso dos problemas ruidosos, existem também soluções dedicadas, geralmente mais elaboradas e mais custosas computacionalmente, que visam aumentar a capacidade do algoritmo em encontrar soluções robustas.

A próxima incerteza apresentada está associada à dificuldade de se obter o valor da função-objetivo, requerendo o emprego de técnicas de aproximação. Esses cenários se dão quando o processo de avaliação da função-objetivo é custoso computacionalmente ou então em situações em que a função-objetivo não pode ser modelada matematicamente, de forma completa ou precisa. Amostragem de valores de parte da população dos algoritmos de otimização e o emprego conjunto de alguma técnica não-paramétrica de aproximação de funções ilustram bem o enfoque das metodologias de tratamento de incertezas neste caso.

Finalmente, como último caso de incerteza, foram apresentados os problemas de otimização em ambientes dinâmicos, em que se apresenta uma superfície de otimização que muda suas propriedades com o passar do tempo. Existem diversos tipos de variação dinâmica que podem ser considerados. Para solucionar esse problema, foram apresentadas características pertinentes que um algoritmo evolutivo deve exibir, visando detectar e reagir a mudanças. Foi mostrado também que alguns algoritmos já

possuem tais mecanismos de forma inerente. Também foi dado destaque ao compromisso existente entre o uso desses mecanismos e o custo computacional: se um procedimento levar muito tempo para ser realizado, a superfície de busca pode mudar nesse meio tempo, reduzindo o desempenho geral do algoritmo. Por fim, foram apresentados os resultados mais recentes e tendências futuras a serem investigadas.

O propósito fundamental desse capítulo foi o de apresentar ao leitor os diversos tipos de incertezas que podem ocorrer em problemas de otimização que admitem tratamento via algoritmos evolutivos e outras meta-heurísticas populacionais. Foi dada ênfase em procedimentos que permitem adaptar esses algoritmos aos diversos tipos de incertezas, visando atenuar os seus efeitos. Por se tratar de uma leitura introdutória, o leitor é convidado a consultar as referências bibliográficas contidas nesse capítulo para obter um conhecimento mais aprofundado sobre otimização em ambientes incertos, particularmente empregando algoritmos evolutivos.

CAPÍTULO 16

Tomada de Decisão em Ambiente Multiobjectivo

Carlos Henggeler Antunes[a,c] *Maria João Alves*[b,c] *João Clímaco*[b,c]

[a] *Departamento de Engenharia Electrotécnica e de Computadores*
Universidade de Coimbra

[b] *Faculdade de Economia*
Universidade de Coimbra

[c] *Instituto de Engenharia de Sistemas e Computadores de Coimbra*

O mais forte argumento em favor da utilização de modelos e métodos para apoio à decisão em presença de critérios múltiplos reside na própria realidade: os problemas reais são intrinsecamente multidimensionais, sendo expectável que a consideração explícita de múltiplos eixos de avaliação do mérito das soluções potenciais concorra para a obtenção de soluções mais adequadas. A complexidade dos problemas reais que surgem nas sociedades tecnológicas modernas é essencialmente caracterizada pela pluralidade de perspectivas de análise, reflectindo aspectos económicos, sociais, políticos, físicos, de engenharia, administrativos, psicológicos, éticos, estéticos, etc, num dado contexto. Dado que, em geral, não existe uma solução admissível (face à tecnologia, aos recursos disponíveis e aos requisitos exigidos) que garanta o melhor valor em todos esses múltiplos aspectos de avaliação, o processo de tomada de decisões neste tipo de problemas será, na maioria das circunstâncias, redutor se o circuns-

crevermos à procura da solução óptima para um desses objectivos ou para algum objectivo "agregado", como seja a eficiência do ponto de vista económico medida por um qualquer indicador.

Os modelos para apoio à tomada de decisões, geralmente sob a forma de modelos de programação matemática, bem como a percepção dos problemas por parte dos decisores, tornam-se, em geral, mais realistas se os diferentes aspectos da realidade forem explicitamente considerados, construindo uma família coerente de critérios para avaliar as alternativas admissíveis. No entanto, note-se que os argumentos em favor da utilização de modelos que consideram explicitamente múltiplos aspectos de avaliação das soluções potenciais não se esgotam em argumentos de natureza realista. As abordagens multicritério comportam um valor acrescentado, quer na fase de estruturação do problema e construção dos modelos, quer na fase de análise crítica dos resultados, ao permitir descobrir um leque de soluções com diferentes características, e não apenas uma solução óptima, estabelecendo distintos compromissos entre os aspectos de avaliação. A agregação de preferências multicritério contribui ainda, pelo menos parcialmente, para a inclusão nos modelos de investigação operacional (IO) da decisão em grupo e da negociação, presentes em muitos casos reais mas historicamente ausentes dos modelos tradicionais de IO.

De uma forma geral, subdividem-se os problemas multicritério em problemas multiatributo e multiobjectivo. O problema multiatributo refere-se usualmente aos casos em que as acções potenciais, em número finito, são explicitamente conhecidas, bem como os respectivos índices de mérito avaliados segundo os vários critérios (definição enumerativa). Um exemplo típico é o problema de decisão que consiste na escolha de um automóvel, de entre as alternativas disponíveis num dado mercado, em que preço, consumo em cidade e em estrada, espaço, etc., são os critérios que são tidos em conta pelo decisor. Este problema pode ser estruturado através de uma matriz de impactos em que o elemento genérico d_{ij} representa o desempenho da alternativa i segundo o critério j.

Para além deste problema de escolha poderíamos considerar problemas de ordenação (*ranking*), cujo resultado seria a ordenação - eventualmente incompleta e/ou admitindo *ex-aequo* - das alternativas da melhor para a menos boa, ou problemas de categorização (*sorting*), cujo resultado seria a afectação de cada alternativa a uma categoria de um conjunto ordenado de categorias previamente definido. (Roy, 1996) vai um pouco mais longe no que se refere à flexibilidade do apoio multicritério à decisão, quando admite que em muitos casos a análise dos problemas pode conduzir apenas ao aclaramento do processo de decisão e/ou ao aconselhamento do decisor, não havendo lugar a propostas de natureza prescritiva.

O problema de optimização multiobjectivo refere-se aos casos em que as acções potenciais são definidas implicitamente por um conjunto de restrições (definição analítica). O espaço das acções potenciais (espaço de decisão) é mapeado no espaço dos objectivos, no qual cada solução (i.e., uma instanciação das variáveis de decisão) tem como representação um vector, cujos componentes são os correspondentes valores de cada função objectivo. Daqui decorre a designação comum de problemas de optimização vectorial. Este capítulo é dedicado a este segundo tipo de problemas, focando sobretudo os problemas de programação linear multiobjectivo (PLMO) mas salientando também algumas características importantes de problemas lineares com variáveis inteiras e problemas não lineares.

Os termos critério e objectivo serão usados de forma intermutável, embora pressupondo a distinção anterior, geralmente adoptada na literatura.

Perante a existência de múltiplas funções objectivo, a noção de solução óptima (que conduz ao melhor valor admissível para uma única função objectivo) cede lugar à noção de solução não dominada. Uma solução não dominada (também designada por óptima de Pareto, eficiente ou não inferior; uma distinção mais formal entre estes termos será feita mais à frente) caracteriza-se por não existir outra solução admissível (i.e., satisfazendo um dado conjunto de restrições) que melhore simultaneamente todos os objectivos: a melhoria numa função objectivo só pode ser alcançada à custa da degradação

do valor de pelo menos uma das outras funções objectivo do modelo matemático.

Sendo o conceito essencial em optimização multiobjectivo, o conceito de solução não dominada é "pobre" no sentido em que é pouco discriminante. Ou seja, as soluções não dominadas são não comparáveis entre si e, portanto, esta operação de comparação não fornece uma recomendação final de qual adoptar como solução final do problema.

Na resolução de um problema com uma única função objectivo o processo de pesquisa da melhor solução é puramente técnico, no sentido em que a solução óptima está implícita no modelo, cabendo ao algoritmo de optimização a sua determinação. Assim, não há lugar à tomada de decisões dado que esta solução é inequivocamente o plano a adoptar face ao modelo matemático instanciado com um conjunto de dados. Num problema multiobjectivo torna-se necessário fazer intervir no processo de pesquisa não apenas meios técnicos de calcular soluções não dominadas mas também informação sobre as preferências do decisor, que enriqueçam o conceito de solução não dominada de modo a permitir discriminar entre elas. A estrutura de preferências do decisor representa um conjunto de opiniões, valores, convicções e perspectivas da realidade em causa, que configuram um modelo pessoal da realidade sobre o qual aquele se apoia para avaliar diferentes possibilidades de acções potenciais. Ou seja, não é possível classificar uma solução como boa ou má apenas com referência ao modelo matemático e às técnicas de resolução: a qualidade de uma decisão é influenciada pelos aspectos organizacionais, políticos, culturais, etc., subjacentes ao processo de decisão (Roy, 1990).

O problema multiobjectivo pode então colocar-se como a escolha, de entre os elementos do conjunto de soluções não dominadas, de uma que constitua a uma solução de compromisso aceitável pelo decisor tendo em atenção as suas preferências, que podem evoluir ao longo do processo de apoio à tomada de decisões. Este processo é entendido como uma entidade dinâmica, constituída por ciclos interactivos (muitas vezes iterativos) de geração de acções potenciais, avaliação, interpretação de informação, alterações de valores, aprendizagem e adaptação de preferências. Em certos casos o modelo matemático multiobjectivo pode servir apenas para seleccionar um número limitado de soluções não dominadas, com vista a uma análise mais detalhada a posteriori, tendo em conta dimensões do problema, em muitos casos de natureza qualitativa, não contempladas no modelo. Note-se que por vezes não se seleccionam para análise a posteriori apenas soluções não dominadas do problema original, mas também soluções ligeiramente dominadas, consideradas indiferentes face àquelas. De facto, faz todo o sentido não excluir de um estudo mais detalhado, incluindo outros vectores de análise, soluções que são apenas matematicamente dominadas por alguma das seleccionadas para esse estudo, sendo de facto indiferentes do ponto de vista prático no que se refere aos valores das funções objectivo do problema original.

Na secção 2 são apresentadas as noções básicas num contexto de programação linear multiobjectivo. Na secção 3 são apresentadas algumas extensões a modelos de programação linear multiobjectivo com variáveis inteiras e programação não linear multiobjectivo. Os processos de cálculo de soluções eficientes usando funções escalares substitutas são descritos na secção 4. Na secção 5 são sucintamente descritos alguns métodos para tratar problemas de programação linear multiobjectivo, nalguns casos facilmente adaptáveis a problemas lineares com variáveis inteiras e problemas não lineares. Presta-se especial atenção às abordagens interactivas. Como ilustração do funcionamento de um método interactivo, para programação linear multiobjectivo, é descrito na secção 6 o funcionamento do método TRIMAP - programação linear TRI-objectivo - Método de Aprendizagem Progressiva das soluções não dominadas - recorrendo a um exemplo ilustrativo. Na secção 7 são brevemente apresentadas vias de integração entre métodos em sistemas de apoio à decisão, visando tirar partido das características particulares de cada método. Na secção 8 são mencionados alguns casos de aplicação com envolvimento dos autores.

1. Programação Linear Multiobjectivo - Formulação e Conceitos Fundamentais

O problema de programação linear com múltiplas funções objectivo consiste na optimização de p funções objectivo lineares sujeitas a um conjunto de restrições lineares:

$$\begin{array}{l} \max z_1 = f_1(\mathbf{x}) = \mathbf{c}_1\,\mathbf{x} = \sum_{j=1}^{n} c_{1,j}\,x_j \\ \\ \max z_2 = f_2(\mathbf{x}) = \mathbf{c}_2\,\mathbf{x} = \sum_{j=1}^{n} c_{2,j}\,x_j \\ \\ \ldots \\ \\ \max z_p = f_p(\mathbf{x}) = \mathbf{c}_p\,\mathbf{x} = \sum_{j=1}^{n} c_{p,j}\,x_j \\ \\ \text{sujeito a: } \left\{ \begin{array}{ll} \sum_{j=1}^{n} a_{i,j}\,x_j = b_i & i = 1,\ldots,m \\ \\ x_j \geq 0 & j = 1,\ldots,n \end{array} \right. \end{array} \tag{16.1}$$

ou

$$\begin{array}{l} \text{Max } \ \mathbf{z} = \mathbf{f}(\mathbf{x}) = C\mathbf{x} \\ \\ \text{sujeito a: } \mathbf{x} \in \mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq 0, A\mathbf{x} = \mathbf{b}, \mathbf{b} \in \mathbb{R}^m\} \end{array} \tag{16.2}$$

C é a matriz dos coeficientes das funções objectivo (dimensão $p \times n$), cujas linhas são os vectores $\mathbf{c}_k$ (coeficientes da função objectivo $\mathbf{f}_k(\mathbf{x})$). A é a matriz dos coeficientes tecnológicos ($m \times n$), que se admite, sem perda de generalidade, tendo todas as restrições sido convertidas em igualdades através da introdução de variáveis desvio (*slack* e *surplus*) auxiliares. $\mathbf{b}$ é o vector dos termos independentes (genericamente recursos disponíveis para restrições do tipo $\leq$ ou requerimentos para restrições do tipo $\geq$). Assume-se que a região admissível $\mathcal{X}$ é não vazia e compacta. Sem perda de generalidade, e de modo a facilitar a notação, considera-se que as funções objectivo são todas a maximizar.

Enquanto em programação com uma única função objectivo as soluções admissíveis do espaço de decisão $\mathbf{x} \in \mathcal{X}$ são mapeadas em $\mathbb{R}$, no caso multiobjectivo o espaço de decisão é mapeado num espaço p-dimensional $\mathcal{F} = \{\mathbf{z} = \mathbf{f}(\mathbf{x}) \in \mathbb{R}^p : \mathbf{x} \in \mathcal{X}\}$ designado por espaço dos objectivos. Neste espaço cada alternativa potencial $\mathbf{x} \in \mathcal{X}$ tem como representação um vector $\mathbf{f}(\mathbf{x}) = (z_1, z_2, \ldots, z_p) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_p(\mathbf{x}))$ cujos componentes são os valores de cada função objectivo para esse ponto da região admissível (fig. 16.1).

Em geral, não existe uma solução admissível $\mathbf{x} \in \mathcal{X}$ que optimize simultaneamente todas as funções objectivo. De um ponto de vista operacional, "Max"representa a operação de determinar soluções eficientes.

Num contexto multiobjectivo, a noção de solução óptima cede lugar ao conceito de solução eficiente. Uma solução admissível para um problema multiobjectivo diz-se eficiente se e só se (sse) não existir outra solução admissível que melhore o valor de uma função objectivo, sem piorar o valor de, pelo menos, outra função objectivo.

Definição 1 (solução eficiente): *Uma solução $\mathbf{x}' \in \mathcal{X}$ é eficiente sse não existe uma outra solução $\mathbf{x} \in \mathcal{X}$ tal que $f_k(\mathbf{x}) \geq f_k(\mathbf{x}')$ para todo o $k (k = 1,\ldots,p)$, sendo a desigualdade estrita para pelo menos um k, $f_k(\mathbf{x}) > f_k(\mathbf{x}')$. $\mathcal{X}_E$ representa o conjunto das soluções eficientes.*

O ponto $\mathbf{z} = \mathbf{f}(\mathbf{x})$ no espaço das funções objectivo é não dominado (não inferior) sse $\mathbf{x} \in \mathcal{X}_E$, ou seja $\mathcal{F}_E = \{\mathbf{z} = \mathbf{f}(\mathbf{x}) \in \mathcal{F} : \mathbf{x} \in \mathcal{X}_E\}$.

Em geral, enquanto o conceito de não dominância se refere ao espaço das funções objectivo, o conceito de eficiência refere-se ao espaço das variáveis de decisão, isto é a imagem de uma solução

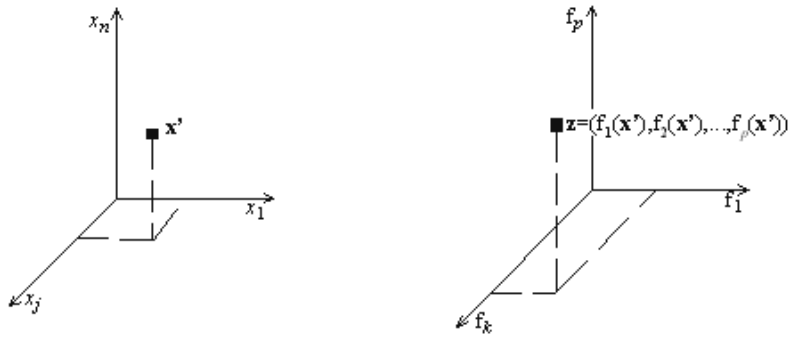


Figura 16.1: O espaço de decisão e o espaço dos objectivos

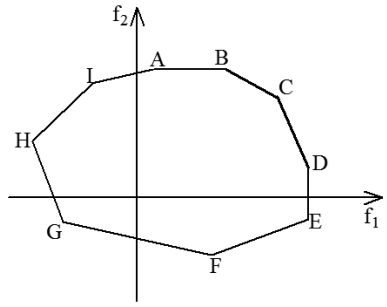


Figura 16.2: Ilustração dos diferentes tipos de soluções.

eficiente é uma solução não dominada. Em problemas de programação matemática multiobjectivo (optimização vectorial), os conjuntos de soluções eficientes e da respectiva imagem no espaço dos objectivos têm geralmente um número infinito de elementos.

Outra noção com interesse é a de solução fracamente eficiente, que pode considerar-se uma relaxação da anterior. Uma solução admissível para um problema multiobjectivo diz-se fracamente eficiente sse não existir outra solução admissível que melhore estritamente o valor de todas as funções objectivo.

Definição 2 (solução fracamente eficiente): *Uma solução* $\mathbf{x}' \in \mathcal{X}$ *é fracamente eficiente sse não existe uma outra solução* $\mathbf{x} \in \mathcal{X}$ *tal que* $f_k(\mathbf{x}) > f_k(\mathbf{x}')$ *para todo o* k $(k = 1, \ldots, p)$. $\mathcal{X}_{FE}$ *representa o conjunto das soluções fracamente eficientes.*

O ponto $\mathbf{z} = \mathbf{f}(\mathbf{x})$ *no espaço das funções objectivo é fracamente não dominado sse* $\mathbf{x} \in \mathcal{X}_{FE}$*, ou seja* $\mathcal{F}_{FE} = \{\mathbf{z} = \mathbf{f}(\mathbf{x}) \in \mathcal{F} : \mathbf{x} \in \mathcal{X}_{FE}\}$.

Note-se que, por definição, o conjunto de soluções fracamente eficientes inclui as soluções (estritamente) eficientes, antes definido. De uma forma geral, por razões de ordem prática, quando se mencionam as soluções fracamente eficientes não se estão a considerar as soluções estritamente eficientes.

A fig. 16.2 ilustra os conceitos de solução não dominada e fracamente não dominada, com duas funções a maximizar: as soluções sobre os segmentos AB e DE são fracamente não dominadas, excepto os pontos B e D, enquanto as soluções sobre os segmentos BC e CD são (estritamente) não dominadas.

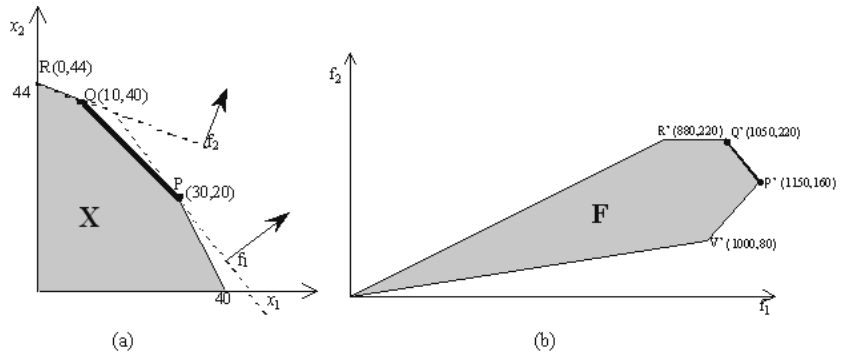


Figura 16.3: Exemplo 1: (a) espaço das variáveis de decisão; (b) espaço das funções objectivo

Para uma melhor ilustração dos conceitos de solução eficiente (não dominada) e fracamente eficiente (fracamente não dominada), considere-se o seguinte exemplo ilustrativo (Clímaco et al., 2003, p. 96).

Exemplo 1:

$$\max f_1(\mathbf{x}) = 25x_1 + 20x_2$$

$$\max f_2(\mathbf{x}) = 2x_1 + 5x_2$$

$$\text{sujeito a: } \begin{cases} x_1 + x_2 \leq 50 \\ 2x_1 + x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 220 \\ x_1, x_2 \geq 0 \end{cases}$$

Na figura 16.3 está representada a região admissível do problema no espaço das variáveis de decisão (a) e no espaço das funções objectivo (b). Como se pode observar, todas as soluções do segmento QR (a que corresponde $Q'R'$ no espaço dos objectivos) optimizam a função objectivo $f_2(\mathbf{x})$, enquanto que apenas o ponto P (P' no espaço dos objectivos) optimiza $f_1(\mathbf{x})$. As soluções do segmento QR, com excepção do ponto Q, são fracamente eficientes. Estas soluções são dominadas por Q que tem valor igual em f_2 e superior em f_1, não havendo nenhuma solução admissível que melhore simultaneamente as duas funções objectivo relativamente a qualquer solução de $[RQ[$. As soluções (estritamente) eficientes deste problema são todos os pontos do segmento QP. No espaço das funções objectivo, os pontos não dominados são os do segmento $Q'P'$.

Note-se que apesar de neste exemplo, bem como no ilustrado na fig. 16.2, as soluções fracamente eficientes estarem sobre segmentos que optimizam uma das funções objectivo, isto não tem que acontecer necessariamente em problemas com mais de duas funções objectivo.

Solução ideal

Designa-se geralmente por solução ideal $\mathbf{z}^*$ (ou ponto utopia) a solução que optimizaria simultaneamente todas as funções objectivo, ou seja, cujas componentes são o óptimo de cada função objectivo na região admissível, quando optimizadas separadamente. Em geral a solução ideal não pertence à

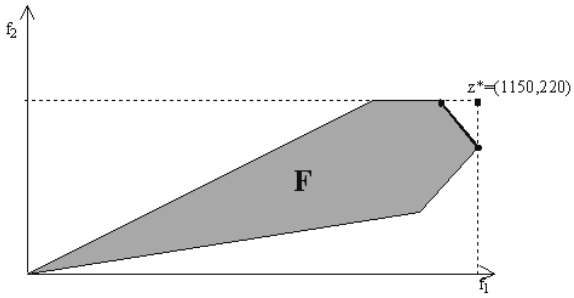


Figura 16.4: Solução ideal do Exemplo 1.

região admissível (caso contrário o problema seria trivial, pois todas as funções teriam o seu óptimo na solução ideal), embora cada uma das componentes z_k^* da solução ideal seja individualmente alcançável. Note-se que embora se possa definir sempre a solução ideal $\mathbf{z}^*$ no espaço dos objectivos, nem sempre existe a respectiva imagem no espaço de decisão (ou seja, pode não existir $\mathbf{x}^*$, mesmo que não admissível, tal que $\mathbf{z}^* = \mathbf{f}(\mathbf{x}^*)$).

A atitude de procurar uma solução o mais próxima possível da solução ideal está na base de processos de cálculos de soluções eficientes. A solução ideal é muitas vezes usada como o ponto de referência (inatingível) do decisor em funções escalares que representam uma distância a minimizar para determinar uma solução eficiente de compromisso.

Na fig. 16.4 está representada a solução ideal para o problema do Exemplo 1.

2. Programação Linear Inteira e Programação Não Linear Multiobjectivo

A consideração de variáveis inteiras, eventualmente apenas binárias, em modelos de programação linear multiobjectivo, bem como a existência de não linearidades nas funções objectivo e/ou nas restrições requer alguns conceitos adicionais.

A definição de soluções eficientes próprias comporta uma noção mais restrita de solução eficiente de modo a eliminar soluções eficientes que apresentem compromissos ilimitados entre objectivos, ou seja soluções em que a relação melhoria/degradação entre os valores das funções objectivo possa ser feita arbitrariamente grande (Geoffrion, 1968).

Definição 3 (solução eficiente própria): *Uma solução $\mathbf{x}' \in \mathcal{X}$ é eficiente própria se é eficiente e existe um número finito $M > 0$ tal que, para cada $\mathbf{x} \in \mathcal{X}$ e para cada função $f_k(\mathbf{x})$, $k = 1, \ldots, p$ com $f_k(\mathbf{x}) > f_k(\mathbf{x}')$, se verifica*

$$\frac{f_k(\mathbf{x}) - f_k(\mathbf{x}')}{f_j(\mathbf{x}') - f_j(\mathbf{x})} \leq M$$

para algum j em que $f_j(\mathbf{x}) < f_j(\mathbf{x}')$. $\mathcal{X}_{PE}$ representa o conjunto das soluções eficientes próprias.

Em programação linear multiobjectivo $\mathcal{X}_{PE} \equiv \mathcal{X}_E$. Também em programação linear inteira e inteira mista todas as soluções eficientes são eficientes próprias. No entanto, em problemas multiobjectivo não lineares podem existir soluções eficientes não próprias.

A fig. 16.5 ilustra o conceito de solução eficiente não própria a partir da representação no espaço dos objectivos de dois problemas não lineares, com ambas as funções a maximizar. Na fig.16.5(a) as soluções eficientes situam-se nos arcos AB e CD, excluindo o ponto D que é fracamente eficiente. As soluções A, B e C são eficientes não próprias. Na fig. 16.5(b), toda a fronteira de A a C é eficiente,

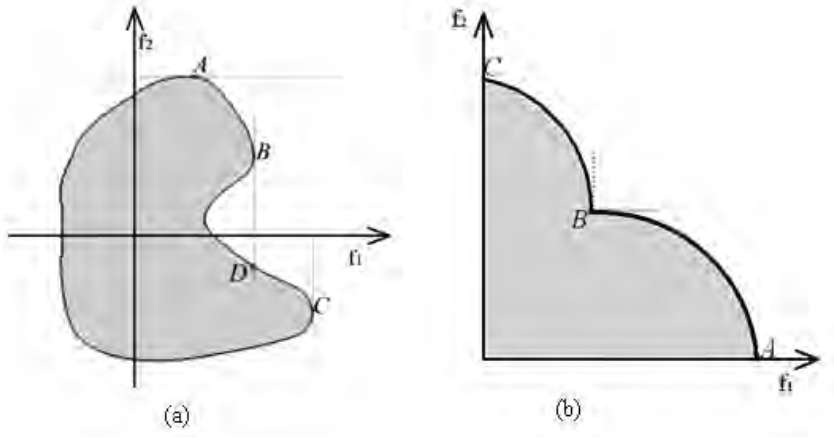


Figura 16.5: Ilustração do conceito de solução eficiente não própria

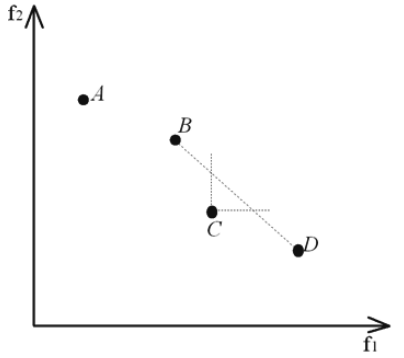


Figura 16.6: Ilustração do conceito de solução eficiente não suportada no caso discreto

passando pelo ponto B que é uma solução eficiente não própria.

Uma outra noção importante é a distinção entre soluções eficientes suportadas e não suportadas.

Definição 4 (solução não dominada/eficiente não suportada): *Um ponto não dominado* $\mathbf{z}' \in \mathcal{F}_E$ *é não suportado se for dominado por uma combinação convexa (não admissível) de pontos pertencentes a* $\mathcal{F}_E$. *A um ponto* $\mathbf{z}' = \mathbf{f}(\mathbf{x}')$ *não dominado não suportado corresponde uma solução* $\mathbf{x}$*' eficiente não suportada. As outras soluções eficientes dizem-se suportadas.*

Em problemas de programação linear multiobjectivo todas as soluções eficientes são suportadas, mas em problemas com $\mathcal{F}$ não convexo podem ocorrer soluções eficientes que sejam não suportadas. Ilustram-se em seguida os casos da programação linear inteira, inteira mista e não linear, multiobjectivo.

A fig. 16.6 ilustra a situação de existência de soluções eficientes não suportadas num problema de programação inteira, com ambas as funções a maximizar. As soluções A, B e D são eficientes suportadas, enquanto que C é uma solução eficiente não suportada porque é dominada por algumas combinações convexas - não admissíveis - de B e D (todas as definidas pela intersecção do cone que emana de C com o segmento que une B e D).

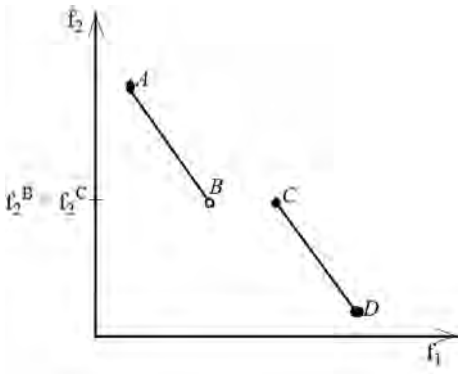


Figura 16.7: Soluções eficientes não suportadas num problema de programação inteira mista

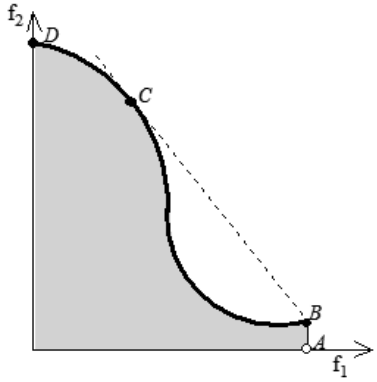


Figura 16.8: Soluções eficientes não suportadas num problema de programação não linear

A fig. 16.7 mostra a fronteira não dominada de um problema de programação linear inteira mista, com ambas as funções a maximizar. A solução A e todas as soluções do segmento CD são eficientes suportadas; as soluções do segmento AB, excluindo os pontos A e B, são eficientes não suportadas porque são dominadas por combinações convexas (não admissíveis) de A e C; a solução B é fracamente eficiente porque não existe nenhuma outra solução que seja estritamente melhor do que esta nas duas funções objectivo, mas é dominada pela solução C que tem valor igual em f_2 e valor superior em f_1.

A fig. 16.8 representa um problema não linear, em que a fronteira não dominada está representada a traço mais grosso (de B a D). As soluções no segmento AB, excluindo B, são soluções fracamente eficientes; as soluções desde B (exclusive) até a C (exclusive) são eficientes não suportadas; a solução B e as soluções do arco CD são eficientes suportadas.

3. Processos de Cálculo de Soluções Eficientes em Programação Linear Multiobjectivo

Nesta secção são abordadas três processos de cálculo de soluções eficientes através da optimização de funções escalares substitutas (funções escalarizantes), que agregam temporariamente numa única dimensão as p funções objectivo originais do modelo incluindo ainda parâmetros de informação das preferências do decisor. A solução óptima da função escalarizante deve ser uma solução eficiente (embora por vezes apenas possa ser garantida a obtenção de soluções fracamente eficientes) do problema

multiobjectivo.

Existem diversos tipos de funções escalarizantes, às quais se requerem normalmente propriedades objectivas como sejam:

- Devem gerar apenas soluções eficientes;
- Devem poder gerar todas as soluções eficientes;
- Devem ser independentes de soluções não eficientes, ou seja, não devem ser sensíveis a alterações da região admissível não eficiente;

bem como propriedades subjectivas, tais como:

- O esforço computacional envolvido não deve ser muito grande (por exemplo, para um problema linear multiobjectivo a função substituta deve ser linear e a dimensão do problema não deve ser muito aumentada);
- Os parâmetros de preferência, quando existem, devem ter uma interpretação simples e não devem exigir do decisor grande esforço cognitivo.

Qual o significado das funções escalarizantes? Dependendo do contexto, isto é, dos princípios subjacentes aos métodos onde são usadas, podem distinguir-se duas aproximações básicas para o significado das funções escalarizantes:

- Constituirem apenas um mero artifício técnico para agregar temporariamente as múltiplas funções objectivo e gerar soluções eficientes a propor ao decisor (sem preocupação em reflectirem uma expressão das suas preferências);
- Serem encaradas como a representação analítica das preferências do decisor.

A apresentação das formas de escalarização mais comuns procura colocar em relevo as relações existentes entre a formulação matemática da função escalar substituta e a intervenção de parâmetros de preferência.

Optimização de uma das funções objectivo considerando as outras $p-1$ como restrições

O processo mais simples de construir um problema escalar é escolher como função substituta a optimizar uma das funções objectivo (normalmente aquela a que o decisor atribui mais importância), enquanto se estabelecem limitações inferiores nas outras $p-1$ funções objectivo que são tratadas como restrições (representando os níveis mínimos que o decisor está disposto a aceitar):

$$\begin{aligned} &\max z_i = f_i(\mathbf{x}) = \mathbf{c}_i\mathbf{x} \\ &\text{sujeito a: } \begin{cases} f_k(\mathbf{x}) = \mathbf{c}_k\mathbf{x} \geq e_k \quad k = 1,\ldots,p,\ k \neq i \\ \mathbf{x} \in \mathcal{X} \end{cases} \end{aligned} \tag{16.3}$$

Resolvendo (16.3) para algum i e variando os valores das limitações e_k é possível obter todo o conjunto das soluções eficientes.

Demonstra-se que se $\mathbf{x}' \in \mathcal{X}$ for solução óptima única de (16.3) para algum i, com e_k qualquer, então x' é uma solução eficiente do problema multiobjectivo (16.1) (Steuer, 1986).

A optimização desta função escalar substituta garante a obtenção de uma solução eficiente do problema multiobjectivo original (16.1) desde que a região admissível reduzida seja não vazia (o

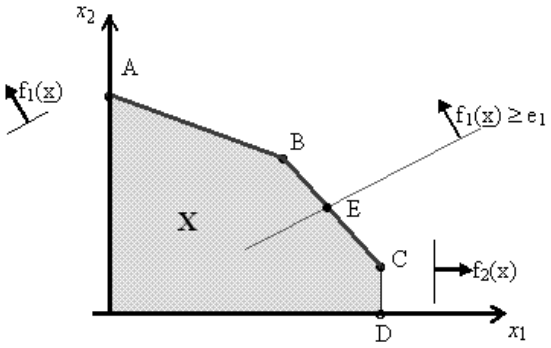


Figura 16.9: Optimização de uma função objectivo restringindo a outra gerando uma solução estritamente eficiente

que pode não acontecer se os níveis mínimos e_k forem demasiado exigentes) e não existam óptimos alternativos da função objectivo escolhida para ser optimizada (caso em que apenas há a garantia de obter pontos fracamente eficientes).

Note-se que, sem a condição de a solução ser única, pode existir um $\mathbf{x}" \in \mathcal{X}$ tal que $f_i(\mathbf{x}") = f_i(\mathbf{x}')$ e $f_k(\mathbf{x}") \geq f_k(\mathbf{x}')$, $k \neq i$, com desigualdade estrita para pelo menos um k, o que apenas garante que $\mathbf{x}'$ seja fracamente eficiente. Juntando um termo de perturbação à função objectivo do problema (16.3), isto é, substituindo $f_i(\mathbf{x}) = \mathbf{c}_i\mathbf{x}$ por

$$\mathbf{c}_i\mathbf{x} + \sum_{k=1,k\neq i}^{p} \rho_k \, \mathbf{c}_k \, \mathbf{x},$$

com $\rho_k > 0$ suficientemente pequenos, garante-se que a solução $\mathbf{x}'$ é (estritamente) eficiente para o problema (16.1).

A fig. 16.9 ilustra este processo de escalarização. A fronteira eficiente é constituída pelas soluções nos segmentos AB e BC. Limitando $f_1(\mathbf{x}) \geq e_1$ e optimizando $f_2(\mathbf{x})$ é obtida a solução eficiente E. Note-se que E é obviamente um vértice da região admissível reduzida, mas não é um vértice da região admissível original.

No caso de existirem soluções óptimas alternativas para (16.3) podem obter-se soluções que não são estritamente eficientes, mas apenas fracamente eficientes. Esta situação é ilustrada na fig. 16.10, em que impondo $f_1(\mathbf{x}) \geq e_1$ e optimizando $f_2(\mathbf{x})$ as soluções no segmento CF são óptimas alternativas para (16.3), mas apenas C é estritamente eficiente.

Um motivo de interesse adicional desta forma de escalarização é que a variável dual associada à restrição correspondente à função objectivo k pode ser interpretada como a taxa de compromisso local entre os objectivos $f_i(\mathbf{x})$ e $f_k(\mathbf{x})$ na solução óptima do problema escalar (16.3). No entanto, há que ter especiais cuidados na interpretação e uso desta informação em situações em que a solução seja degenerada, dado que neste caso estes valores não são únicos (óptimos alternativos do dual).

Embora esta forma de escalarização seja simples de compreender pelos decisores, captando a atitude de dar mais importância a uma função objectivo e aceitando limitações inferiores para as outras, a escolha da função objectivo a optimizar pode revelar-se difícil em muitos problemas. No quadro operacional de um dado método, a fixação da função a optimizar durante todo o processo torna o método pouco flexível e os resultados demasiado dependentes da função seleccionada.

Através do problema (16.3) é possível obter todos os pontos da fronteira não dominada, quer sejam ou não vértices da região admissível do problema original.

A informação de preferências associada a este tipo de escalarização consiste em:

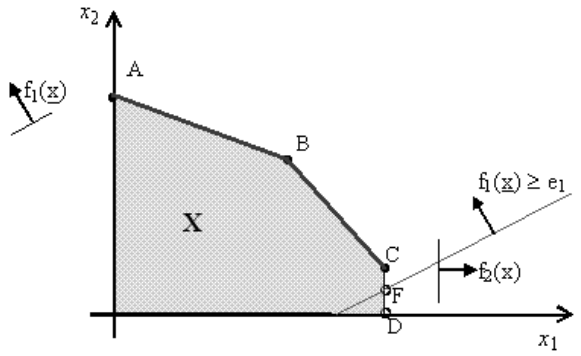


Figura 16.10: Optimização de uma função objectivo restringindo a outra gerando soluções fracamente eficientes

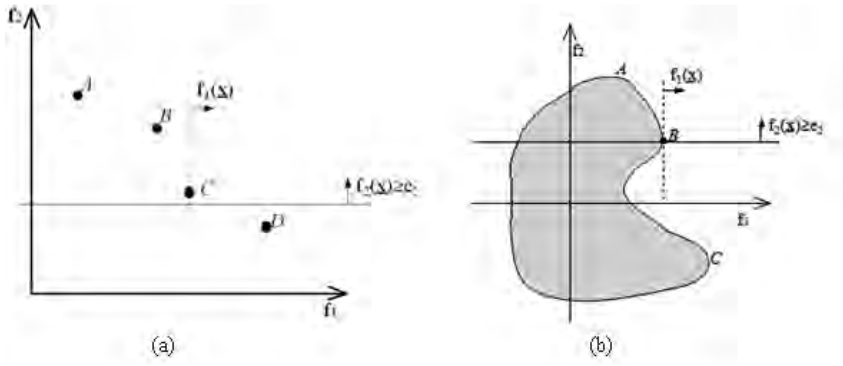


Figura 16.11: Optimização de uma função objectivo restringindo a outra: (a) num problema de programação inteira; (b) num problema não linear

- Informação inter-critérios: escolha da função objectivo a optimizar;
- Informação intra-critérios: imposição de níveis mínimos para as outras funções objectivo.

Este processo de cálculo de soluções eficientes pode também ser aplicado a problemas de programação inteira, inteira mista ou não linear multiobjectivo, permitindo obter qualquer tipo de solução eficiente para estes problemas.

A fig. 16.11 mostra exemplos de programação inteira (a) e de programação não linear (b) bi-objectivo, em que se restringe $f_2(\mathbf{x})$ e se optimiza $f_1(\mathbf{x})$; em (a) obtém-se a solução eficiente C (não suportada) e em (b) obtém-se a solução eficiente B (não própria).

Soma ponderada das funções objectivo

Um dos processos de cálculo de soluções eficientes mais utilizado em programação linear multiobjectivo consiste na resolução de um problema escalar cuja função objectivo é uma soma ponderada das p funções objectivo originais considerando pesos λ_k positivos:

$$\max z_\lambda = \lambda_1 f_1(\mathbf{x}) + \lambda_2 f_2(\mathbf{x}) + \ldots + \lambda_p f_p(\mathbf{x}) = \sum_{k=1}^{p} \lambda_k \, \mathbf{c}_k \, \mathbf{x} \qquad (16.4)$$

sujeito a: $\mathbf{x} \in \mathcal{X}$

O conjunto de pesos admissível é definido por

$$\Lambda^0 = \left\{ \boldsymbol{\lambda} \in \mathbb{R}^p : \lambda_k \geq 0, \ k = 1, 2, \ldots, p, \ \sum_{k=1}^{p} \lambda_k = 1 \right\}$$

e o seu interior por

$$\Lambda = \left\{ \boldsymbol{\lambda} \in \mathbb{R}^p : \lambda_k > 0, \ k = 1, 2, \ldots, p, \ \sum_{k=1}^{p} \lambda_k = 1 \right\}.$$

Fixando um vector de pesos obtém-se uma função linear escalar ponderada das p funções objectivo, a optimizar em $\mathcal{X}$.

Demonstra-se que, em programação linear multiobjectivo, $\mathbf{x}' \in \mathcal{X}$ é uma solução eficiente para (16.1) sse for uma solução óptima para (16.4), para um vector de pesos $\boldsymbol{\lambda} \in \Lambda$ (Steuer, 1986).

Note-se que se podem obter soluções eficientes considerando $\boldsymbol{\lambda} \in \Lambda^0$, com algum $\lambda_k = 0$ (em particular, as soluções que optimizam individualmente cada uma das funções objectivo). Contudo, nesta situação, e caso existam óptimos alternativos, podem existir soluções óptimas de (16.4) que são apenas soluções fracamente eficientes do problema multiobjectivo. Ou seja, $\mathbf{x}'$ é uma solução eficiente para (16.1) se for uma solução óptima para (16.4), com $\boldsymbol{\lambda} \in \Lambda^0$, e uma das condições se verificar:

1. $\boldsymbol{\lambda} \in \Lambda$, ou
2. $\mathbf{x}'$ é solução única de (16.4) com $\boldsymbol{\lambda} \in \Lambda^0$.

A fig. 16.12 (correspondente ao Exemplo 1 apresentado atrás) ilustra este processo de escalarização. Optimizando a soma ponderada das funções objectivo cujo vector gradiente está representado na figura (z_λ) obtém-se a solução Q. Repare-se que, neste exemplo, o uso de peso nulo para $f_1(\mathbf{x})$, isto é, $\lambda_1 = 0$ e $\lambda_2 = 1$, conduz a soluções óptimas alternativas para o problema (16.4) entre as quais existem soluções fracamente eficientes. Esta situação corresponde à optimização de apenas $f_2(\mathbf{x})$. Como se observou atrás, todas as soluções do segmento QR optimizam esta função mas apenas o ponto Q é estritamente eficiente. As outras soluções deste segmento são fracamente eficientes.

O cálculo de soluções fracamente eficientes pode ser evitado substituindo o(s) peso(s) nulo(s) por um valor positivo ϵ bastante pequeno (por exemplo, 10^{-3} ou 10^{-4}). No exemplo representado na fig. 16.12, se for optimizada a soma ponderada $0.001\, f_1(\mathbf{x}) + 0.999\, f_2(\mathbf{x})$ obtém-se apenas a solução eficiente Q.

Da resolução do problema escalar (16.4) para um dado vector de pesos obtém-se sempre pelo menos uma solução básica (vértice) do problema multiobjectivo original, existindo vários vectores de pesos que conduzem à mesma solução básica. No caso do exemplo anterior, todos os vectores de pesos $\boldsymbol{\lambda}$ de $(0,1)$ a $(\frac{3}{8}, \frac{5}{8})$ conduzem ao vértice Q e os vectores de pesos de $(\frac{3}{8}, \frac{5}{8})$ a $(1,0)$ conduzem ao vértice P. Para os pesos $\lambda_1 = \frac{3}{8}$ e $\lambda_2 = \frac{5}{8}$ obtêm-se simultaneamente os pontos P e Q, visto que a optimização da soma ponderada das funções objectivo com este vector de pesos particular conduz ao segmento PQ (óptimos alternativos).

É útil analisar este processo de escalarização usando o quadro simplex multiobjectivo associado à função escalar soma ponderada que é optimizada na região admissível original. O quadro simplex multiobjectivo inicial (obtido acrescentando ao quadro simplex normal uma linha por cada função objectivo) tem a forma:

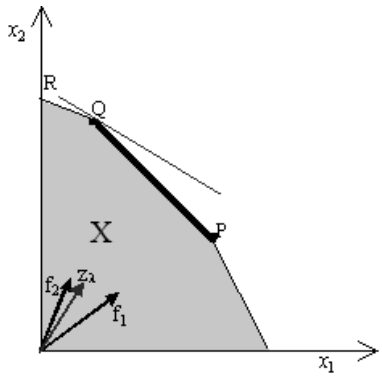


Figura 16.12: Optimização de uma soma ponderada das funções objectivo

A	$\mathbf{b}$
$-C$	0

Relativamente à base B este quadro pode ser transformado em:

$\mathbf{x}_N$	$\mathbf{x}_B$	
$B^{-1}N$	I	$B^{-1}\mathbf{b}$
$C_B B^{-1}N - C_N$	$\mathbf{0}$	$C_B B^{-1}\mathbf{b}$

onde B e N, C_B e C_N são as sub-matrizes de A e de C correspondentes às variáveis básicas ($\mathbf{x}_B$) e não básicas ($\mathbf{x}_N$), respectivamente.

Definição 5 (bases adjacentes): *As bases B_1 e B_2 são adjacentes sse podem ser obtidas uma a partir da outra por uma operação de pivotação.*

Definição 6 (matriz dos custos reduzidos): *A matriz dos custos reduzidos relativa à base B é dada por $W = C_B B^{-1}N - C_N$.*

Definição 7 (base eficiente): *B é uma base eficiente sse for uma base óptima do problema escalar ponderado (16.4), para algum vector de pesos $\boldsymbol{\lambda} \in \Lambda$, isto é, B é uma base eficiente sse o sistema $\{\boldsymbol{\lambda}^T W \geq 0,\ \boldsymbol{\lambda} \in \Lambda\}$ for coerente.*

Definição 8 (variável não básica eficiente): *A variável não básica x_j é eficiente em relação à base B sse existir $\boldsymbol{\lambda} \in \Lambda$ tal que*

$$\boldsymbol{\lambda}^T W \geq \mathbf{0}$$
$$\boldsymbol{\lambda}^T W_{\cdot,j} = 0$$

onde $W_{\cdot,j}$ é a coluna de W correspondente a x_j (ou seja, o custo reduzido de x_j pode ser feito nulo).

A definição 8 significa que, para uma dada base eficiente, se x_j for uma variável não básica eficiente, qualquer pivotação admissível (cujo elemento pivot seja positivo, ou negativo correspondente a uma variável básica degenerada) associada a x_j conduz a uma base eficiente adjacente (ou seja, obtida a partir da anterior por essa operação de pivotação). Se a pivotação que conduz de uma base a outra for não degenerada, os pontos extremos correspondentes a essas bases são diferentes e a aresta que os liga é constituída por soluções eficientes.

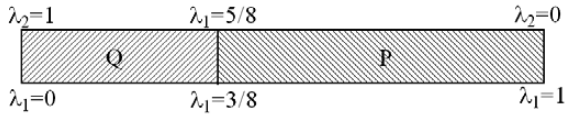


Figura 16.13: O diagrama paramétrico para um problema com 2 funções objectivo (exemplo 1)

Como as bases (vértices) eficientes são conexas é possível construir um método simplex multiobjectivo como extensão do método simplex (ver, por exemplo, (Steuer, 1986)), com o auxílio de sub-problemas de teste de eficiência de variáveis não básicas, que calcula todas as bases (vértices) eficientes, podendo a partir desta informação caracterizar também arestas eficientes não limitadas e faces eficientes (de diferentes dimensões).

A representação gráfica do conjunto de pesos $\boldsymbol{\lambda}$ que conduz a uma solução básica eficiente pode ser obtida através da decomposição do diagrama paramétrico (espaço dos pesos) Λ. Note-se que, devido à condição $\lambda_1+\lambda_2+\ldots+\lambda_p=1$, Λ pode ser representado num diagrama de dimensão $p-1$. Na fig. 16.13 é representado o diagrama paramétrico (unidimensional) de um problema com duas funções objectivo – o Exemplo 1 apresentado atrás (ver fig. 16.12) – na situação em que são conhecidas todas as bases eficientes (o diagrama paramétrico está totalmente preenchido). Na fig. 16.14, o diagrama paramétrico (bidimensional) é relativo a um problema com três funções objectivo e apenas são conhecidas as soluções básicas eficientes que optimizam individualmente cada função objectivo.

A partir do quadro simplex multiobjectivo correspondente a uma solução básica eficiente para (16.1), o conjunto de pesos correspondente é definido por $\left\{\boldsymbol{\lambda}^T W \geq \mathbf{0}, \boldsymbol{\lambda} \in \Lambda\right\}$. O elemento w_{kj} da matriz dos custos reduzidos W representa a variação da função objectivo $f_k(\mathbf{x})$ devido ao incremento unitário da variável não básica x_j que se torna variável básica. Cada coluna de W correspondente a uma variável não básica eficiente representa uma tendência de variação unitária das funções objectivo ao longo da respectiva aresta eficiente.

A região que engloba o conjunto de pesos correspondente a uma solução básica eficiente (região onde $\left\{\boldsymbol{\lambda}^T W \geq \mathbf{0},\ \boldsymbol{\lambda} \in \Lambda\right\}$) designa-se por *região de indiferença*. O decisor pode ser indiferente a todas as combinações de pesos nessa região, porque elas conduzem à mesma solução eficiente (figs. 16.13 e 16.14). Uma fronteira comum a duas regiões de indiferença significa que as respectivas soluções básicas eficientes estão ligadas por uma aresta eficiente, correspondente a uma variável não básica eficiente ser tornada variável básica. Se um ponto $\boldsymbol{\lambda} \in \Lambda$ pertence a várias regiões de indiferença, significa que estas correspondem a soluções eficientes localizadas na mesma face eficiente (esta face é apenas fracamente eficiente se esse ponto $\boldsymbol{\lambda}$ se situa na fronteira do diagrama paramétrico, ou seja $\boldsymbol{\lambda} \in \Lambda^0 \backslash \Lambda$).

A análise do diagrama paramétrico pode ser usada como um valioso utensílio na aprendizagem da "geometria" da região não dominada. As regiões de indiferença dependem das ordens de grandeza relativas dos valores das funções objectivo (ou seja, do comprimento relativo dos gradientes) e da geometria da região admissível. O tamanho da região de indiferença é de algum modo uma medida da robustez da solução face à variação dos pesos. Sempre que as diversas funções objectivo forem expressas em unidades com ordens de grandeza muito diferentes é aconselhável proceder à normalização das funções objectivo (o que altera a decomposição do diagrama paramétrico em termos dos espaços nele ocupados pelas regiões de indiferença).

Esta forma de escalarização é atractiva pela aparente facilidade de ser explicada aos decisores, captando a atitude de expressar o grau de importância que atribui a cada função objectivo. No entanto, esta informação, embora aparentemente simples, revela-se de grande dificuldade para o decisor e não há garantia de que as soluções encontradas resolvendo o problema (16.4) estejam de acordo com as preferências subjacentes à especificação dos pesos. Por exemplo, é possível obter a solução eficiente que optimiza $f_k(\mathbf{x})$ com $\lambda_k = 0$.

Através da resolução do problema (16.4) usando o método simplex só é possível obter vértices

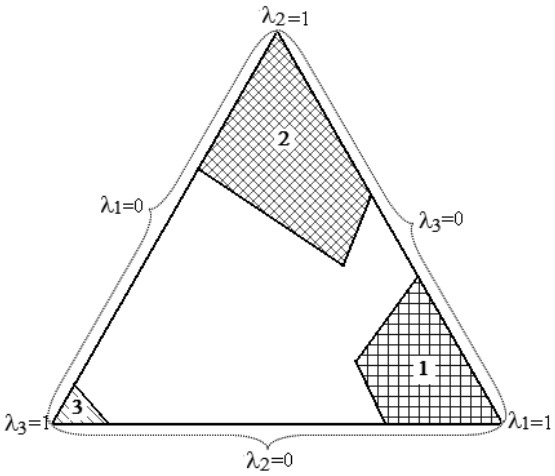


Figura 16.14: O diagrama paramétrico (para um problema com 3 funções objectivo) mostrando as regiões de indiferença correspondentes à solução que optimiza cada função objectivo

(soluções básicas) da fronteira não dominada.

A informação de preferências associada a este tipo de escalarização consiste em:

- Informação inter-critérios: escolhendo coeficientes de importância relativa (pesos) para as funções objectivo.

Este processo de cálculo de soluções eficientes pode também ser aplicado a problemas de programação inteira, inteira mista ou não linear multiobjectivo, mas não permite obter soluções eficientes não suportadas. A fig. 16.15(a) ilustra esta situação com um caso de programação inteira. Os pontos B e D são soluções óptimas alternativas da função objectivo ponderada cujo gradiente está representado na figura (z_λ). Um ligeiro aumento no peso atribuído a f_1 conduz apenas ao ponto D e um ligeiro aumento no peso de f_2 conduz apenas ao ponto B, não existindo nenhum vector de pesos que permita alcançar o ponto não dominado C (não suportado).

Num problema (não linear) em que o domínio das soluções admissíveis é convexo e as funções objectivo são côncavas é possível calcular-se todas as soluções eficientes próprias utilizando pesos estritamente positivos. Se o problema possuir soluções eficientes não próprias, estas podem também ser obtidas a partir da optimização de somas pesadas desde que se admita que os pesos podem ser nulos. A fig. 16.15(b) ilustra este caso, em que os pontos A e B só podem ser obtidos se se considerar um dos pesos igual a zero.

Minimização da distância a um ponto de referência

Uma solução de compromisso potencialmente interessante para o decisor é a que minimiza um dado tipo de distância à solução ideal, ou a outro ponto de referência do decisor que represente os níveis de cada função que gostaria de atingir (mesmo que não admissíveis). O que se pretende é então calcular uma solução eficiente admissível que esteja tão perto quanto possível (segundo uma dada métrica) das aspirações do decisor.

A fig. 16.16(a) mostra os contornos de isodistância em relação a $\mathbf{z}^*$ para as métricas L_1 (*city block*), L_2 (Euclideana) e L_∞ (de Chebyshev). Na fig. 16.16(b), os pontos $\mathbf{z}^1$, $\mathbf{z}^2$ e $\mathbf{z}^\infty$ minimizam as distâncias à solução ideal $\mathbf{z}^*$, usando as métricas L_1, L_2 e L_∞, respectivamente.

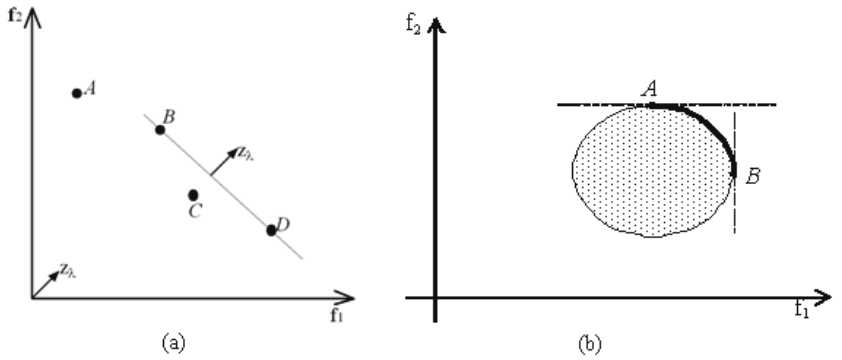


Figura 16.15: Optimização de somas ponderadas das funções objectivo (a) num problema de programação inteira e (b) num problema não linear convexo

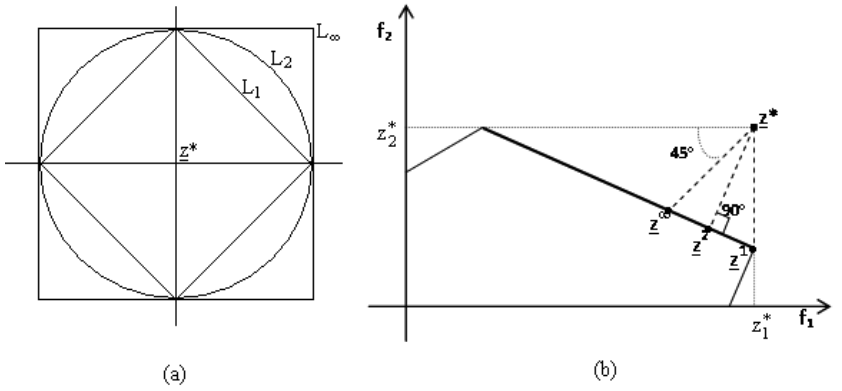


Figura 16.16: Métricas L_1, L_2 e L_∞: (a) contornos de isodistância, e (b) uso em funções distância.

Usando uma métrica L_β e tomando a solução ideal como ponto de referência o problema consiste em

$$\begin{aligned} &\min \|\mathbf{z}^* - \mathbf{f}(\mathbf{x})\|_\beta \\ &\text{s. a: } \mathbf{x} \in \mathcal{X} \end{aligned} \tag{16.5}$$

ou, considerando uma métrica L_β ponderada, com $\boldsymbol{\lambda} \geq \mathbf{0}$

$$\begin{aligned} &\min \boldsymbol{\lambda}\|\mathbf{z}^* - \mathbf{f}(\mathbf{x})\|_\beta \\ &\text{s. a: } \mathbf{x} \in \mathcal{X} \end{aligned} \tag{16.6}$$

Para $\beta = 1$ todos os desvios do ponto de referência são tomados em consideração na proporção directa da sua grandeza, enquanto para $2 < \beta < \infty$, maiores desvios vão tendo cada vez mais importância, até que para $\beta = \infty$ apenas o maior desvio é tido em conta.

Seja $\mathbf{x}^0$ solução do problema (16.5). Considerando o pior caso, isto é, a maior diferença entre as componentes dos vectores $\mathbf{z}^*$ e $\mathbf{f}(\mathbf{x}^0)$ através da métrica L_∞ (de Chebyshev) o problema

$$\min_{\mathbf{x}\in\mathcal{X}} \left\{ \max_{k=1,\ldots,p} (z_k^* - f_k(\mathbf{x})) \right\} \tag{16.7}$$

tem como solução $\mathbf{x}^0$ sse $\mathbf{x}^0$ e v^0 forem solução do problema (16.8)

$$\begin{aligned} &\min \quad v \\ &\text{s. a: } \begin{cases} v \geq z_k^* - f_k(\mathbf{x}) \;\; ; \;\; k = 1, \ldots, p \\ \mathbf{x} \in \mathcal{X} \\ v \geq 0 \end{cases} \end{aligned} \tag{16.8}$$

De forma mais genérica pode considerar-se uma métrica L_∞ ponderada (L_∞^λ):

$$\min_{\mathbf{x}\in\mathcal{X}} \left\{ \max_{k=1,\ldots,p} \lambda_k (z_k^* - f_k(\mathbf{x})) \right\} \tag{16.9}$$

O problema escalar (16.7) é um caso particular deste, em que todos os pesos λ_k são iguais a 1.

Note-se que, dado um problema de programação linear multiobjectivo, apenas resultam problemas escalares lineares usando as métricas L_1 ou L_∞.

Para garantir que as soluções obtidas por meio de (16.9) são estritamente eficientes (e não apenas fracamente eficientes) pode usar-se a métrica de Chebyshev (ponderada) aumentada ($L_\infty^{\lambda\epsilon}$):

$$\min_{\mathbf{x}\in\mathcal{X}} \left(\left\{ \max_{k=1,\ldots,p} \lambda_k (z_k^* - f_k(\mathbf{x})) \right\} + \rho \sum_{k=1}^{p} (z_k^* - f_k(\mathbf{x})) \right) \tag{16.10}$$

O segundo termo em (16.10) é uma perturbação acrescentada ao problema min-max normal, com ρ um escalar positivo suficientemente pequeno, que se destina a assegurar a eficiência estrita da solução.

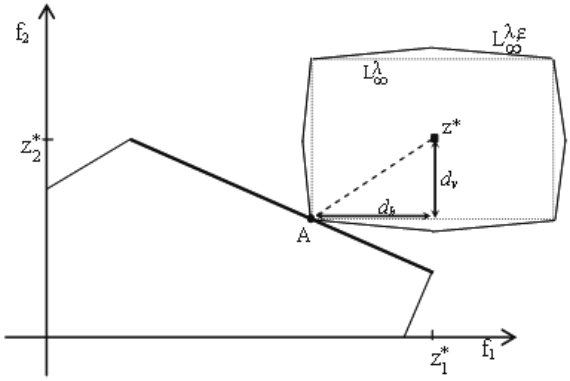


Figura 16.17: Minimização da distância de Chebyshev ponderada e aumentada à solução ideal

O problema escalar (16.10) pode escrever-se de forma equivalente

$$\begin{aligned} &\min \quad v + \rho \sum_{k=1}^{p} \left(z_k^* - f_k(\mathbf{x})\right) \\ &\text{s. a:} \begin{cases} v \geq \lambda_k \left(z_k^* - f_k(\mathbf{x})\right) \quad k = 1, \ldots, p \\ \mathbf{x} \in \mathcal{X} \\ v \geq 0 \end{cases} \end{aligned} \tag{16.11}$$

Note-se que a função objectivo do problema (16.11) pode ser substituída por $v - \rho \sum_{k=1}^{p} f_k(\mathbf{x})$ visto que o termo restante é constante.

A minimização da distância à solução ideal segundo a métrica de Chebyshev ponderada (L_∞^λ) ou ponderada e aumentada ($L_\infty^{\lambda\epsilon}$) é ilustrada na fig. 16.17. O ponto A é a solução que minimiza a distância a $\mathbf{z}^*$ segundo (L_∞^λ) ou ($L_\infty^{\lambda\epsilon}$) considerando um vector particular de pesos $\boldsymbol{\lambda}$ em que $\lambda_1 < \lambda_2$. O vector $\boldsymbol{\lambda}$ determina a direcção de projecção do ponto de referência $\mathbf{z}^*$ na região não dominada, tal que $d_v/d_h = \lambda_1/\lambda_2$.

Demonstra-se que se $\mathbf{x}' \in \mathcal{X}$ é uma solução para o problema de Chebyshev ponderado aumentado (16.11) para algum $\boldsymbol{\lambda} \geq \mathbf{0}$, então $\mathbf{x}'$ é uma solução eficiente para o problema multiobjectivo original. Habitualmente usam-se pesos normalizados, i.e. $\boldsymbol{\lambda} \in \Lambda^0 = \left\{\boldsymbol{\lambda} \in \mathbb{R}^p : \ \lambda_k \geq 0, \ k = 1, \ldots, p, \ \sum_{k=1}^{p} \lambda_k = 1\right\}$.

Considerando $\boldsymbol{\lambda} \in \Lambda^0$ e a solução ideal deslocada $\mathbf{z}^{**} = \mathbf{z}^* + \boldsymbol{\epsilon}$ como ponto de referência (em que os ϵ_k são constantes arbitrárias estritamente positivas), (Steuer, 1986) prova que existe sempre um ρ positivo que garante que qualquer solução eficiente de um problema com região admissível poliédrica ou um conjunto discreto finito possa ser obtida a partir do problema escalar (16.11). Assim, para ρ positivo suficientemente pequeno, além da optimização de (16.11) ser condição suficiente de eficiência – o que é válido para o caso geral – é também condição necessária para os casos referidos, que incluem a programação linear multiobjectivo e a programação inteira multiobjectivo. A métrica pesada e aumentada de Chebyshev não permite, contudo, caracterizar por completo o conjunto das soluções eficientes de problemas com soluções eficientes não próprias. Para estes casos, (Steuer, 1986) propõe uma abordagem lexicográfica, que também pode ser aplicada aos outros casos, mas que é mais difícil de implementar pois são necessárias duas fases de optimização. Na primeira fase considera-se a métrica L_∞ (ponderada) não aumentada, ou seja, apenas se minimiza a variável v. Na segunda

fase, minimiza-se $\sum_{k=1}^{p}(z_k^* - f_k(\mathbf{x}))$ no conjunto das soluções que minimizam v de modo a eliminar as soluções fracamente eficientes obtidas na primeira fase.

Nos problemas escalares anteriores é adoptada a solução ideal como ponto de referência, embora seja possível considerar outros pontos de referência (em geral, níveis de aspiração para cada função objectivo especificados pelo decisor). Se o ponto de referência usado não verificar $\mathbf{z}^r \geq \mathbf{f}(\mathbf{x})$ para todo o $\mathbf{x} \in \mathcal{X}$, então a variável v no problema escalar deve ser considerada sem restrição de sinal ($v \in \mathbb{R}$). Nestes casos poderá não se tratar da minimização de uma distância, já que $\mathbf{z}^r$ pode atravessar a fronteira não dominada e tornar-se atingível, deixando de se poder falar em métrica de Chebyshev. No entanto, garante-se igualmente que se obtém uma solução pelo menos fracamente eficiente no programa escalar não aumentado e estritamente eficiente no programa escalar aumentado. Estes programas minimizam funções que podem ser designadas genericamente por funções escalarizantes de *realização* (*achievement scalarizing functions* segundo (Lewandowski e Wierzbicki, 1988)).

A minimização da distância à solução ideal (ou a outro ponto de referência que represente níveis de aspiração para o decisor) pode ser interpretada como a minimização do desconforto de obter uma solução não dominada de compromisso $\mathbf{z}^0$ em vez do ponto ideal $\mathbf{z}^*$.

Através do problema (16.11) é possível obter todos os pontos da fronteira não dominada de um problema de programação linear multiobjectivo, quer sejam ou não vértices da região admissível.

A informação de preferências associada a este tipo de escalarização consiste em:

- Informação intra-critérios: estabelecimento de pontos de referência;
- Informação inter-critérios: coeficientes de ponderação da métrica L_∞.

4. Métodos multiobjectivo de apoio à decisão

Embora não seja fácil categorizar os diversos tipos de métodos de apoio à decisão multiobjectivo propostos na literatura, uma das classificações mais geralmente adoptadas baseia-se na fase do processo de decisão em que é requerida a intervenção do decisor. Assim, podem distinguir-se métodos em que:

(a) Não há articulação de preferências do decisor (métodos geradores). No caso de problemas de PLMO são geradas todas as soluções extremas não dominadas (e eventualmente todas as faces não dominadas de dimensão máxima) que são apresentadas ao decisor para escolha da solução preferida, face à informação gerada pelas sucessivas etapas de cálculo (que não requerem qualquer tipo de intervenção por parte do decisor)[1].

(b) A articulação de preferências do decisor é feita:

(b1) *a priori* (onde se destacam os métodos de função valor, ou utilidade) – inicialmente é obtida do decisor informação de preferências que serve para construir uma função valor que agrega numa única dimensão todos os critérios explicitamente considerados no modelo. A construção desta função valor (ou utilidade, quando há probabilidades associadas aos resultados de cada acção potencial, embora esta distinção seja muitas vezes ignorada na literatura) tem *tanto de arte como de ciência* (Keeney e Raiffa, 1976; Yu, 1985, cap. 6). Ou seja, apesar de existir modelação multicritério, o problema é depois reduzido ao caso monocritério e a optimização desta função valor conduz à solução de compromisso óptima.

(b2) progressivamente (métodos interactivos) – etapas de cálculo são alternadas com etapas de diálogo, em que o decisor é chamado a expressar as suas preferências perante soluções

[1] Neste caso, como a intervenção do decisor ocorre somente após a geração de todo o conjunto de soluções não-dominadas, também se diz que as preferências são informadas *a posteriori*.

que lhe são propostas, até se atingir uma condição de paragem (variável de método para método) que pode pressupor ou não a existência de uma função valor implícita do decisor. A assunção da existência implícita de uma função valor do decisor significa que este é incapaz de especificar a forma exacta da função, mas está em condições de fornecer informação local sobre as suas preferências, coerente com essa função, face a questões colocadas pelo método.

A utilização de métodos geradores em problemas complexos do mundo real revela-se proibitiva. Por um lado, o esforço computacional para calcular todos os pontos extremos não dominados é demasiado elevado mesmo para problemas lineares de dimensão média. Por outro, a quantidade de resultados gerada (sem qualquer intervenção do decisor no processo) é por vezes enorme, sem que isso se traduza num aumento da qualidade da informação colocada à disposição do decisor.

Os métodos que assumem a existência de uma função valor (a ser optimizada) reflectindo as preferências de um decisor racional têm sido sujeitos a muitas críticas, sobretudo com base em estudos empíricos que demonstraram que os decisores não são sempre coerentes com as funções valor destinadas a agregar os critérios e representar as suas preferências, agindo geralmente mais de acordo com a sua própria maneira de ver o problema do que procurando conciliar esta com os axiomas que são a justificação teórica da existência da função valor (Bell e Farquhar, 1986).

Os métodos interactivos, em que existe uma articulação progressiva das preferências do decisor ao longo do processo interactivo, são os mais adequados para constituírem o núcleo de sistemas de apoio à decisão para problemas multiobjectivo, pela capacidade em incorporar aprendizagem e adaptação na expressão de preferências ao longo do processo de decisão, onde o decisor é chamado a desempenhar um papel activo. Os métodos interactivos alternam etapas de cálculo de soluções não dominadas com etapas de diálogo, em que a intervenção do decisor é usada para guiar o processo interactivo de decisão, reduzindo o âmbito da pesquisa e o esforço computacional associado.

Podem distinguir-se duas implementações de metodologias interactivas resultantes de duas concepções básicas para o papel da interactividade:

(a) Orientada para a procura. O decisor age de acordo com uma estrutura de preferências pré-existente e estável (representada por uma função valor implícita), e permanece coerente com ela ao longo da utilização da metodologia, respondendo de forma determinística às questões colocadas. Ou seja, o objectivo da interacção é a procura de uma proposta óptima (que não depende da evolução do processo interactivo) face à estrutura de preferências, que é assim descoberta durante o processo. Nestas circunstâncias parece razoável impor a convergência matemática do método interactivo. A convergência é garantida e controlada pelo método.

(b) Orientada para a aprendizagem. É recusada a assunção de que existe uma estrutura de preferências pré-existente e estável com a qual o decisor é coerente. As preferências do decisor podem ser parcialmente instáveis e evolutivas. O objectivo da interacção é a aprendizagem das preferências, ou seja, a clarificação do que, na perspectiva do decisor, possa ser uma solução satisfatória. A convergência não é garantida e é controlada pelo decisor, fazendo então sentido falar em convergência psicológica (significando a identificação de uma solução eficiente como satisfatória graças à informação entretanto recolhida). O processo termina com uma solução de compromisso satisfatória de acordo com a informação disponível e a vontade do decisor quando não parecer possível a emergência de novas intuições sobre o problema.

Face aos métodos geradores, os métodos interactivos revelam vantagens inequívocas quer ao reduzir o esforço computacional, quer ao limitar o esforço cognitivo imposto ao decisor. Face aos métodos baseados em funções valor (ou utilidade) os métodos interactivos evitam a obtenção prévia de informação de preferências para formular a função valor, que se supõe ser a sua representação analítica,

permitindo a aprendizagem e a evolução de preferências à medida que vai sendo reunida informação ao longo do processo interactivo.

A interactividade significa oferecer ao decisor um ambiente operacional que suscite a exploração, a reflexão, a emergência de novas intuições, permitindo-lhe compreender com mais profundidade o problema de decisão em causa, contribuindo para moldar e fazer evoluir as suas preferências no sentido de guiar o processo de pesquisa, centrando-o nas regiões onde se localizam as soluções que julga mais interessantes, até encontrar uma solução de compromisso satisfatória ou chegar à conclusão que é necessário rever os dados iniciais e/ou o reformular o modelo matemático. A aprendizagem deve ser entendida não apenas no sentido do aumento do conhecimento disponível, mas sobretudo no do aperfeiçoamento das capacidades do decisor de modo a fazer um uso adequado desse conhecimento.

Os métodos interactivos tendem a ser vistos como a abordagem mais adequada para o estudo destes problemas: *...the future of multiple objective programming is in its interactive application* (Steuer, 1986, p. 386). Por outro lado, face à acessibilidade de elevadas capacidades de cálculo e potencialidades gráficas com custos cada vez mais acessíveis, encontra-se disponível a base tecnológica para a construção de sistemas de apoio à decisão flexíveis conjugando as características interactivas dos métodos multiobjectivo com adequados ambientes de interacção e meios gráficos de apresentação de informação.

Métodos interactivos em PLMO

Os métodos interactivos podem ser classificados de acordo com uma grande diversidade de parâmetros, não existindo uma categorização aceite de forma universal. Nesta secção são apresentadas algumas formas possíveis de categorização dos métodos interactivos de PLMO. A primeira baseia-se na estratégia de redução do âmbito da pesquisa (explícita ou implicitamente), segundo a qual se podem distinguir métodos de (Steuer, 1986; Clímaco et al., 2003):

- Redução progressiva da região admissível;
- Redução progressiva do diagrama paramétrico (espaço dos pesos);
- Contracção progressiva do cone dos gradientes das funções objectivo;
- Pesquisa direccional.

As três primeiras estratégias reduzem progressivamente a região eficiente que pode ser pesquisada em fases subsequentes.

Perante a diversidade de técnicas de cálculo de soluções eficientes, podem classificar-se os métodos de acordo com o tipo de função escalar substituta (função escalarizante) usada para agregar temporariamente os objectivos:

- Optimização de uma função objectivo restringindo as outras;
- Soma ponderada das funções objectivo;
- Minimização de uma distância a um ponto de referência.

A possibilidade de o decisor intervir para solicitar a realização de uma dada operação em qualquer momento do processo de decisão interactivo ou ser obrigado a uma sequência pré-determinada de fases de cálculo e de diálogo, permite classificar os métodos em:

- Não estruturados;
- Estruturados.

Estas classificações não são mutuamente exclusivas, dado que mesmo dentro de cada parâmetro de categorização há métodos que combinam diferentes estratégias de redução do âmbito da pesquisa e técnicas de cálculo de soluções eficientes. Por outro lado, estas classificações não são exaustivas, pois existem métodos que é difícil englobar em qualquer delas.

Seguidamente são brevemente descritos os métodos STEM, de Zionts e Wallenius, *Interval Criterion Weights*, *Pareto Race* e TRIMAP, considerados representativos das várias categorias anteriormente apresentadas. Estes métodos são detalhadamente descritos, com exemplos ilustrativos do respectivo funcionamento, em (Clímaco et al., 2003).

Método STEM

O *Step Method* (STEM), desenvolvido por Benayoun et al. (1971) é um método interactivo de redução da região admissível. Em cada fase de diálogo, após uma fase de cálculo, o decisor é chamado a especificar as quantidades que está disposto a sacrificar nas funções objectivo cujos valores considera satisfatórios de modo a melhorar as restantes. Em cada fase de cálculo é minimizada uma distância ponderada de Chebyshev à solução ideal.

Em cada iteração, o problema a optimizar reflecte as escolhas do decisor feitas em iterações precedentes, através da redução da região admissível. É apresentada ao decisor a solução de compromisso calculada em cada iteração minimizando uma distância ponderada de Chebyshev à solução ideal. Se os valores das funções objectivo são considerados satisfatórios, o processo termina; caso contrário, o decisor deve especificar quais pretende relaxar, e de que quantidade, por forma a melhorar os outros objectivos. A região admissível é então reduzida através das limitações nos valores das funções objectivo (calculadas com base nas quantidades de relaxamento introduzidas pelo decisor).

O método de Zionts e Wallenius

O método de Zionts e Wallenius (1976; 1983) reduz progressivamente o diagrama paramétrico (espaço dos pesos), de acordo com as preferências do decisor expressas através das respostas em cada interacção face a comparações entre pares de soluções e julgamentos sobre as tendências de variação unitária ao longo de arestas que têm origem na solução corrente e conduzem a outras soluções não dominadas. Em cada fase de cálculo é optimizada uma soma ponderada das funções objectivo.

Partindo das respostas dadas pelo decisor, o método introduz então restrições no diagrama paramétrico, reduzindo progressivamente o domínio admissível para a selecção de um novo vector de pesos. O processo termina quando o diagrama paramétrico for reduzido a uma região suficientemente pequena de forma a que seja identificada uma solução final, convergindo para o óptimo de uma função valor implícita não necessariamente linear do decisor (na realidade, para o vértice eficiente que garante o maior valor para a função), ou a informação de preferências expressa pelo decisor torne a solução actual mais interessante do que as que lhe são propostas para comparação. Parte-se do princípio que as respostas do decisor nas fases de diálogo são coerentes com essa função valor implícita, embora no caso de serem detectadas incoerências seja dada a possibilidade de eliminar as restrições mais antigas no diagrama paramétrico.

O método *Interval Criterion Weights*

O método *Interval Criterion Weights* (ICW), desenvolvido por Steuer (1977; 1986), é um procedimento interactivo que reduz progressivamente o cone dos critérios (cone convexo gerado pelos gradientes das funções objectivo). Esta redução é feita de acordo com as preferências manifestadas pelo decisor ao escolher a solução preferida numa amostra de soluções não dominadas que lhe é apresentada em cada fase de diálogo. Esta amostra é obtida por técnicas de filtragem a partir de um conjunto de soluções

não dominadas, de modo a colocar à apreciação do decisor um número não muito elevado de soluções. Em cada fase de cálculo são optimizadas várias somas ponderadas das funções objectivo.

Por um lado, o método ICW usa várias combinações regularmente dispersas no diagrama paramétrico (espaço dos pesos) para definir um conjunto de somas ponderadas, evitando assim obter informação sobre as preferências do decisor através da indicação explícita de pesos. Por outro lado, este método também não faz apelo à especificação de valores satisfatórios para as funções objectivo usados para reduzir a região admissível (como, por exemplo, no STEM). A informação dada pelo decisor ao escolher a solução da amostra que corresponde mais de perto às suas preferências é usada para contrair o cone dos critérios em torno do gradiente da função objectivo soma ponderada que deu origem a essa solução, resultando o cone de critérios reduzido da iteração seguinte. O cone dos critérios é assim gradualmente contraído e deslocado até se focar numa pequena porção da superfície da região admissível, contendo o vértice eficiente com maior valor para a função valor implícita do decisor.

O método *Pareto Race*

O método *Pareto Race*, proposto por Korhonen e Wallenius (1988) baseado em trabalhos de Korhonen (1987) e Korhonen e Laakso (1986b; 1986a), é um método de pesquisa direccional livre, que permite ao decisor mover-se por qualquer ponto sobre a região eficiente. A informação requerida ao decisor consiste fundamentalmente na especificação das funções objectivo a melhorar alterando a direcção do movimento. As soluções eficientes são obtidas por meio da optimização de uma função escalarizante baseada em ponto de referência e programação paramétrica em relação aos termos independentes das restrições (metas não flexíveis).

A partir de níveis de aspiração para os valores das funções objectivo, especificados inicialmente pelo decisor, é construída uma direcção de referência (direcção que, partindo de um ponto admissível no espaço dos objectivos, oferece uma variação nos valores das funções objectivo que está de acordo com as preferências do decisor). Esta direcção de referência é então projectada sobre o conjunto das soluções eficientes, gerando uma trajectória (subconjunto das soluções eficientes) que é apresentada ao decisor. O decisor pode assim percorrer a fronteira eficiente, controlando a direcção do movimento (privilegiando diferentes funções objectivo) e a velocidade (obtendo soluções mais ou menos próximas umas das outras) como se estivesse a conduzir um automóvel (daí a denominação *Pareto Race*) sobre essa superfície.

O método TRIMAP

O método TRIMAP, desenvolvido por Clímaco e Antunes (1987; 1989), é um método de pesquisa livre baseado na aprendizagem progressiva e selectiva do conjunto das soluções não dominadas, que combina a redução da região admissível com a redução do diagrama paramétrico (espaço dos pesos). O decisor pode especificar limitações inferiores para os valores da funções objectivo e impor restrições nos pesos. Em cada fase de cálculo é optimizada uma soma ponderada das funções objectivo.

O TRIMAP combina três procedimentos fundamentais: decomposição do diagrama paramétrico, introdução de restrições no espaço dos objectivos e introdução de restrições nos pesos. As limitações introduzidas nos valores das funções objectivo podem ser traduzidas automaticamente para o diagrama paramétrico, que é usado como um valioso meio para recolher e apresentar a informação ao decisor. Esta possibilidade é muito útil dado que os decisores estão geralmente mais à vontade quando o diálogo é conduzido em termos dos valores das funções objectivo, que constitui o espaço que lhes é mais familiar. Inicialmente são calculadas as soluções eficientes que optimizam cada uma das funções objectivo, fornecendo ao decisor uma primeira informação sobre a gama de valores de cada objectivo, e a solução eficiente que minimiza uma distância ponderada de Chebyshev à solução ideal. Através da análise comparativa dos gráficos do diagrama paramétrico e do espaço dos objectivos, o decisor pode

fazer uma cobertura progressiva e selectiva do diagrama paramétrico, decidindo em cada interacção o interesse de pesquisar soluções em regiões deste espaço ainda não exploradas. A redução do âmbito da pesquisa é feita principalmente impondo limitações nos valores das funções objectivo, que são traduzidas para o diagrama paramétrico.

O método TRIMAP será analisado mais detalhadamente na secção 6, descrevendo-se através de um exemplo ilustrativo como o método pode constituir um utensílio para apoio à decisão em problemas de programação linear com três funções objectivo.

5. O Método TRIMAP (programação linear TRIobjectivo - Método de Aprendizagem Progressiva das soluções não dominadas)

O método TRIMAP, desenvolvido por Clímaco e Antunes (1987; 1989), é um método de pesquisa livre onde se pretende proporcionar ao decisor uma aprendizagem progressiva e selectiva do conjunto das soluções não dominadas. O TRIMAP combina a redução da região admissível com a redução do diagrama paramétrico. O decisor pode especificar as suas preferências por meio do estabelecimento de limitações inferiores para os valores da funções objectivo e da imposição de restrições no diagrama paramétrico.

A finalidade essencial do método é a de ajudar o decisor a eliminar os subconjuntos de soluções não dominadas que não lhe interessam, e não a de assegurar a convergência para a solução de compromisso óptima de uma função utilidade implícita, que se supõe pré-existente ao processo de decisão. O processo interactivo termina quando o decisor considera conhecer o bastante sobre o conjunto das soluções não dominadas, sendo sempre possível rever decisões e julgamentos anteriores à medida que vai sendo reunido maior conhecimento sobre o problema. Ou seja, a informação de preferências do decisor não necessita ser coerente em sucessivas interacções. Usando a terminologia de (Roy, 1987), no TRIMAP a convergência (de uma função utilidade) deve dar lugar à criação, de modo que o processo interactivo seja um processo construtivo e não a determinação de algo pré-existente.

O TRIMAP combina três procedimentos fundamentais: decomposição do diagrama paramétrico, introdução de restrições no espaço dos objectivos e introdução de restrições no diagrama paramétrico. A característica mais inovadora do método é a possibilidade de traduzir as limitações introduzidas nos valores das funções objectivo para o diagrama paramétrico, que é usado como um meio coerente para recolher e apresentar a informação ao decisor. O TRIMAP é vocacionado para problemas com três funções objectivo, o que, embora constituindo uma limitação, permite o uso de meios gráficos adequados ao diálogo com o decisor. O principal objectivo é possibilitar ao decisor um preenchimento progressivo e selectivo do diagrama paramétrico, que lhe dê informação sobre a forma da superfície não dominada, evitando deste modo um estudo exaustivo de regiões onde os valores das funções objectivo sejam muito semelhantes (o que frequentemente acontece em casos reais).

A redução do âmbito da pesquisa é feita principalmente impondo limitações nos valores das funções objectivo (que é o tipo de informação que menos esforço exige da parte do decisor), que são traduzidas para o diagrama paramétrico. A introdução destas limitações adicionais pode ainda ser usada para obter soluções não dominadas que não sejam pontos extremos do poliedro admissível. É também possível impor restrições directamente no diagrama paramétrico. Através da análise comparativa do diagrama paramétrico e do espaço dos objectivos, o decisor pode fazer uma cobertura progressiva e selectiva do diagrama paramétrico, decidindo em cada interacção o interesse de pesquisar soluções em áreas do triângulo ainda não exploradas.

Inicialmente são calculadas as soluções eficientes que optimizam cada uma das funções objectivo, fornecendo ao decisor uma primeira informação sobre a gama de valores de cada objectivo. Como informação auxiliar é também calculada a solução eficiente (que não é, em geral, um vértice) que minimiza uma distância ponderada de Chebyshev à solução ideal, como no método STEM.

A selecção dos pesos para o cálculo de soluções eficientes, pode ser feita de duas formas:

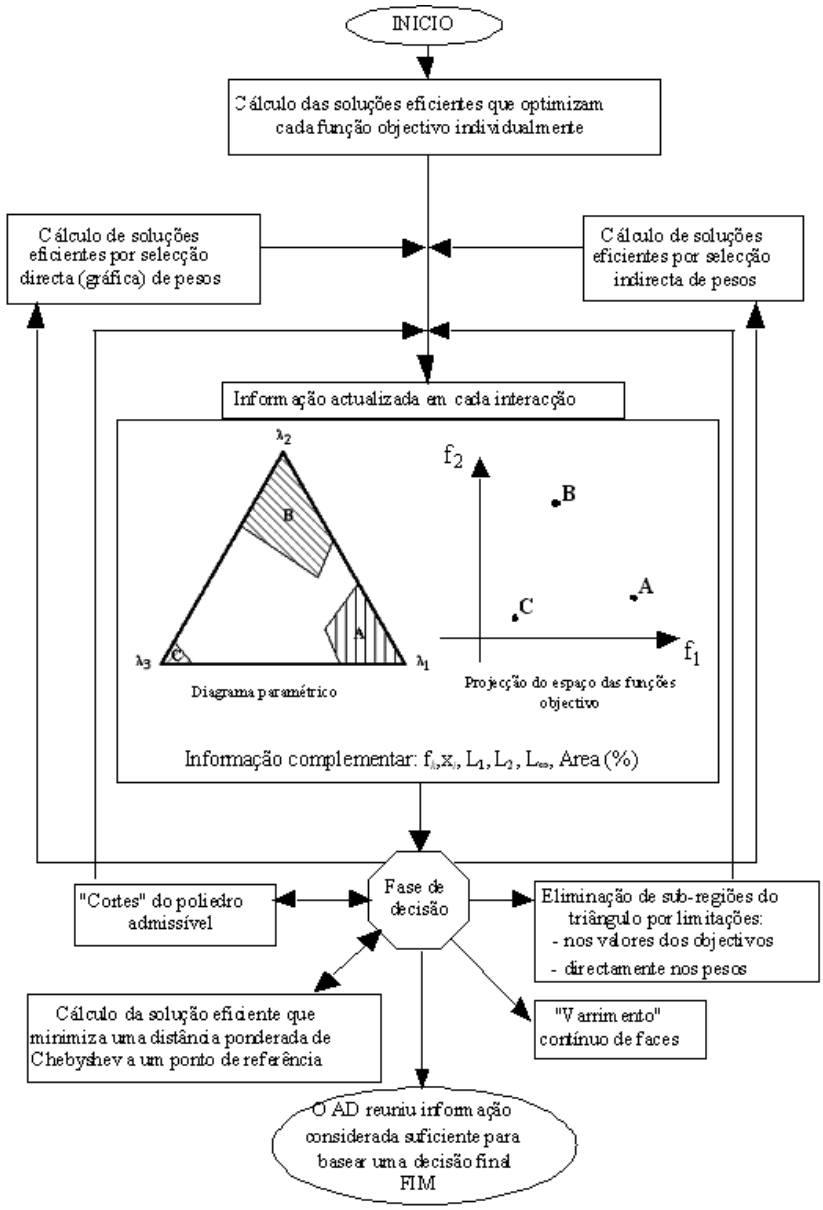


Figura 16.18: Fluxograma do método TRIMAP

- Directamente, em que o decisor escolhe um vector de pesos de uma zona do triângulo não preenchida, que lhe parece importante para continuar a pesquisa;
- Indirectamente, construindo uma função ponderada cujo gradiente é normal ao plano que passa por três soluções não dominadas já calculadas escolhidas pelo decisor. Se os pesos assim obtidos não forem todos não negativos, é feita uma pequena perturbação no gradiente da função objectivo ponderada por forma a assegurar essa condição.

A introdução de limitações adicionais nos valores das funções objectivo, e a respectiva tradução para o diagrama paramétrico, permite que o diálogo com o decisor seja feito em termos dos valores das funções objectivo acumulando a informação resultante no diagrama paramétrico. O estabelecimento da limitação adicional $f_k(\mathbf{x}) \geq e_k$, $(e_k \in \mathbb{R},\ k \in \{1,2,3\})$ conduz à construção do problema auxiliar

$$\begin{aligned} &\max f_k(\mathbf{x}) \\ &\text{s. a: } \mathbf{x} \in \mathcal{X}_a \equiv \{\mathbf{x} \in \mathcal{X} : f_k(\mathbf{x}) \leq e_k\} \end{aligned} \tag{16.12}$$

Maximizando $f_k(\mathbf{x})$ em $\mathcal{X}_a$ são obtidas soluções (básicas) óptimas alternativas (note-se que o gradiente da função a maximizar tem a mesma direcção do gradiente da restrição adicional). Os vértices eficientes do poliedro admissível $\mathcal{X}_a$ que optimizam (16.12) são seleccionados e as sub-regiões do diagrama paramétrico que correspondem a cada um destes pontos são calculadas e representadas graficamente. Estas sub-regiões do diagrama paramétrico são as regiões de indiferença definidas por $\left\{\boldsymbol{\lambda}^T W \geq 0,\ \boldsymbol{\lambda} \in \Lambda\right\}$, relativas a cada base eficiente alternativa. A união de todas estas regiões de indiferença determina a sub-região do diagrama paramétrico onde a limitação adicional no valor da função objectivo é satisfeita. Se o decisor estiver apenas interessado em soluções não dominadas que satisfaçam $f_k(\mathbf{x}) \geq e_k$, então é suficiente restringir a pesquisa a esta sub-região. Se o decisor pretender impor mais do que uma limitação então (16.12) é resolvido para cada uma delas e as sub-regiões correspondentes no diagrama paramétrico são preenchidas com diferentes padrões (ou cores) permitindo assim visualizar claramente as zonas onde existem intersecções. A introdução destas limitações adicionais pode também ser usada para obter soluções não dominadas que, em geral, não são vértices do poliedro admissível original.

É também possível eliminar regiões do diagrama paramétrico impondo limitações directamente na variação dos pesos (dos tipos $\lambda_i/\lambda_j \geq u_{ij},\ i,j \in \{1,2,3\},\ i \neq j,\ u_{ij} \in \mathbb{R}^+$, ou $0 < u_L \leq \lambda_k \leq u_H < 1$, com $k \in \{1,2,3\}$).

O TRIMAP permite ainda percorrer faces não dominadas, com uma interface semelhante ao do método *Pareto Race*, entre dois dos seus vértices previamente calculados. É mostrada uma linha que avança ou recua sobre a projecção da face a uma velocidade controlada pelo utilizador. Os valores das funções objectivo correspondentes aos pontos percorridos são dinamicamente apresentados num gráfico de barras.

O decisor pode identificar um ponto de referência no espaço dos objectivos e o programa calcula a solução não dominada que minimiza uma distância ponderada de Chebyshev a esse ponto.

Em cada interacção de TRIMAP são apresentados ao decisor dois gráficos principais. O primeiro é o diagrama paramétrico mostrando as regiões de indiferença correspondentes às soluções extremas não dominadas já conhecidas. O segundo é uma projecção do espaço dos objectivos mostrando as soluções não dominadas já calculadas. Estão também disponíveis indicadores complementares sobre cada solução: distâncias L_1, L_2 e L_∞ à solução ideal e a área da região de indiferença (percentagem ocupada da área total do triângulo). Outro tipo de informação gráfica disponível sobre cada solução é o gráfico em teia de aranha (*spider web*), também designado por mapas de radar (*radar charts*) ou estrutura em estrela (*star structures*), que constitui uma forma útil de condensar as várias dimensões de avaliação num único gráfico (Kasanen et al., 1991). O gráfico em teia de aranha usado em TRIMAP considera o ponto ideal (ou outro ponto de referência) no centro do gráfico, fornecendo uma impressão visual da diminuição de distâncias necessária para atingir esse ponto.

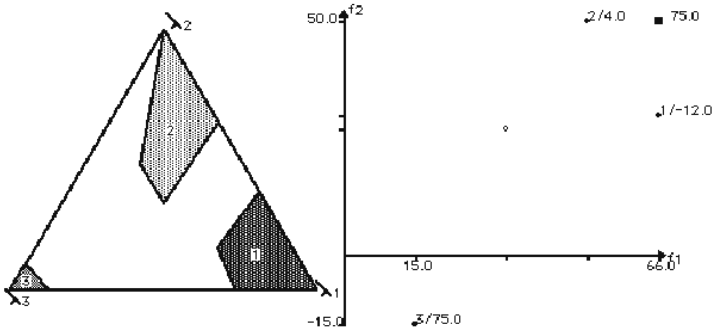


Figura 16.19: O diagrama paramétrico e uma projecção do espaço dos objectivos após a optimização individual de f_1, f_2 e f_3

Exemplo ilustrativo da aplicação do TRIMAP

Para ilustrar a aplicação de TRIMAP a problemas lineares multiobjectivo, consideremos o seguinte problema, com quatro variáveis de decisão, três funções objectivo a maximizar e três restrições:

$$\text{Max} \begin{bmatrix} 3 & 1 & 2 & 1 \\ 1 & -1 & 2 & 4 \\ -1 & 5 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$\text{s. a:} \begin{bmatrix} 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \leq \begin{bmatrix} 60 \\ 60 \\ 50 \end{bmatrix} \qquad \mathbf{x} \geq \mathbf{0}\ , \quad \mathbf{x} \in \mathbb{R}^4$$

x_5, x_6, x_7 são variáveis folga (*slacks*) correspondentes às restrições 1, 2 e 3, respectivamente.

O TRIMAP começa por calcular as soluções não dominadas que optimizam separadamente cada função objectivo. A fig. 16.19 mostra o diagrama paramétrico com as regiões de indiferença correspondentes às soluções que optimizam f_1, f_2 e f_3 (soluções 1, 2 e 3, respectivamente), e a projecção ortogonal em $f_1 \times f_2$ do espaço dos objectivos. O valor a seguir à identificação da solução representa a função que é projectada (f_3 neste caso). O quadrado preto representa a solução ideal. O círculo representa a solução que minimiza uma distância ponderada de Chebyshev a um ponto de referência (inicialmente a solução ideal).

A análise do diagrama paramétrico revela-nos que existe pelo menos outra solução alternativa à solução 2 que também optimiza f_2. Colocando o cursor do rato na zona do diagrama paramétrico à esquerda da região de indiferença da solução 2, e escolhendo um vector de pesos para construir uma soma ponderada das funções objectivo encontramos a solução 4 (fig. 16.20) que é um óptimo alternativo da solução 2 em relação a f_2. A aresta não dominada que liga estas duas soluções extremas não dominadas é também já conhecida.

A tabela seguinte mostra a informação disponível sobre as soluções não dominadas já conhecidas:

Solução	f_1	f_2	f_3	Área (%)	$\mathbf{x}_B$	L_∞
1	66.0	30.0	-12.0	13.43	$x_1 = 18.0$; $x_3 = 6.0$; $x_7 = 14.0$	87.0
2	51.0	50.0	4.0	17.01	$x_1 = 14.0$; $x_4 = 9.0$; $x_5 = 5.0$	71.0
3	15.0	-15.0	75.0	1.25	$x_2 = 15.0$; $x_5 = 45.0$; $x_7 = 20.0$	65.0
4	12.5	50.0	25.0	7.87	$x_4 = 12.5$; $x_5 = 22.5$; $x_6 = 35.0$	53.5

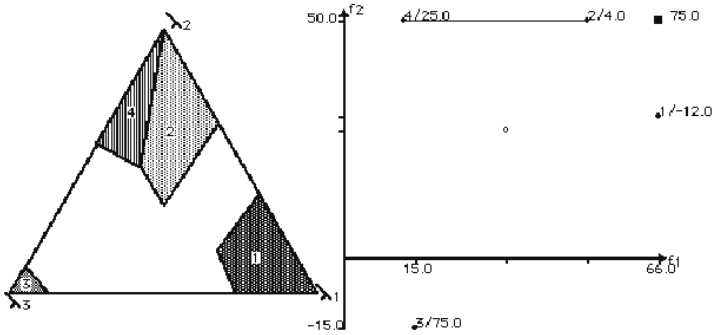


Figura 16.20: Cálculo da solução 4 por posicionamento do rato sobre uma zona não coberta do diagrama paramétrico

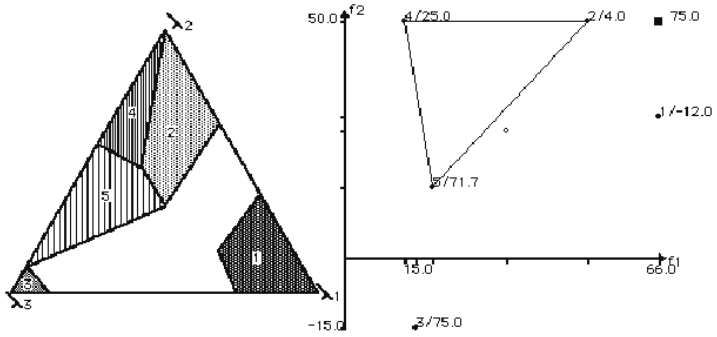


Figura 16.21: Cálculo da solução 5 optimizando uma função ponderada cujo gradiente é normal ao plano que passa pelas soluções 2, 3 e 4 no espaço dos objectivos.

Sobre a solução que minimiza uma distância ponderada (com pesos que se destinam a ter em conta as ordens de grandeza em que são expressas as funções objectivo) de Chebyshev à solução ideal está disponível a informação: $f_1 = 33.9$; $f_2 = 26.9$; $f_3 = 44.2$; $x_1 = 5.7$, $x_2 = 7.2$, $x_3 = 1.6$, $x_4 = 6.3$; $L_\infty = 32.1$.

Nesta fase do processo de decisão interactivo o método TRIMAP é útil ao possibilitar quer explorar regiões não cobertas no diagrama paramétrico, quer analisar mais detalhadamente a região eventualmente já identificada como interessante pelo decisor.

A solução 5 (fig. 16.21) foi calculada optimizando uma função ponderada cujo gradiente é normal ao plano que passa pelas soluções 2, 3 e 4 seleccionadas pelo decisor. A análise do diagrama paramétrico na fig. 16.21 revela-nos que é já conhecida uma face não dominada definida pelas soluções extremas 2, 4 e 5.

Solução	f_1	f_2	f_3	Área (%)	$\mathbf{x}_B$	L_∞
5	18.3	15.0	71.7	17.0	$x_2 = 11.7$; $x_4 = 6.7$; $x_5 = 28.3$	47.7

Para além de possibilitar uma pesquisa livre, TRIMAP é especialmente útil para o estabelecimento de limitações nos valores das funções objectivo. Esta informação é traduzida para o diagrama paramétrico, sendo apresentadas as regiões correspondentes às soluções não dominadas que satisfazem as restrições impostas pelo decisor. Suponhamos que em relação a f_1 o decisor não está interessado em soluções distantes da solução ideal mais do que 40. Esta informação de preferências corresponde

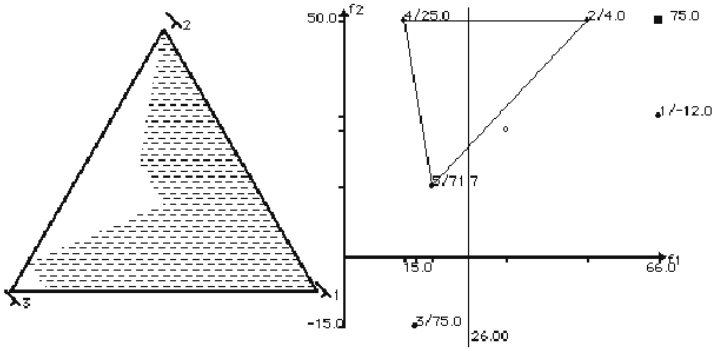


Figura 16.22: Introdução da restrição adicional f1(x) ? 26

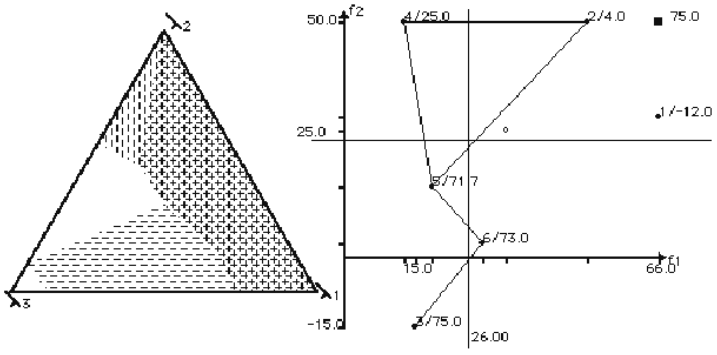


Figura 16.23: Introdução das restrições adicionais $f_1(\mathbf{x}) \geq 26$ e $f_2(\mathbf{x}) \geq 25$.

a introduzir a restrição adicional $f_1(\mathbf{x}) \geq 26$, que é satisfeita na região do diagrama paramétrico assinalada na fig. 16.22. As soluções extremas 1 e 2 (bem como a solução que minimiza a distância de Chebyshev à solução ideal) já conhecidas satisfazem esta restrição adicional imposta pelo decisor.

Vamos supor que o decisor escolhe, por selecção directa (utilizando o posicionamento do rato), um conjunto de pesos de uma zona do triângulo não preenchida satisfazendo esta limitação. Deste modo é obtida a solução 6 (fig. 16.24).

À medida que vai reunindo informação o decisor pode ir diminuindo o âmbito da pesquisa, sobretudo à custa da imposição de limitações adicionais nas funções objectivo, que são traduzidas para o diagrama paramétrico. Suponhamos que para além de manter a informação de preferências correspondente a $f_1(\mathbf{x}) \geq 26$, o decisor está agora também em condições de expressar uma limitação inferior em f_2, por exemplo $f_2(\mathbf{x}) \geq 25$. Na fig. 16.23 as regiões do diagrama paramétrico correspondentes a cada limitação são preenchidas com tramas diferentes colocando em evidência a respectiva intersecção, onde o decisor deve prosseguir a pesquisa se pretender manter-se coerente com estas expressões das suas preferências actuais.

Pesquisando novas soluções na região onde estas duas limitações nos valores das funções objectivo são simultaneamente satisfeitas, encontramos a solução 7 (fig. 16.24). Nesta situação conhecemos a face não dominada definida pelas soluções extremas 2-4-5; existe uma outra face definida pelas soluções extremas 3-5-6 que é dominada pelos pontos situados sobre as arestas 3-6 e 6-5 (comparar com a projecção do espaço dos objectivos), e existem pelo menos duas outras faces não dominadas ainda não completamente conhecidas, uma que inclui as soluções 2-5-6-7 e outra que inclui as soluções

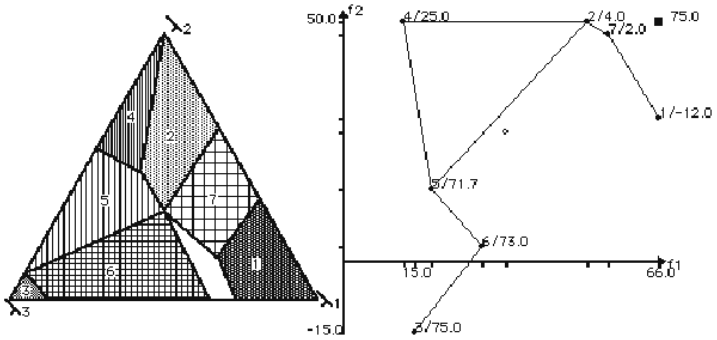


Figura 16.24: Cálculo da solução 7 na região onde as limitações $f_1(\mathbf{x}) \geq 26$ e $f_2(\mathbf{x}) \geq 25$ são simultaneamente satisfeitas

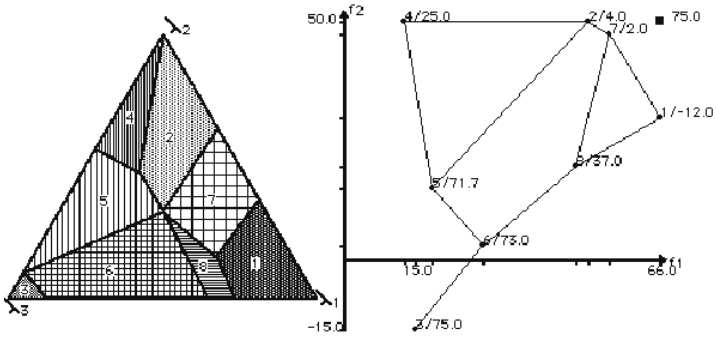


Figura 16.25: Nesta situação são conhecidas todas as soluções (vértices) não dominadas

1-7.

De modo a tentar obter mais informação sobre estas faces não dominadas vamos supor que o decisor, por indicação de um ponto – conjunto de pesos – nesta zona não preenchida do triângulo, calculou a solução 8 (fig. 16.25). Na fig. 16.25 são conhecidas todas as soluções extremas não dominadas, dado que o diagrama paramétrico está totalmente preenchido. Note-se que não é este, em geral, o objectivo da aplicação do TRIMAP.

O problema tem 8 soluções extremas não dominadas e existem 3 faces não dominadas (uma definida pelas soluções 2-4-5, outra por 2-5-6-8-7 e outra pelas soluções 1-7-8).

Solução	f_1	f_2	f_3	Área (%)	$\mathbf{x}_B$	L_∞
6	29.0	3.0	73.0	23.71	$x_2 = 13.0;\ x_3 = 8.0;\ x_5 = 15.0$	47.0
7	55.5	47.5	2.0	15.09	$x_1 = 14.5;\ x_3 = 2.5;\ x_4 = 7.0$	73.0
8	48.5	19.5	37.0	4.62	$x_1 = 7.5;\ x_2 = 7.0;\ x_3 = 9.5$	38.0

6. Integração entre Métodos em Sistemas de Apoio à Decisão

A experiência adquirida com o desenvolvimento das implementações computacionais e com as aplicações de métodos interactivos de programação linear multiobjectivo, bem como a análise comparativa das suas características conceptuais, fizeram surgir a necessidade de dispor de ferramentas

computacionais mais flexíveis, capazes de constituir uma base de experimentação quer da adequação dos métodos a tipos de problemas e a fases do processo interactivo de decisão, quer das possibilidades de comutação entre métodos assegurando a máxima transferência de informação, quer mesmo da desagregação dos métodos em procedimentos elementares. Esta linha de evolução teve continuidade com os ambientes interactivos TOMMIX e SOMMIX.

O TOMMIX é um ambiente interactivo vocacionado para problemas com três funções objectivo, incluindo os métodos STEM, Zionts-Wallenius, TRIMAP, Pareto Race e Interval Criterion Weights (representativos de diferentes estratégias de redução do âmbito da pesquisa, técnicas de cálculo de soluções eficientes e formas de informação de preferências), onde a estrutura interna dos métodos é respeitada (Antunes et al., 1992). A partir da avaliação das características conceptuais de cada método e das suas implementações isoladas foi desenvolvida a base integrada de métodos TOMMIX que permite tirar partido da combinação de diferentes tipos de métodos interactivos de programação linear multiobjectivo, tornando possível a transferência de informação entre cada um deles utilizável nas fases subsequentes do processo de decisão, e usando meios de interacção com o utilizador e formas de apresentação de resultados comuns. A base de métodos está especialmente adaptada (embora não limitada) a problemas de programação linear com três funções objectivo, o que permite o uso de meios gráficos particularmente úteis no diálogo com o decisor. Por esta razão foi-lhe dado o nome de TOMMIX (*three-objective methods mixed*).

A base de métodos inclui cinco métodos interactivos de programação linear multiobjectivo: STEM, Zionts-Wallenius (ZW), *Interval Criterion Weights* (ICW), *Pareto Race* (PR) e TRIMAP. Estes métodos:

- São representativos de diferentes estratégias para reduzir o âmbito da pesquisa:
  - redução da região admissível (STEM, TRIMAP),
  - redução do espaço dos pesos (ZW, TRIMAP),
  - contracção do cone dos critérios (ICW),
  - pesquisa direccional (PR);
- Admitem ou não a intervenção do utilizador para solicitar a realização de uma dada função em qualquer momento do processo interactivo:
  - estruturado (STEM, ZW, ICW),
  - não estruturado (PR, TRIMAP);
- Usam diferentes técnicas de cálculo de soluções não dominadas:
  - soma ponderada das funções objectivo (ZW, ICW, TRIMAP),
  - minimização de uma distância a um ponto de referência (STEM, PR, TRIMAP),
  - programação paramétrica em relação aos termos independentes das metas flexíveis (PR);
- Requerem (directa ou indirectamente) distinta informação de preferências do decisor:
  - níveis de aspiração para funções objectivo (PR, TRIMAP),
  - limitações inferiores para funções objectivo (STEM, PR, TRIMAP),
  - limitações nos pesos (TRIMAP),
  - comparações par a par entre soluções (ZW),
  - escolha de uma solução preferida de entre uma amostra de soluções não dominadas (ICW),
  - avaliação de tendências de variação unitária ao longo de arestas eficientes (ZW);

- Apresentam diferentes formas de comunicação de informação com utilizador.

O TOMMIX permite o uso isolado de cada método oferecendo também a possibilidade de trocar de método em qualquer interacção com o decisor. É, assim, possível tirar partido das potencialidades de cada método em diferentes fases do processo de decisão, captar distintas atitudes do decisor, bem como avaliar a qualidade da aplicação de cada um dos métodos em problemas de decisão particulares. Esta base de métodos de programação linear multiobjectivo permite total flexibilidade nas transições entre métodos, dado que as limitações adicionais (que reduzem o âmbito da pesquisa) introduzidas usando um dado método não interferem com as características fundamentais dos outros métodos.

A investigação tendente a estender de forma coerente este ambiente para problemas com mais de três funções objectivo deu origem ao ambiente interactivo SOMMIX (Clímaco et al., 1997), que é baseado num painel de controlo que oferece ao utilizador uma gama de comandos, constituindo o resultado da extensão de TOMMIX ao caso genérico de programação linear multiobjectivo. As ferramentas para calcular novas soluções eficientes, bem como os processos de diálogo com o utilizador são do mesmo tipo dos usados no TOMMIX, onde a estrutura dos métodos era preservada. Os comandos correspondem, no essencial, a todos os procedimentos elementares que é possível identificar nos métodos integrados no TOMMIX, e que podem ser usados de forma independente dos próprios métodos. A informação gráfica sobre as soluções eficientes já calculadas é fornecida sobretudo através de gráficos de barras, *spider web* e projecções do espaço dos pesos. Detalhes sobre o funcionamento e potencialidades de utilização do SOMMIX podem ser consultados em (Clímaco et al., 1997).

7. Breve referência a estudos de aplicação

Estas ferramentas de apoio à decisão em problemas de programação linear multiobjectivo têm sido usadas na realização de vários estudos nas áreas dos planeamento da expansão de sistemas produtores de energia eléctrica, planeamento da modernização de redes de telecomunicações face à introdução de novas tecnologias e novos serviços, e planeamento regional baseado em análise input-output e estudo das interacções energia-economia-ambiente.

Um modelo de planeamento energético considera três funções objectivo quantificando o custo total dos planos de expansão (a minimizar), a fiabilidade do sistema produtor (a maximizar) e o impacte ambiental (a minimizar). As variáveis de decisão referem-se ao tipo de tecnologias de geração (fuel, nuclear e carvão) consideradas para a expansão do sistema produtor. As restrições podem ser classificadas em três categorias: de carga, operacionais das unidades geradoras e orçamentais. Detalhes sobre a modelação matemática deste problema bem como o respectivo estudo usando o método TRIMAP podem ser consultados em (Clímaco et al., 1995). Em (Clímaco e Antunes, 1990) um modelo semelhante é usado para efectuar uma análise comparativa entre os métodos STEM, Zionts-Wallenius e TRIMAP.

Esta abordagem foi depois aprofundada no sentido do desenvolvimento de um novo modelo prestando particular atenção quer à gestão do lado da procura, quer aos impactes ambientais. Neste caso foram também consideradas três funções objectivo a minimizar: custo total de expansão, impacte ambiental associado à capacidade instalada e impacte ambiental associado à produção de energia. A gestão do lado da procura (demand-side management, DSM) é modelada como um novo grupo gerador, competindo com os grupos geradores do lado da oferta. O modelo considera cinco categorias de restrições relacionadas com a fiabilidade do sistema produtor, a disponibilidade das unidades geradoras, a capacidade do grupo gerador equivalente DSM, a capacidade total a instalar ao longo do período de planeamento e as emissões de poluentes. As tecnologias consideradas para expansão do sistema produtor são, para além do grupo equivalente DSM, fuel, carvão, nuclear e gás natural. O estudo foi realizado o usando o método TRIMAP e os detalhes da modelação, bem como alguns resultados ilustrativos, podem ser vistos em (Martins et al., 1996).

Outros dos estudos realizados dizem respeito ao planeamento estratégico da modernização de redes de telecomunicações, face à introdução de novas tecnologias e novos serviços. Em (Antunes et al., 1993) é desenvolvido um modelo de programação linear multiobjectivo baseado num diagrama de transição de estados (cujos nodos caracterizam uma linha de assinante num dado instante do período de planeamento). O modelo considera três funções objectivo que quantificam o valor associado à evolução das linhas de assinante, o grau de modernização associado à desejabilidade de novos serviços (ambos a maximizar), e o custo da dependência externa associada às estratégias de evolução (a minimizar). As restrições expressam limitações inferiores para a satisfação da procura estimada e para a penetração das tecnologias de suporte, limitações técnicas associadas à instalação de linhas e limitações de carácter orçamental. Os resultados foram obtidos usando o método TRIMAP.

Este modelo foi depois estendido para o estudo dos cenários de evolução de redes de acesso de banda larga, num contexto residencial e de pequenas e médias empresas (Antunes et al., 1998). Este modelo é baseado numa tabela de transições de estado que expressa as combinações admissíveis de categorias de serviço (serviço telefónico tradicional, serviço melhorado tipo acesso básico em banda estreita ISDN, serviço comutado de banda larga assimétrico, serviço comutado de banda larga simétrico, serviço distributivo de banda larga tipo CATV tipicamente não comutado e serviço comutado de banda larga avançado) e de arquitecturas tecnológicas para a rede de acesso (par de cobre, par de cobre melhorado tipo ADSL, híbrido fibra-coaxial HFC, FTTC).

Em (Alves et al., 1997) é apresentado um modelo de planeamento regional com quatro funções objectivo, baseado no quadro input-output de uma região de Portugal, que quantificam o consumo privado, o nível de emprego (ambas a maximizar), o deficit da balança comercial da região e o consumo de energia (ambas a minimizar). As restrições derivam de relações de equilíbrio entre os diferentes sectores da economia, limitações impostas na variação de algumas variáveis do modelo, bem como restrições definidoras (por exemplo, o consumo privado definido como uma função do rendimento das famílias). A tabela input-output está organizada em 12 sectores (uma agregação dos 49 sectores das contas nacionais). O output de cada sector é representado por uma variável de decisão. Todas as variáveis são consideradas exógenas à excepção da formação bruta de capital fixo e do consumo colectivo. Este estudo foi realizado com o SOMMIX, com o intuito de demonstrar as potencialidades no apoio à decisão de uma ferramenta baseada num painel de controlo em relação a métodos como o STEM e o Zionts-Wallenius.

O TRIMAP foi usado para o estudo das interacções entre o sistema energético, a economia a nível nacional e os impactes ambientais resultantes de diferentes políticas (Antunes et al., 2002). O modelo de programação linear multobjectivo, baseado numa tabela input-output na qual são considerados 44 sectores económicos, com dados representativos da realidade Portuguesa, inclui como funções objectivo o produto interno bruto, as importações de energia e a auto-produção de electricidade. As restrições dizem respeito à capacidade de armazenamento e stocks de segurança de hidrocarbonetos, às emissões de CO2, ao défice público, às limitações para as exportações e para as importações, à capacidade de produção da economia, ao valor acrescentado bruto e à balança de pagamentos.

CAPÍTULO 17

Algoritmos Evolutivos Multi-Objectivo

Ricardo H. C. Takahashi [*] *António Gaspar-Cunha* [**] *Carlos M. Fonseca* [***]

[*] *Departamento de Matemática*
Universidade Federal de Minas Gerais

[**] *Instituto de Polímeros e Compósitos / I3N*
Universidade do Minho

[***] *Departamento de Engenharia Informática*
Universidade de Coimbra

Os capítulos anteriores deste livro trataram de diferentes técnicas para resolver problemas de otimização formulados como problemas de escolha dos valores de um conjunto de variáveis de decisão satisfazendo um conjunto de restrições, de forma a minimizar (ou maximizar) uma determinada função (a função-objetivo). Na verdade, grande parte da teoria de otimização, desenvolvida ao longo do século XX, diz respeito a problemas formulados dessa maneira.

No entanto, muitos dos problemas encontrados no mundo prático seriam mais adequadamente formulados em termos de múltiplos objetivos. Por exemplo, em situações de projeto de sistemas de engenharia, normalmente é importante minimizar o custo de construção do sistema, ao mesmo tempo em que também se faz necessário otimizar índices de desempenho diversos. Para o fim de se estabelecer um exemplo concreto, imagine-se uma situação de projeto de um motor elétrico. Deseja-se: minimizar o custo de produção do motor; minimizar o peso final do motor; maximizar a eficiência energética do

motor; maximizar o conjugado eletromecânico fornecido; minimizar a interferência eletromagnética induzida pelo motor em outros equipamentos; maximizar o valor da sobretensão que o motor será capaz de suportar, de curta e de longa duração; etc. É importante notar que o resultado final do projeto, que será o motor efetivamente construído, deverá demonstrar um bom comportamento no que diz respeito a todos esses objetivos, de forma que o procedimento de projeto não poderá se resumir a uma otimização considerando um dos objetivos apenas. Além disso, deve-se notar que em alguma medida, os diferentes objetivos serão tipicamente contraditórios entre si. No exemplo do motor, pode-se perceber que, dentre as diferentes alternativas de projeto final do motor, existirão aquelas que terão menor custo de construção e menor eficiência energética, e outras de maior custo e mais eficientes. Tipicamente, não existirá um único motor que seja o de menor custo construtivo possível e simultaneamente também o de maior eficiência possível. Então, problemas com múltiplos objetivos tipicamente colocarão para o projetista um problema de tomada de decisão que envolve a análise das trocas (ou *trade-offs*) entre objetivos.

O problema de projeto de um motor elétrico, conforme enunciado no parágrafo anterior, se enquadra na classe dos chamados problemas de *otimização multiobjetivo*. Será visto em maior detalhe, nas próximas seções deste capítulo, que a formulação de tais problemas geralmente conduz a que todo um conjunto de soluções, chamadas *soluções eficientes*[1], que irão satisfazer à condição de otimalidade estabelecida em termos das várias funções-objetivo do problema. De maneira intuitiva, pode-se imaginar que essas seriam aquelas soluções que atingiram o limite de desempenho imposto pela física do problema, no sentido em que, por exemplo, para baixar o custo de produção do motor ainda mais, em relação a uma solução que já seja ótima, seja preciso sacrificar por exemplo um pouco da eficiência energética do motor. Devido à multiplicidade das soluções de problemas formulados dessa maneira, a solução completa de um problema de projeto (que necessariamente deverá indicar um único motor a ser fabricado, no caso do exemplo em questão) usualmente irá envolver duas etapas: a etapa de determinação das soluções eficientes, e a etapa de tomada de decisão, para seleção de uma única solução a ser efetivamente implementada.

Este capítulo irá tratar especificamente da etapa de geração das soluções eficientes. O capítulo 16 irá abordar a questão da tomada de decisão. É importante mencionar, neste ponto, que há três abordagens gerais para a integração da tomada de decisão em problemas de otimização multiobjetivo, que seriam com a decisão sendo tomada *a priori*, ou *progressivamente*, ou ainda *a posteriori*. Na primeira, *a priori*, os critérios a serem utilizados na tomada de decisão seriam explicitados antes da geração de qualquer solução eficiente. Esses critérios já poderiam, portanto, ser considerados no processo de otimização, de forma a conduzir diretamente a uma única solução que, além de ser eficiente, será também a preferida. Na segunda, *progressiva*, um tomador de decisão irá tomar conhecimento das possibilidades de solução existentes para o problema à medida em que estas forem sendo geradas, no decorrer da execução do algoritmo de otimização. Esse tomador de decisão irá então interagir com o algoritmo de otimização, guiando-o para a determinação das soluções eficientes que correspondam à sua preferência, evitando assim a necessidade da geração de todo o conjunto de soluções eficientes. Por fim, a terceira abordagem, *a posteriori*, irá supor que todo um conjunto de amostras do conjunto de soluções eficientes do problema de otimização multiobjetivo deverá ser gerado, para ser apresentado a um tomador de decisão. Esse tomador de decisão, após examinar as diferentes soluções contidas nesse conjunto, irá formar sua opinião a respeito de qual solução deverá ser preferida.

Esta última situação, da tomada de decisão *a posteriori*, foi assumida como contexto-padrão pela grande maioria dos estudos envolvendo a otimização multiobjetivo dentro da abordagem da computação evolutiva[2]. Este capítulo irá apresentar, especificamente, a abordagem da computação evolu-

[1] Na literatura são encontrados os seguintes sinônimos para *soluções eficientes*: soluções Pareto-ótimas, soluções não-dominadas, soluções não-inferiores.

[2] Recentemente, têm aparecido algumas exceções a esse padrão; ver, por exemplo, as referências (Battiti e Passerini, 2010; Deb et al., 2010; Karahan e Koksalan, 2010; Kim et al., 2011).

tiva para tratar os problemas de otimização multiobjetivo, de forma que será pressuposto aqui que o mecanismo de otimização deva cumprir o papel de gerar uma amostragem representativa do conjunto de soluções eficientes, antes de se iniciar qualquer interação com um tomador de decisão.

Deve-se mencionar que, antes do aparecimento da abordagem evolutiva, já existiam outras abordagens para o problema de otimização em contextos envolvendo múltiplos objetivos. Os métodos tradicionais de otimização, construídos para a minimização de um único objetivo, lidam com os problemas em que existem múltiplos objetivos relevantes de duas formas diferentes. Essas formas são: (i) constrói-se uma única função objetivo como uma soma ponderada das diversas funções de interesse; ou (ii) tratam-se todos os objetivos, menos um, como restrições, ficando apenas um objetivo formulado como função-objetivo a ser minimizada. Dessa forma, é possível que uma otimização de uma única função-objetivo conduza a uma solução que seja adequada no que diz respeito a vários objetivos diferentes. Há entretanto diversas possíveis objeções a essas abordagens, relacionadas com dificuldades de aplicação prática e com a perda de soluções possivelmente interessantes. Mas talvez a objeção mais importante a ser feita seja a de que os problemas com múltiplos objetivos normalmente envolvem escolhas dentre soluções que representam diferentes compromissos entre os diferentes objetivos. Portanto, para que a escolha seja feita adequadamente, é importante que o projetista tenha conhecimento das alternativas disponíveis, de forma a poder compará-las – o que significa que métodos que forneçam como saída uma única solução estarão omitindo informação essencial a respeito do problema.

Devido a tais considerações, foi desenvolvida uma teoria para fundamentar a otimização multiobjetivo, inicialmente no âmbito dos métodos determinísticos de otimização, que visava construir métodos capazes de gerar conjuntos de soluções pertencentes a aquele *conjunto de soluções eficientes*. A lógica básica desses métodos consistia em reformular o problema de otimização multiobjetivo, de forma tal a gerar um problema de otimização mono-objetivo (chamado *problema escalarizado*) que gerasse como solução um vetor de variáveis de decisão que garantidamente pertencesse ao conjunto de soluções eficientes. Esse problema seria parametrizado de tal forma que, variando-se os parâmetros do problema escalarizado, seria possível em princípio gerar todas as soluções pertencentes ao conjunto de soluções eficientes. Essa abordagem, por construção, iria requerer a execução completa de um algoritmo de otimização mono-objetivo para gerar cada uma das soluções pertencentes ao conjunto de soluções eficientes do problema multi-objetivo. As referências (Chankong e Haimes, 1983) e (Ehrgott, 2000) descrevem em detalhe essa abordagem.

Com o surgimento dos Algoritmos Evolutivos, capazes de fazer evoluir simultaneamente todo um conjunto de diferentes tentativas de solução – as *populações* – foi possível elaborar técnicas para a obtenção, em uma única execução do algoritmo, de todo um conjunto de soluções que representam diferentes compromissos entre os objetivos do problema. Essas técnicas, denominadas de MOEA, do inglês *Multi-Objective Evolutionary Algorithms*, ou de EMO, do inglês *Evolutionary Multicriteria Optimization*, vieram a ocupar uma posição de destaque dentre as metodologias capazes de tratar problemas de otimização multiobjetivo, constituindo hoje o elenco de ferramentas mais utilizado em problemas práticos dessa natureza (Deb, 2001; Coello Coello et al., 2002).

Diversas das razões que contribuíram para a popularidade dessa abordagem se aplicam tanto ao caso multiobjetivo quanto ao caso em que há uma única função-objetivo:

1. Diversas técnicas de otimização multiobjetivo clássicas requerem que os problemas sejam enquadrados em formulações específicas. Esse é o caso, por exemplo, de formulações que requerem funções-objetivo e restrições lineares com variáveis inteiras, ou funções-objetivo e restrições convexas, ou ainda funções-objetivo convexas com restrições lineares, ou outras. Os MOEA não apresentam esse tipo de limitação, apresentando grande flexibilidade no que diz respeito à estrutura das funções-objetivo e restrições do problema.

2. Os AE's são capazes de tratar problemas que envolvem funções multimodais, com múltiplas bacias de atração de diferentes mínimos locais. A maior parte dos algoritmos de otimização

tradicionais não contêm mecanismos para lidar adequadamente com esse tipo de função.

3. Como outras heurísticas, as técnicas de computação evolutiva admitem que se fixe o tempo de execução, de forma a retornar as melhores soluções encontradas até então. Embora não haja garantia de otimalidade exata dessas soluções, isso torna essa classe de métodos aplicável em problemas nos quais há a limitação do tempo disponível para a entrega da solução – o que inviabilizaria o uso de diversas técnicas existentes.

4. Os AEs são relativamente simples em termos computacionais e fáceis de implementar.

5. É fácil combinar (hibridizar) EAs com outras técnicas de optimização, tais como procura local e outras metaheuristicas;

6. Os AEs são facilmente adaptáveis devido ao facto do mecanismo evolutivo ser separado da representação do problema. Assim, um mesmo AE poderá ser usados em diferentes tipos de problemas, sendo normalmente necessário apenas o tratamento específico da codificação das variáveis do problema.

Uma vantagem adicional dos AE's multiobjetivo, que não se verifica no caso de uma única função objetivo, reside nos ganhos computacionais associados à evolução simultânea de um grande número de soluções, que irão gerar todo um conjunto de amostras do conjunto de soluções eficientes do problema, com uma única execução do algoritmo. A referência (Giel e Lehre, 2010) mostra formalmente essa vantagem, em termos da redução da complexidade computacional do problema.

A Optimização Multi-Objectivo (MOO) constitui hoje um tema de investigação importante para cientistas e engenheiros, não só devido à natureza multi-objectivo da maioria dos problemas do mundo real, mas também devido ao facto de existirem ainda muitos pontos de investigação em aberto nesta área.

Este capítulo está organizado nos seguintes tópicos:

- A seção 1 formaliza o problema da otimização multiobjetivo, estudando sua estrutura matemática e identificando as funções que devem ser executadas por algoritmos de otimização multiobjetivo.

- A seção 2 apresenta os elementos construtivos básicos dos algoritmos evolutivos multiobjetivo, mostrando que alguns desses elementos são idênticos a elementos análogos presentes nos algoritmos evolutivos mono-objetivo, enquanto outros são específicos para o caso multiobjetivo. Essa seção se encerra com a apresentação dos pseudo-códigos de dois AE's multiobjetivo clássicos, o NSGA-II (Non-Dominated Sorting Genetic Algorithm II) e o SPEA2 (Strength Pareto Evolutionary Algorithm 2).

- A seção 3 discute algumas funções que vêm sendo incorporadas a versões recentes de algoritmos evolutivos multiobjetivo, procurando também indicar algumas questões que são tema de pesquisa corrente na área.

1. Otimização Multiobjetivo

No capítulo 1, os problemas de otimização foram definidos como:

$$\mathbf{x}^* = \arg\min \mathbf{f}(\mathbf{x})$$

$$\text{sujeito a: } \begin{cases} g_1(\mathbf{x}) \leq 0 \\ g_2(\mathbf{x}) \leq 0 \\ \quad \vdots \\ g_m(\mathbf{x}) \leq 0 \end{cases} \tag{17.1}$$

Essa formulação ainda se aplica aos *problemas de otimização multiobjetivo*. O problema é formulado, nesse caso, como a questão de determinar o vetor $\mathbf{x}^*$ que minimiza $\mathbf{f}(\mathbf{x})$. Novamente, as funções de restrição $g_1(\mathbf{x})$ a $g_m(\mathbf{x})$ serão definidas como funções que são satisfeitas quando são menores ou iguais a zero. A diferença em relação ao caso de um único objetivo é que agora $\mathbf{f}(\mathbf{x})$ passa a ser definido como um vetor de funções-objetivo:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_p(\mathbf{x}) \end{bmatrix}$$

Imediatamente se coloca a questão: o que significa a *minimização* nesta nova situação? Um ponto central da teoria de otimização multiobjetivo é a definição de mínimo a ser adotada.

Cabe em primeiro lugar examinar por quê essa dificuldade não ocorria (ou, pelo menos, não ficava aparente) quando o problema tinha apenas uma função objetivo. Naquele caso, a função $f(\mathbf{x})$ retornava um único número real, ou seja, um escalar. Um conjunto de k números escalares a_i sempre pode ser *ordenado* usando a relação $\leq$, ou seja, sempre é possível escrever: $a_1 \leq a_2 \leq \ldots \leq a_k$. O *mínimo* de um conjunto de números pode ser definido como o elemento que aparece no extremo esquerdo da ordenação $a_1 \leq a_2 \leq \ldots \leq a_k$, ou seja, como o elemento que é menor ou igual que todos os demais. Note-se que pode haver mais de um elemento nessa situação, se o menor de todos ocorrer mais de uma vez. Assim, por exemplo, o conjunto $\{3, 7, 5, 3, 4, 2, 9, 2\}$ pode ser ordenado da seguinte forma: $2 \leq 2 \leq 3 \leq 3 \leq 4 \leq 5 \leq 7 \leq 9$. O elemento 2, que aparece no extremo esquerdo da ordenação, é chamado de *mínimo* do conjunto. O operador relacional de *menor ou igual* ($\leq$), que fez com que esse conjunto de números fosse ordenado dessa maneira, é chamado de *ordem total*, porque foi capaz de estabelecer uma única ordenação encadeando todos os elementos do conjunto.

A relação de menor ou igual, em essência, é o fundamento para definir o que é um mínimo. Essa relação permite comparar dois elementos, identificando-se o melhor dos dois como aquele que é menor, dentre os dois. Quando essa relação pode ser estabelecida entre todos os pares de elementos de um conjunto, numa ordenação total, passa a existir um elemento que é menor ou igual que todos os demais e, portanto, o melhor de todos do conjunto. Problemas de otimização visam identificar exatamente esse elemento "melhor de todos", definido dessa forma.

A relação de menor ou igual, quando utilizada para comparar vetores em uma comparação componente a componente, já não é mais capaz de gerar uma ordem total em geral. No caso de conjuntos de vetores, alguns pares de vetores serão relacionados dessa forma, ou seja, em alguns casos, cada componente $\mathbf{x}_i^1$ de um vetor $\mathbf{x}^1$ será menor ou igual a cada componente $\mathbf{x}_i^2$ de um vetor $\mathbf{x}^2$. Em outros casos, algumas componentes de $\mathbf{x}^1$ serão menores, e outras maiores, que as correspondentes componentes de $\mathbf{x}^2$. Essa é a origem da dificuldade para se definir o que é um mínimo no caso de vetores.

Para generalizar o conceito de mínimo, de forma a estabelecer um conceito aplicável a vetores, recorre-se à noção de *ordem parcial*. Define-se um operador relacional de *precedência*, $\overset{\text{P}}{\longleftarrow}$, que deverá

possuir as seguintes propriedades:

(i) $a \xleftarrow{\text{P}} a$ (reflexividade)

(ii) se $a \xleftarrow{\text{P}} b$ e $b \xleftarrow{\text{P}} a$ então $a = b$ (anti-simetria)

(iii) se $a \xleftarrow{\text{P}} b$ e $b \xleftarrow{\text{P}} c$ então $a \xleftarrow{\text{P}} c$ (transitividade)

Uma relação ($\xleftarrow{\text{P}}$) que tenha tais propriedades quando aplicada aos elementos de um conjunto é chamada de *ordem parcial*, porque ela faz com que seja possível definir uma precedência entre os elementos desse conjunto. No caso de conjuntos cujos elementos são vetores de números reais, é possível definir uma relação de precedência específica, a relação $\preceq$, que será útil para o propósito de reestabelecer uma noção de minimização significativa para o caso de conjuntos de vetores. Sejam os vetores **a** e **b**:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}$$

A relação $\preceq$ entre esses vetores é definida como:

$$\mathbf{a} \preceq \mathbf{b} \quad \text{se:} \quad a_i \leq b_i \text{ para todo } i = 1, \ldots, p \tag{17.2}$$

O operador $\preceq$ definido dessa forma claramente atende às propriedades (i), (ii) e (iii), e portanto é uma *ordem parcial*. Vetores de números reais então podem ser ordenados utilizando a relação $\preceq$, de forma que isso se aplica aos vetores de valores das funções-objetivo de problemas de otimização multiobjetivo. A grande diferença da nova situação, quando comparada à otimização de uma única função-objetivo, é que agora a relação de precedência pode não se aplicar a todos os pares de elementos de um conjunto de vetores.

Para exemplificar esse ponto, considere-se o seguinte conjunto:

$$\left\{ \begin{bmatrix} 3 \\ 7 \end{bmatrix}, \begin{bmatrix} 2 \\ 6 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \end{bmatrix}, \begin{bmatrix} 9 \\ 8 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \end{bmatrix} \right\} \tag{17.3}$$

Pode-se verificar que as seguintes relações são válidas:

$$\begin{bmatrix} 2 \\ 6 \end{bmatrix} \preceq \begin{bmatrix} 3 \\ 7 \end{bmatrix} \preceq \begin{bmatrix} 9 \\ 8 \end{bmatrix} \tag{17.4}$$

Também é válida a relação:

$$\begin{bmatrix} 4 \\ 5 \end{bmatrix} \preceq \begin{bmatrix} 4 \\ 5 \end{bmatrix} \preceq \begin{bmatrix} 9 \\ 8 \end{bmatrix} \tag{17.5}$$

No entanto, neste caso não é possível estabelecer relações utilizando o operador $\preceq$ entre um elemento $\begin{bmatrix} 4 \\ 5 \end{bmatrix}$ e o elemento $\begin{bmatrix} 2 \\ 6 \end{bmatrix}$, nem entre um elemento $\begin{bmatrix} 4 \\ 5 \end{bmatrix}$ e o elemento $\begin{bmatrix} 3 \\ 7 \end{bmatrix}$. Nesses casos, a primeira componente de um dos vetores é maior que a primeira do outro, enquanto o contrário ocorre na segunda componente. Por esse motivo, o operador $\preceq$ aplicado a conjuntos de vetores é apenas uma *ordem parcial*, e não uma *ordem total*. A figura 17.1 ilustra o significado da relação $\preceq$.

No caso de conjuntos cujos elementos são números escalares, pode-se verificar que o operador $\leq$, atende às propriedades (i), (ii) e (iii) e, além disso, para quaisquer dois escalares a_i e a_j sempre ocorrerá ou $a_i \leq a_j$ ou então $a_j \leq a_i$. Como vimos, por este motivo a relação $\leq$ é chamada de *ordem total* para conjuntos de números reais. Um conjunto ordenado com uma ordem total sempre terá um elemento que se encontra à esquerda de todos os demais do conjunto, quando este é ordenado.

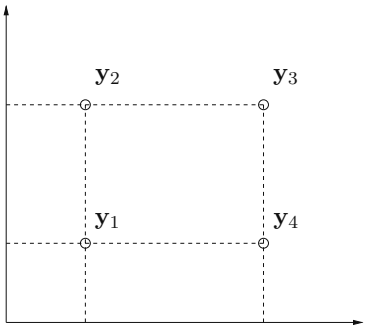


Figura 17.1: Os pontos $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$ e $\mathbf{y}_4$ são tais que $\mathbf{y}_1 \preceq \mathbf{y}_2$, $\mathbf{y}_1 \preceq \mathbf{y}_3$ e $\mathbf{y}_1 \preceq \mathbf{y}_4$, ou seja, $\mathbf{y}_1$ precede todos os demais. Além disso, $\mathbf{y}_2 \preceq \mathbf{y}_3$ e $\mathbf{y}_4 \preceq \mathbf{y}_3$, ou seja, $\mathbf{y}_3$ é precedido por todos os demais. Por fim, observa-se ainda que $\mathbf{y}_2 \not\preceq \mathbf{y}_4$, e $\mathbf{y}_4 \not\preceq \mathbf{y}_2$, ou seja, não há relação de precedência entre $\mathbf{y}_2$ e $\mathbf{y}_4$.

Esse elemento é chamado de *o menor elemento* do conjunto, ou de *mínimo* do conjunto. Já uma ordem parcial pode ou não levar a uma ordenação em que haja um único elemento do conjunto à esquerda de todos os demais. No caso do conjunto mostrado na equação (17.3), que é ordenado em duas sequências diferentes, cada sequência conta com um elemento mínimo diferente, $\begin{bmatrix} 4 \\ 5 \end{bmatrix}$ para uma sequência e $\begin{bmatrix} 2 \\ 6 \end{bmatrix}$ para a outra sequência. Se nesse conjunto fosse introduzido por exemplo o elemento $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$, este se tornaria o elemento mínimo das duas sequências, ou *o menor elemento* do conjunto, pois são válidas as relações $\begin{bmatrix} 0 \\ 0 \end{bmatrix} \preceq \begin{bmatrix} 4 \\ 5 \end{bmatrix}$ e $\begin{bmatrix} 0 \\ 0 \end{bmatrix} \preceq \begin{bmatrix} 2 \\ 6 \end{bmatrix}$, mesmo não havendo uma ordenação total do conjunto.

Como nem sempre existirá um elemento que seja *o menor* de um conjunto cuja ordenação é parcial, como no caso do conjunto em (17.3), o resultado de uma minimização nesse conjunto deve fazer referência a outro tipo de entidade. Considere-se o conjunto representado na equação (17.3). A ordem parcial $\preceq$ define duas sequências de ordenações diferentes que encadeiam os elementos desse conjunto, indicadas nas equações (17.4) e (17.5). Os elementos que são mínimos dessas duas sequências, no caso os elementos $\mathbf{a} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$ e $\mathbf{b} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$, não são comparáveis entre si, mas compartilham a característica de não terem nenhum outro elemento diferente deles que os preceda (note-se que o elemento $\mathbf{b}$ é precedido por outro elemento igual a ele). Ou seja, não existe no conjunto nenhum elemento $\mathbf{c}$ diferente de $\mathbf{a}$ para o qual seja válida a relação $\mathbf{c} \preceq \mathbf{a}$, nem nenhum elemento $\mathbf{c}$ diferente de $\mathbf{b}$ que satisfaça a relação $\mathbf{c} \preceq \mathbf{b}$. Esses elementos, que são mínimos de alguma sequência definida pela ordem parcial no conjunto, são chamados de *elementos minimais* do conjunto no que diz respeito a tal ordem parcial. Todo conjunto ordenado segundo uma ordem parcial sempre terá pelo menos um elemento minimal, e em geral poderá possuir vários elementos minimais. Na figura 17.1, o elemento $\mathbf{y}_1$ é o elemento minimal, considerando o conjunto constituído pelos elementos $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$ e $\mathbf{y}_4$.

Neste ponto estamos em posição de definir o que seriam as soluções do problema de otimização multiobjetivo (17.1). Considerem-se duas soluções factíveis $\mathbf{x}_a$ e $\mathbf{x}_b$ para o problema (ou seja, soluções que satisfazem as restrições). Se se verificar a relação $\mathbf{f}(\mathbf{x}_a) \preceq \mathbf{f}(\mathbf{x}_b)$ para $\mathbf{f}(\mathbf{x}_a) \neq \mathbf{f}(\mathbf{x}_b)$, a solução $\mathbf{x}_a$ será certamente melhor que a solução $\mathbf{x}_b$, pois os valores de todas as funções objetivo de $\mathbf{x}_a$ serão ou melhores ou no mínimo iguais em relação à outra solução. Então, as soluções minimais do conjunto de soluções factíveis, no que diz respeito à ordem parcial $\preceq$, coincidem com o que se espera que sejam "as

melhores soluções" do problema (17.1), e são chamadas de *soluções eficientes* desse problema. Como consequência, um problema de otimização multiobjetivo (17.1) irá possuir, em geral, um conjunto de soluções eficientes, que apenas em casos muito particulares irá se reduzir a uma única solução eficiente.

É importante neste ponto enunciar formalmente as definições de *dominância*, de *solução eficiente* e de *conjunto Pareto-ótimo* e *fronteira Pareto-ótima*:

Definição (Dominância): Sejam $\mathbf{x}_a$ e $\mathbf{x}_b$ dois vetores de decisão factíveis do problema de otimização multiobjetivo (17.1). Se $\mathbf{f}(\mathbf{x}_a) \preceq \mathbf{f}(\mathbf{x}_b)$ e $\mathbf{f}(\mathbf{x}_a) \neq \mathbf{f}(\mathbf{x}_b)$, então se diz que $\mathbf{x}_a$ *domina* $\mathbf{x}_b$. Essa situação é indicada pela notação: $\mathbf{x}_a \prec \mathbf{x}_b$.
Com certo abuso de linguagem também diremos aqui, quando não houver possibilidade de causar ambiguidade, que se $\mathbf{f}(\mathbf{x}_a) \preceq \mathbf{f}(\mathbf{x}_b)$ e $\mathbf{f}(\mathbf{x}_a) \neq \mathbf{f}(\mathbf{x}_b)$, então $\mathbf{f}(\mathbf{x}_a)$ *domina* $\mathbf{f}(\mathbf{x}_b)$.

Definição (Solução Eficiente): Um vetor de variáveis de decisão $\mathbf{x}^*$ factível é uma *solução eficiente* do problema de otimização multiobjetivo (17.1) se não existir qualquer outra solução factível desse problema que domine $\mathbf{x}^*$.
Também com abuso de linguagem diremos aqui, quando não houver possibilidade de causar ambiguidade, que se $\mathbf{f}(\mathbf{x}^*)$ não é dominada pela imagem de nenhuma outra solução, sendo $\mathbf{x}^*$ factível, então $\mathbf{f}(\mathbf{x}^*)$ é um *ponto eficiente*.

Definição (Conjunto Pareto-Ótimo): O conjunto $\mathcal{P}$ de todas as soluções eficientes do problema de otimização multiobjetivo (17.1) é denominado *conjunto Pareto-ótimo* desse problema.

Definição (Fronteira Pareto-Ótima): A imagem do conjunto $\mathcal{P}$ de todas as soluções eficientes do problema de otimização multiobjetivo (17.1), constituída de todos os pontos eficientes no espaço de objetivos, é denominada *fronteira Pareto-ótima* desse problema.

A figura 17.2 ilustra os conceitos de dominância, de ponto eficiente, e de fronteira Pareto-ótima.

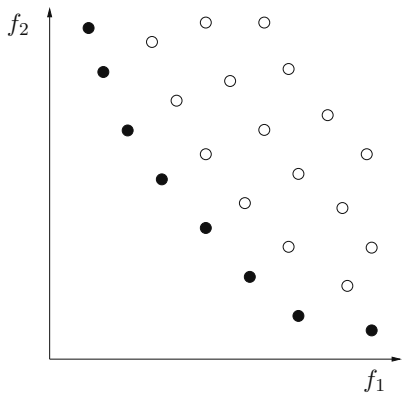


Figura 17.2: Os pontos brancos nesta figura são dominados por algum outro ponto, enquanto os pontos pretos, por sua vez, são não-dominados, no espaço de objetivos $\mathcal{Y}$. Desta forma, a fronteira Pareto-ótima relativa ao conjunto de todos os pontos representados na figura é constituída pelo conjunto dos pontos pretos, que são os pontos eficientes.

Finalmente podemos esboçar a tarefa que cabe a um algoritmo de otimização multiobjetivo. Idealmente, o resultado de uma otimização multiobjetivo deveria ter como resultado o conjunto $\mathcal{P}$, de todas as soluções eficientes do problema (17.1). No entanto, na prática, a geração numérica de todas as soluções contidas nesse conjunto não é possível em grande parte das vezes, devido à elevada car-

dinalidade desse conjunto. O caso extremo ocorre quando as variáveis de decisão do problema forem contínuas, quando a cardinalidade de $\mathcal{P}$ usualmente será infinita. Nesse caso, o conjunto Pareto-ótimo será um objeto contínuo ou contínuo por partes, no espaço $\mathbb{R}^n$, como mostrado na figura 17.5, adiante. Mas mesmo em problemas de variáveis discretas, muito frequentemente o conjunto Pareto-ótimo poderá conter um número muito elevado de elementos. Assim, o resultado esperado de um procedimento de otimização multiobjetivo normalmente é um conjunto de amostras do conjunto $\mathcal{P}$, que deve estar distribuído de forma a representar esse conjunto (ou seja, de forma a dar uma boa idéia do formato geométrico e da localização desse conjunto).

Interpretação Geométrica

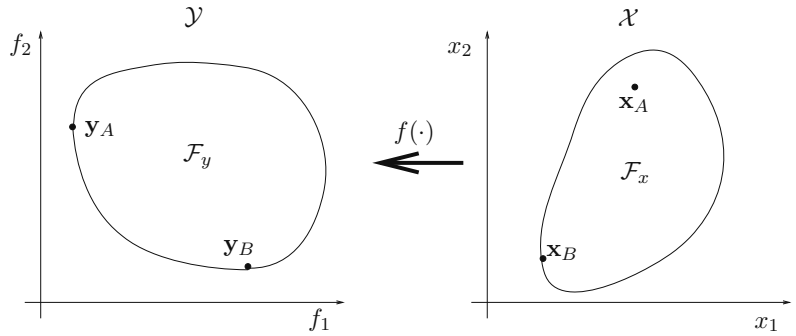


Figura 17.3: Mapeamento do espaço de parâmetros $\mathcal{X}$ no espaço de objetivos $\mathcal{Y}$ feito pela função $\mathbf{f}(\mathbf{x})$. A região factível no espaço de parâmetros é designada por $\mathcal{F}_x$, sendo o conjunto-imagem da função $f(\cdot)$ sobre o conjunto dos pontos factíveis designado por $\mathcal{F}_y$. O ponto de mínimo da função $f_1(\mathbf{x})$ no espaço de parâmetros é designado por $\mathbf{x}_A$, e sua imagem no espaço $\mathcal{Y}$ é designada por $\mathbf{y}_A$. O ponto de mínimo da função $f_2(\mathbf{x})$ no espaço de parâmetros é designado por $\mathbf{x}_B$, e sua imagem no espaço $\mathcal{Y}$ é designada por $\mathbf{y}_B$.

Uma interpretação geométrica pode ajudar a constituir um melhor entendimento da estrutura do problema de otimização multiobjetivo. Para o fim de estabelecer tal interpretação, iremos provisoriamente admitir que o vetor de variáveis de decisão $\mathbf{x}$ do problema de otimização multiobjetivo (17.1) tenha componentes reais, ou seja, $\mathbf{x} \in \mathbb{R}^n$. Nesse caso, o conjunto factível $\mathcal{F}_x$ será uma região do espaço de variáveis de decisão $\mathcal{X} = \mathbb{R}^n$, conforme encontra-se representado na figura 17.3. A imagem do vetor de funções-objetivo estará no espaço de objetivos $\mathcal{Y} = \mathbb{R}^p$, sendo que o conjunto dos pontos de $\mathcal{Y}$ que correspondem à imagem de pontos em $\mathcal{F}_x$ será denotado por $\mathcal{F}_y$. Deve-se notar que, nos eixos da representação cartesiana do espaço de objetivos $\mathcal{Y}$, estão representados os valores das diversas funções objetivo $f_1(\mathbf{x})$, $f_2(\mathbf{x})$, ..., $f_p(\mathbf{x})$. Na figura 17.3, o ponto $\mathbf{y}_A \in \mathcal{Y}$ corresponde ao vetor de objetivos que ocorre como imagem do ponto $\mathbf{x}_A$ que minimiza a função f_1. O ponto $\mathbf{y}_B \in \mathcal{Y}$ corresponde ao vetor de objetivos que ocorre como imagem do ponto $\mathbf{x}_B$ que minimiza a função f_2. Observe-se que embora um ponto de mínimo de alguma função f_i possa ocorrer no interior da região $\mathcal{F}_x$ (na figura 17.3, esse é o caso do ponto $\mathbf{x}_A$), os pontos correspondentes às imagens de todos os mínimos no espaço $\mathcal{Y}$ necessariamente se encontrarão na fronteira do conjunto $\mathcal{F}_y$ (vide o ponto $\mathbf{y}_A$).

A relação de dominância entre soluções factíveis pode ser representada geometricamente no espaço de objetivos $\mathcal{Y}$. O conceito de dominância encontra-se ilustrado na figura 17.4. Nessa figura, são mostrados os pontos $\mathbf{y}_A$ e $\mathbf{y}_B$ pertencentes ao espaço $\mathcal{Y}$. Um cone paralelo aos eixos coordenados do espaço $\mathcal{Y}$ é colocado com vértice no ponto $\mathbf{y}_B$. Esse cone será denominado *cone polar*. Todos os pontos no interior do cone polar dominam $\mathbf{y}_B$, de forma que $\mathbf{y}_B$ é dominado por $\mathbf{y}_A$. Mais precisamente, devemos dizer que o ponto $\mathbf{x}_B$, cuja imagem é $\mathbf{y}_B$, é dominado pelo ponto $\mathbf{x}_A$, cuja imagem é $\mathbf{y}_A$. A partir desta figura, pode-se inferir que quando se coloca um cone polar sobre a imagem $\mathbf{y}^*$ de uma

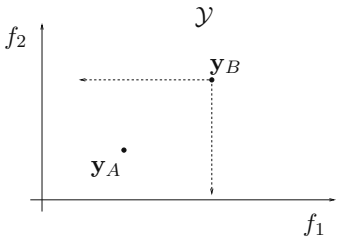


Figura 17.4: Representação da dominância no espaço de objetivos $\mathcal{Y}$. O ponto $\mathbf{y}_B$ será dominado por qualquer ponto localizado no cone com vértice em $\mathbf{y}_B$ (cone polar) que se encontra representado na figura. Por exemplo, $\mathbf{y}_A$ domina $\mathbf{y}_B$.

solução eficiente $\mathbf{x}^*$, não pode haver, no interior desse cone, a imagem de nenhuma outra solução factível (do contrário, essa solução não seria eficiente).

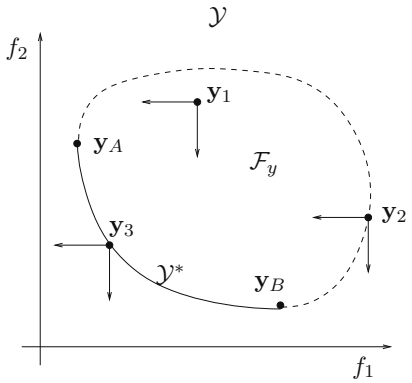


Figura 17.5: A fronteira de Pareto $\mathcal{Y}^* \subset \mathcal{Y}$, associada ao conjunto-imagem $\mathcal{F}_y$ da função $\mathbf{f}(\mathbf{x})$ encontra-se representada em linha contínua. O restante da fronteira do conjunto $\mathcal{F}_y$ está representado em linha tracejada. O conjunto $\mathcal{Y}^*$ possui extremos nos pontos $\mathbf{y}_A$ e $\mathbf{y}_B$, correspondentes aos mínimos individuais das funções $f_1(\mathbf{x})$ e $f_2(\mathbf{x})$. O cone polar colocado sobre o ponto $\mathbf{y}_1$, situado no interior do conjunto-imagem $\mathcal{F}_y$, terá interseção não vazia com esse conjunto. Todos os pontos situados nessa interseção dominam $\mathbf{y}_1$, de forma que este não constitui uma solução eficiente. O ponto $\mathbf{y}_2$, situado agora na fronteira de $\mathcal{F}_y$, tem seu cone polar ainda apresentando interseção não vazia com $\mathcal{F}_y$. Como $\mathbf{y}_2$ é dominado por todos os pontos situados nessa interseção, então $\mathbf{y}_2$ também não pode ser uma solução eficiente. Finalmente, o ponto $\mathbf{y}_3$, situado em uma parte da fronteira de $\mathcal{F}_y$ que faz parte da fronteira Pareto-ótima, tem seu cone polar situado de tal forma que sua interseção com o conjunto $\mathcal{F}_y$ é agora vazia. Como não há nenhuma solução que domina $\mathbf{y}_3$ (se existisse, esta estaria no cone polar), então $\mathbf{y}_3$ tem de ser uma solução eficiente do problema.

Essa observação tem como consequência que todo ponto $\mathbf{y}_A$ pertencente ao interior do conjunto $\mathcal{F}_y$ será dominado, pois o interior de um cone polar com vértice em $\mathbf{y}_A$ será sempre não-vazio, e haverá portanto algum ponto nesse cone que dominará $\mathbf{y}_A$. Assim, todo ponto eficiente terá de ter sua imagem não no interior, mas na fronteira do conjunto $\mathcal{F}_y$, de forma que um cone polar situado sobre um desses pontos não terá nenhuma interseção com o próprio conjunto $\mathcal{F}_y$, de forma que não exista nenhum ponto que possa dominar esses pontos eficientes. Essa estrutura do problema está representada na figura 17.5. Por esse motivo, o conjunto-imagem do conjunto Pareto-ótimo é usualmente denominado *fronteira Pareto-ótima* do problema de otimização multiobjetivo.

Finalmente, pode-se estabelecer o que seria o conjunto de soluções $\hat{\mathcal{X}}^*$ que se espera que sejam determinadas a partir de um procedimento de otimização multiobjetivo. Espera-se, desse procedimento, que produza: (i) um conjunto de soluções que esteja contido no conjunto Pareto-ótimo ($\hat{\mathcal{X}}^* \subset \mathcal{X}^*$); e (ii) que represente uma amostragem do conjunto Pareto-ótimo capaz de descrever esse conjunto,

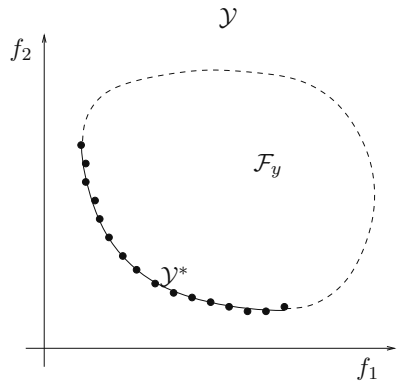


Figura 17.6: A fronteira de Pareto $\mathcal{Y}^* \subset \mathcal{Y}$, associada ao conjunto-imagem $\mathcal{F}_y$ da função $\mathbf{f}(\mathbf{x})$ encontra-se representada em linha contínua. Essa fronteira Pareto-ótima encontra-se amostrada por um conjunto de pontos $\hat{\mathcal{Y}}^*$ que representam $\mathcal{Y}^*$, no sentido de que todo ponto de $\mathcal{Y}^*$ está próximo (e portanto representado) por pelo menos um ponto no conjunto $\hat{\mathcal{Y}}^*$.

no sentido de que toda solução pertencente ao conjunto $\mathcal{Y}^*$ tenha pelo menos um ponto de $\hat{\mathcal{X}}^*$ em sua proximidade, e que portanto a "represente". Na maioria dos algoritmos de otimização multi-objetivo que foram desenvolvidos até hoje, essa proximidade é medida no conjunto-imagem $\mathcal{Y}$, de forma que esses algoritmos são construídos de forma a produzir uma amostragem representativa da fronteira Pareto-ótima $\mathcal{Y}^*$. Deve-se notar que uma amostragem representativa da fronteira Pareto-ótima $\mathcal{Y}^*$ não necessariamente implica uma correspondente amostragem representativa do conjunto Pareto-ótimo $\mathcal{X}^*$, e vice-versa. A figura 17.6 mostra um conjunto $\hat{\mathcal{Y}}^*$ de amostras que seriam, nesse sentido, representativas do conjunto $\mathcal{Y}^*$, e que portanto deveriam ser o resultado de um algoritmo de otimização multiobjetivo.

2. Algoritmos Evolutivos Multiobjetivo

Algoritmos evolutivos, como foi visto nos capítulos anteriores deste livro, são constituídos essencialmente por operações de:

- Variação: a partir de soluções anteriores, devem ser geradas novas soluções;
- Atribuição de Aptidão: a cada solução gerada pelo algoritmo, o algoritmo deve atribuir um valor de aptidão, a ser usado para comparar essa solução com outras soluções;
- Seleção: dadas as aptidões de várias soluções disponíveis após a aplicação das operações de variação, devem-se escolher algumas para serem mantidas na população, enquanto outras serão eliminadas. Essa escolha é feita
- Arquivamento: as melhores soluções até então encontradas pelo algoritmo devem ser armazenadas.

Nos algoritmos genéticos mono-objetivo, por exemplo:

- Os operadores genéticos de mutação e de cruzamento exercem a função de mecanismos de variação.
- Dependendo do mecanismo de seleção empregado, a atribuição de aptidão pode ser feita com a atribuição do próprio valor da função-objetivo, ou utilizando uma transformação não-linear

desse valor, ou ainda pelo uso de um ranqueamento das soluções. Às vezes, o mecanismo de atribuição de aptidão inclui também um operador de nicho, que tem a função de aumentar a aptidão relativa de soluções que se localizem em regiões pouco exploradas do espaço, de forma a gerar uma pressão para o aumento da diversidade da população.

- As seleções baseadas em roleta (com diversas variações) podem utilizar aptidões construídas a partir de transformações não-lineares do valor de função objetivo, ou a partir de ranqueamentos. Já as seleções baseadas em torneio normalmente utilizam a aptidão diretamente expressa como o valor da função-objetivo.

- Os AG's mono-objetivo normalmente contam com um arquivamento baseado em elitismo, que às vezes encontra-se implícito no mecanismo de seleção.

Outros tipos de AE's contam com diferentes operadores, que sempre deverão executar essas funções.

Algorimos Evolutivos Multiobjetivo também deverão contar com mecanismos que executem essas quatro funções. Normalmente, é possível reaproveitar os mesmos mecanismos de variação utilizados pelo correspondente algoritmo mono-objetivo, para o fim de construção de uma versão multiobjetivo do algoritmo. Assim, as operações de mutação e de cruzamento de um AG mono-objetivo poderão ser utilizadas, *ipsis literis*, em versões multiobjetivo do mesmo algoritmo. Também a função de seleção pode ser realizada, no caso multiobjetivo, com mecanismos idênticos aos empregados no caso mono-objetivo. Já as funções de atribuição de aptidão e de arquivamento irão requerer estruturas específicas para o caso multiobjetivo. A discussão sobre esse ponto é apresentada nas próximas subseções.

Nesta seção, serão estudados em particular os algoritmos *Non-Dominated Sorting Genetic Algorithm II*, ou NSGA-II, proposto por (Deb et al., 2002), e *Strength Pareto Evolutionary Algorithm 2*, ou SPEA2, proposto por (Zitzler et al., 2001). Esses algoritmos recebem tal destaque porque são os dois algoritmos evolutivos multiobjetivo que vieram sendo mais amplamente empregados nesta última década, sendo em grande parte responsáveis pela rápida popularização da otimização multiobjetivo evolutiva. Esses algoritmos são os representantes mais ilustres da chamada *2ª geração* de EA's multiobjetivo, caracterizada pelo uso de técnicas de arquivamento de soluções, em combinação com a seleção pela dominância de Pareto. Hoje já existem algoritmos mais avançados que, não obstante, ainda não atingiram tamanha disseminação na comunidade científica e nos campos de aplicação da computação evolutiva. Na próxima seção, serão comentados alguns desses desenvolvimentos posteriores ao NSGA-II e ao SPEA2.

Aptidão em AE's Multiobjetivo

A atribuição de aptidão em AE's multiobjetivo deve ser feita de forma a cumprir os seguintes requisitos:

- Se existir uma relação de dominância entre duas soluções, $\mathbf{x}_a \prec \mathbf{x}_b$, então a aptidão da solução $\mathbf{x}_a$ deve necessariamente ser maior que a da solução $\mathbf{x}_b$.

- Caso não haja relação de dominância entre duas soluções, ou seja, $\mathbf{x}_a \not\prec \mathbf{x}_b$ e $\mathbf{x}_b \not\prec \mathbf{x}_a$, então deve-se fazer a atribuição de aptidão a essas soluções de forma que a solução mais importante para garantir a representatividade da amostragem do conjunto de Pareto (normalmente aquela com menor número de vizinhos sem relação de dominância) tenha maior aptidão.

O primeiro desses requisitos é necessário para que o algoritmo convirja para o conjunto Pareto-ótimo, no qual estarão soluções cuja aptidão deverá ser maior que a de qualquer solução fora desse conjunto. O segundo requisito é também indispensável quando se assume que o propósito de um procedimento de otimização multiobjetivo seja a geração de um conjunto de amostras do conjunto Pareto-ótimo que seja representativo.

São possíveis diferentes maneiras de construir mecanismos de atribuição de aptidão que atendam a tais requisitos. Esses mecanismos devem partir de um índice que garanta o ordenamento de acordo com as relações de precedência. Alguns diferentes índices utilizados na literatura são:

Índice de Fronteira (I_f): Atribui-se o índice 0 a todas as soluções não-dominadas da população (fronteira não-dominada). Essas soluções são então excluídas do conjunto de trabalho e, sobre as soluções restantes, determinam-se as soluções não-dominadas relativas a esse conjunto, que corresponde à nova fronteira não-dominada, relativa ao novo conjunto. A essas soluções atribui-se o índice 1, e à fronteira subsequente, obtida no passo seguinte, o índice 2, e assim por diante, aumentando-se o índice de uma unidade para cada nova fronteira, até que todos os pontos da população estejam rotulados.

Índice de Dominação (I_d): Realiza-se a contagem do número de soluções que dominam cada uma das soluções contidas na população. Às soluções não dominadas por nenhuma outra, atribui-se o índice 0. Às soluções dominadas por uma única outra solução, atribui-se o índice 1. Às soluções dominadas por duas outras soluções, atribui-se o índice 2, e assim por diante.

Índice de Força-Pareto (I_p): Realiza-se a contagem do número de soluções que são dominadas por cada uma das soluções contidas na população. Às soluções não dominadas por nenhuma outra, atribui-se o índice 0. Como índice de cada solução dominada $\mathbf{x}^i$, totaliza-se a soma do número de soluções que são dominadas pelas soluções que dominam $\mathbf{x}^i$.

O índice I_f foi inicialmente empregado no algoritmo NSGA (Srinivas e Deb, 1994), e integra também o algoritmo NSGA-II (Deb et al., 2002). O índice I_d foi utilizado no algoritmo MOGA (Fonseca e Fleming, 1993, 1995). O índice I_p foi proposto junto com o algoritmo SPEA2 (Zitzler et al., 2001). Além dessas duas alternativas, são possíveis outras formas de se criarem índices que preservem as relações de precedência. Chamamos particularmente a atenção para a metodologia utilizada no algoritmo IBEA (Zitzler e Kunzli, 2004), que utiliza uma lógica bem distinta dessas acima.

A figura 17.7 ilustra os índices obtidos dessas formas. Deve-se notar que os índices atribuídos às soluções não-dominadas da população coincidem nos três casos, sendo sempre 0. Já as soluções de alguma forma dominadas são ordenadas de maneiras diferentes segundo os diferentes índices. Nos três casos, entretanto, garante-se que soluções de mesmo índice não tenham relação de precedência entre si. Também se garante que, sempre que houver relação de precedência entre duas soluções, a solução que domina irá possuir índice menor que o da solução que é dominada, ou seja:

$$\mathbf{x}_a \prec \mathbf{x}_b \;\Rightarrow\; I_*(\mathbf{x}_a) < I_*(\mathbf{x}_b),$$

sendo isso válido para I_* dado por I_f, por I_d ou por I_p. A partir deste ponto, para falar genericamente de um indicador que pode ser qualquer desses três, utilizaremos a notação I_*. Nos algoritmos a serem apresentados a seguir, a notação $I_*(\mathcal{P})$ será utilizada para representar o vetor de resultados do cálculo do valor do índice I_* para cada elemento de um conjunto $\mathcal{P}$ de pontos do espaço de variáveis de decisão.

Após o cômputo do índice I_* sobre o conjunto $\mathcal{P}$ dos pontos pertencentes à população de um AE multiobjetivo, já passa a existir uma ordenação das soluções pertencentes a essa população. Se um desses índices fosse utilizado como valor de aptidão, no algoritmo, pode-se perceber que seria induzida uma pressão seletiva que iria favorecer a seleção de indivíduos de maior precedência, na ordem parcial $\preceq$. Dessa forma, a população iria convergir em direção ao conjunto Pareto-ótimo do problema de otimização multiobjetivo.

Entretanto, deve-se notar que a simples aplicação de um desses índices como valor da aptidão dos indivíduos não iria produzir o efeito de que a população convergisse para amostras bem distribuídas no conjunto Pareto-ótimo, pois não haveria nenhum mecanismo causando o efeito de uma distribuição

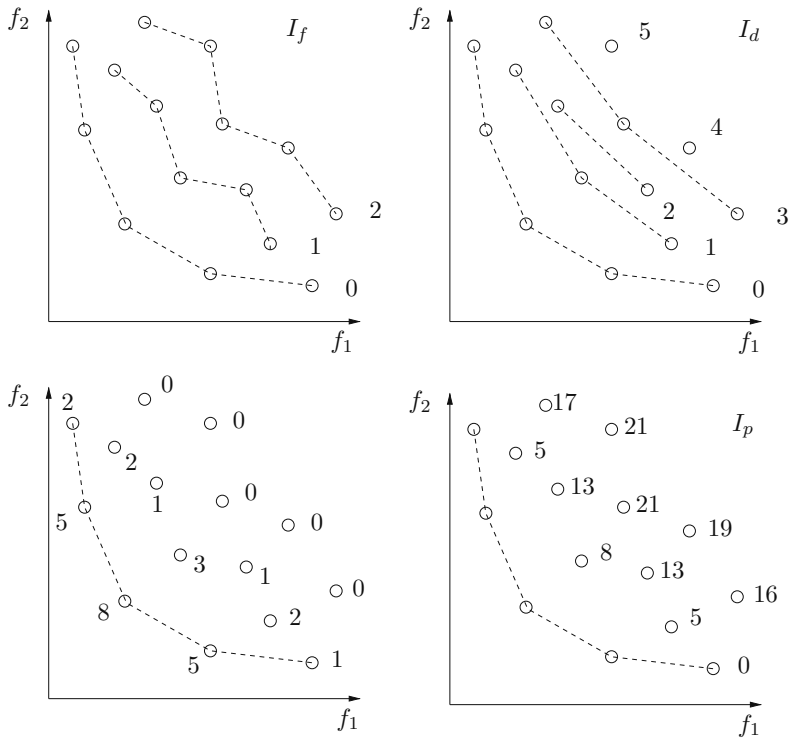


Figura 17.7: Exemplo de cômputo do Índice de Fronteira (I_f, no canto superior esquerdo), do Índice de Dominação (I_d, no canto superior direito) e do Índice de Força-Pareto (I_p, no canto inferior esquerdo), para o mesmo conjunto de pontos. No canto inferior esquerdo, estão registrados, para cada ponto, o número de outros pontos que ele domina. Esse número é utilizado no cômputo no índice I_p.

representativa do conjunto[3]. Por esse motivo, torna-se necessário acrescentar algum fator que produza um refinamento do ordenamento baseado no índice I_* que seja capaz de induzir essa boa distribuição. Esse fator de correção deve preservar a ordem relativa entre as soluções que tiverem índices I_* diferentes, pois essa ordem é a garantia de que as relações de precedência sejam atendidas; assim, o fator de correção usualmente é aplicado de forma a apenas desempatar as soluções que tiverem o mesmo índice. Duas maneiras de realizar tal correção são:

Distância de Aglomeração (δ_a): Em problemas com dois objetivos, faz-se a ordenação das q soluções de mesmo índice I_* (essas soluções, portanto, não têm relação de dominância entre si), de acordo com a ordem crescente de um dos objetivos (por exemplo f_1), ficando estas soluções rotuladas com índices que vão de $i = 1$ até $i = q$. Consequentemente, essas soluções ficarão ordenadas pela ordem decrescente do outro objetivo (f_2). Para cada solução de $i = 2$ até $i = q - 1$ calcula-se o número:

$$\delta_a(\mathbf{x}^i) = [f_2(\mathbf{x}^{i-1}) - f_2(\mathbf{x}^{i+1})] + [f_1(\mathbf{x}^{i+1}) - f_1(\mathbf{x}^{i-1})]$$

Esse número é proporcional ao perímetro do retângulo cujos vértices opostos encontram-se sobre as soluções vizinhas da solução i, no espaço de objetivos. Quanto maior o número $\delta_a(\mathbf{x}^i)$, mais distante a solução i estará das vizinhas, e portanto mais importante será esta solução para

[3] O índice I_p até possui algum efeito de evitar a aglomeração de pontos em uma pequena região, causado pela contagem das dominâncias herdadas pelos pontos. No entanto, deve-se notar que esse índice não consegue, por exemplo, discriminar entre duas soluções não-dominadas diferentes.

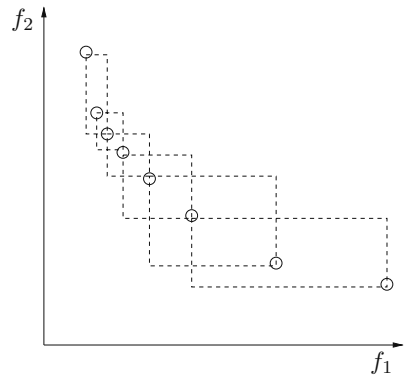


Figura 17.8: Interpretação geométrica da distância de aglomeração δ_a. Cada solução $\delta_a(\mathbf{x}^i)$ tem sua imagem no espaço de objetivos, $\mathbf{y}^i$, no interior de um retângulo cujos vértices opostos estão localizados nas imagens vizinhas $\mathbf{y}^{i-1}$ e $\mathbf{y}^{i+1}$. O indicador $\delta_a(\mathbf{x}^i)$ corresponde à metade do perímetro desse retângulo (ou seja, a soma de sua base mais sua altura).

garantir a representatividade do conjunto de soluções. Por convenção, as soluções extremas, com $i = 1$ e $i = q$, terão $\delta_a(\mathbf{x}^1) = \infty$ e $\delta_a(\mathbf{x}^q) = \infty$, o que significa que essas soluções são consideradas mais importantes que quaisquer outras, para garantir a representatividade do conjunto. A figura 17.8 ilustra o significado de δ_a.

k-Vizinhança (δ_k): Estabelece-se inicialmente um valor para k. Um possível critério para se definir k pode ser, por exemplo, fazer $k = \sqrt{N}$, sendo N o tamanho da população. Calcula-se então a distância (por exemplo euclidiana), no espaço de objetivos, entre cada par de soluções pertencentes à população. Para cada solução $\mathbf{x}^i$ determina-se a distância $\delta_k(\mathbf{x}^i)$ que corresponde a k-ésima menor distância de $\mathbf{x}^i$ em relação às demais soluções da população.

Tanto o número $\delta_a(\mathbf{x}^i)$ quanto o número $\delta_k(\mathbf{x}^i)$ podem ser interpretados como medidas da densidade da amostragem existente na região ao redor do ponto $\mathbf{x}^i$. Assim, para que ocorra uma melhor amostragem do conjunto Pareto-ótimo, o algoritmo deverá privilegiar, a cada momento, a seleção de soluções em regiões menos densamente ocupadas, ou seja, soluções com menor valor de δ_a ou de δ_k. O indicador δ_a foi proposto junto com o algoritmo NSGA-II (Deb et al., 2002). Já o indicador δ_k foi empregado no algoritmo SPEA2 (Zitzler et al., 2001).

Deve estar claro que é possível definir outras medidas de densidade de amostragem capazes de executar função semelhante à desempenhada por δ_a e δ_k. Por exemplo, o algoritmo RPSGA (Gaspar-Cunha, 2000; Gaspar-Cunha e Covas, 2004) utiliza uma técnica de *clusterização* proposta por (Roseman e Gero, 1985) para gerar uma medida desse tipo. Para fazer referência, genericamente, a qualquer medida de densidade com tais propriedades, utilizaremos aqui a notação δ_*. A notação $\delta_*(I_*, \mathcal{P})$ será utilizada para representar o vetor de resultados do cálculo do valor do indicador δ_* para cada elemento de um conjunto $\mathcal{P}$ de pontos do espaço de variáveis de decisão, após a aplicação prévia da operação $I_*(\mathcal{P})$ sobre os elementos desse conjunto.

Finalmente, dados um índice I_*, relativo às relações de precedência entre soluções, e um índice δ_*, ligado à densidade de amostragem, pode-se construir um único ranqueamento das soluções pertencentes à população do AE, pela aplicação primeiro de I_*, e depois de δ_* para desempatar as soluções de mesmo valor de I_*. A ordem total implícita nesse ranqueamento será utilizada para criar uma pressão seletiva dirigida à obtenção de um conjunto de amostras pertencentes ao conjunto Pareto-ótimo e que ainda constitua uma amostragem representativa desse conjunto. Assim, dadas N soluções $\mathbf{x}_1, \ldots, \mathbf{x}_N$, um ranqueamento é produzido de forma a associar um número inteiro de 1 até N a cada solução. Essa função de ranqueamento $R(\mathbf{x})$ é tal que:

- Se $I_*(\mathbf{x}_a) < I_*(\mathbf{x}_b)$ então $R(\mathbf{x}_a) < R(\mathbf{x}_b)$
- Se $I_*(\mathbf{x}_a) = I_*(\mathbf{x}_b)$ então:
  - Se $\delta_*(\mathbf{x}_a) < \delta_*(\mathbf{x}_b)$ então $R(\mathbf{x}_a) < R(\mathbf{x}_b)$
  - Se $\delta_*(\mathbf{x}_b) < \delta_*(\mathbf{x}_a)$ então $R(\mathbf{x}_b) < R(\mathbf{x}_a)$
  - Se $\delta_*(\mathbf{x}_b) = \delta_*(\mathbf{x}_a)$ então sorteia-se $R(\mathbf{x}_b) < R(\mathbf{x}_a)$ ou $R(\mathbf{x}_a) < R(\mathbf{x}_b)$

Deve-se notar que a aplicação de I_* e δ_* pode ainda deixar lugar a algumas soluções ficarem empatadas, com mesmo valor dos dois índices. No entanto, este será um evento relativamente raro, ao contrário do que ocorreria pela aplicação apenas de I_*. No caso desse tipo de empate, o desempate é feito aleatoriamente.

O ranqueamento $R(\mathbf{x})$ pode enfim ser utilizado para definir a aptidão[4], que será utilizada em um mecanismo de seleção, seja ele do tipo *roleta* ou *torneio*, exatamente como nos AG's mono-objetivo. A notação $\mathbf{R}(I_*, \delta_*, \mathcal{P})$ será utilizada aqui para representar o vetor de resultados do ranqueamento dos pontos pertencentes ao conjunto $\mathcal{P}$, após a aplicação prévia das operações $I_*(\mathcal{P})$ e $\delta_*(I_*, \mathcal{P})$ sobre o conjunto. Já a notação

$$\mathcal{V} \leftarrow \text{seleciona}(I_*, \delta_*, \mathbf{R}, \mathcal{P}, n)$$

irá representar a operação de atribuir ao conjunto $\mathcal{V}$ as n soluções pertencentes ao conjunto $\mathcal{P}$ que ocupam as n primeiras soluções no ranqueamento $\mathbf{R}(I_*, \delta_*, \mathcal{P})$.

Arquivamento em AE's Multiobjetivo

No caso dos algoritmos evolutivos mono-objetivo, o arquivamento da melhor solução é uma operação simples, que envolve guardar o vetor de variáveis de decisão $\mathbf{x}$ que represente, em cada momento, a melhor solução até então encontrada pelo algoritmo durante sua execução.

No caso de algoritmos multiobjetivo, essa função de arquivamento é significativamente mais complexa. Para situar a dificuldade, suponha-se inicialmente o seguinte procedimento ingênuo de arquivamento que poderia ser empregado pelo menos como conceito:

- Seja um conjunto $\mathcal{P}_{k-1}$ que constitui o arquivo de todas as soluções não-dominadas encontradas até a iteração $k-1$ do algoritmo.
- Na iteração k, para cada nova solução $\mathbf{x}_i$ encontrada, são feitas as seguintes operações:
  - Se $\mathbf{x}_i$ é dominada por algum elemento de $\mathcal{P}_{k-1}$, então o arquivo é mantido inalterado (a solução $\mathbf{x}_i$ não é introduzida no arquivo).
  - Se $\mathbf{x}_i$ domina uma ou mais soluções contidas em $\mathcal{P}_{k-1}$, então as soluções dominadas são excluídas do arquivo, e a solução $\mathbf{x}_i$ é incluída no arquivo.
  - Se $\mathbf{x}_i$ não tem relação de dominância (nem domina nem é dominada) com nenhuma das soluções contidas em $\mathcal{P}_{k-1}$, então $\mathbf{x}_i$ trata-se de uma nova estimativa de solução não-dominada, sendo incluída no arquivo.
- A execução dessas operações para todas as novas soluções geradas na iteração k, sobre o arquivo $\mathcal{P}_{k-1}$ vindo da iteração anterior, irá gerar o novo arquivo $\mathcal{P}_k$ ao final da iteração k.

[4] Da maneira como foi definido $R(\mathbf{x})$, quanto menor o seu valor, maior será a aptidão do indivíduo, e portanto maior a chance dele ser selecionado.

A dificuldade com esse procedimento é que muito frequentemente a cardinalidade do conjunto Pareto-ótimo de um problema de otimização multiobjetivo será muito elevada, ou até mesmo infinita, como no caso da maioria dos problemas de variáveis contínuas. Assim, haverá uma tendência para que esse procedimento gere um arquivo $\mathcal{P}_k$ que irá crescer muito rapidamente. Isso causa problemas não apenas relacionados com o armazenamento de um grande número de soluções, mas também com o esforço computacional para avaliar novas soluções, pois cada nova solução tem de ser comparada com todas as soluções do arquivo.

É preciso, portanto, de alguma forma limitar o número de soluções a serem armazenadas no arquivo. Consequentemente, a questão da *amostragem representativa*, discutida anteriormente no contexto da construção da função de aptidão $\mathbf{R}(I_*, \delta_*, \mathcal{P})$, também será relevante para o arquivamento, pois será necessário, dentre as diferentes alternativas para arquivar soluções não-dominadas, escolher uma estratégia que garanta essa representatividade, ou diversidade, do conjunto arquivado.

Suponha-se que deva ser montado um arquivo que irá conter as q melhores soluções que o algoritmo tiver encontrado até um dado momento. As seguintes hipóteses podem ocorrer:

- É possível que, até o momento em questão, ainda não tenham sido encontradas q soluções não-dominadas. Nesse caso, uma alternativa é deixar o arquivo incompleto, contendo apenas as soluções não-dominadas até então determinadas. Outra alternativa é também incluir no arquivo algumas das soluções dominadas. Para se proceder a essa inclusão, é possível simplesmente utilizar o ranqueamento $\mathbf{R}(I_*, \delta_*, \mathcal{P})$, conforme construído anteriormente, para escolher as soluções a serem acrescentadas.

- É possível também que num dado momento tenham sido determinadas exatamente q soluções não-dominadas. Nesse caso, essas soluções deverão constituir o arquivo.

- Por fim, é possível ainda que o número de soluções não-dominadas disponível, num dado momento, seja maior que o tamanho do arquivo, q. Nesse caso, algumas soluções não-dominadas deverão ser descartadas. Novamente, uma solução simples é utilizar o ranqueamento $\mathbf{R}(I_*, \delta_*, \mathcal{P})$, preservando as soluções que ocupam as q primeiras posições do ranking e descartando as demais. No entanto, é comum o uso de outro tipo de estratégia para efetuar esse descarte.

No caso do algoritmo NSGA-II (Deb et al., 2002), o arquivamento é feito utilizando-se a própria população como arquivo de tamanho N. Após a geração de N novas soluções por meio dos operadores de variação, as N melhores soluções do conjunto de $2N$ elementos que contém as N novas soluções e mais as N soluções da população anterior são diretamente selecionadas utilizando o ranqueamento $\mathbf{R}(I_f, \delta_a, \mathcal{P})$, de forma determinística. Assim, o arquivamento das melhores soluções fica implícito no próprio mecanismo de seleção da nova população.

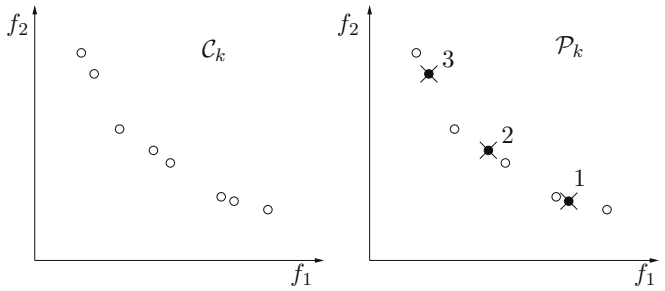


Figura 17.9: Representação da sequência de descarte de pontos utilizada no arquivamento do algoritmo SPEA2.

No caso do algoritmo SPEA2 (Zitzler et al., 2001), é empregado um arquivo das melhores soluções, mantido separado da população. Em caso do número de soluções não-dominadas ser menor que q, o

arquivo é completado simplesmente utilizando o ranqueamento $\mathbf{R}(I_p, \delta_k, \mathcal{P})$. Já em caso do número de soluções não-dominadas exceder o tamanho do arquivo, ao invés de se utilizar o ranqueamento, emprega-se um critério baseado num índice que mede a densidade das soluções do arquivo apenas em relação às outras soluções também pertencentes ao arquivo, não mais considerando toda a população. O procedimento, chamado de *truncagem*, é descrito a seguir:

- O processo se inicia com um conjunto $\mathcal{C}$ de soluções não-dominadas, de cardinalidade $c > q$, do qual devem ser excluídas $c - q$ soluções, de forma a constituir o conjunto $\mathcal{Q}$, que deverá ter cardinalidade q.
- Determinam-se as distâncias dois-a-dois, no espaço de objetivos, entre todos os elementos de $\mathcal{C}$.
- Tomam-se os elementos aos quais se associa a menor distância. Esses serão pelo menos dois elementos, distantes um do outro dessa mínima distância. Desses elementos, escolhe-se aquele que apresentar a menor distância em relação a um outro elemento, para ser excluído do conjunto $\mathcal{C}$. Caso persista o empate entre elementos (ou seja, mais de um elemento cuja segunda menor distância a outro ponto do conjunto é igual) considera-se a terceira menor distância como critério de desempate, e assim por diante.
- O procedimento se repete, até que restem apenas q soluções em $\mathcal{C}$, as quais são então armazenadas em $\mathcal{P}$.

Esse procedimento é ilustrado na figura 17.9. A notação

$$\mathcal{P} \leftarrow \text{truncagem}(\mathcal{C}, q)$$

será utilizada para designar a aplicação dessa operação de truncagem sobre o conjunto $\mathcal{C}$ de maneira a que a cardinalidade do conjunto $\mathcal{P}$ resultante seja igual a q.

Estrutura de AE's Multiobjetivo

Para exemplificar a construção de algoritmos evolutivos multiobjetivo, são agora apresentadas as estruturas do NSGA-II e do SPEA2. O pseudo-código do algoritmo NSGA-II encontra-se representado no Algoritmo 1, e o pseudo-código do algoritmo SPEA2 encontra-se descrito no Algoritmo 2. As seguintes definições serão empregadas nos algoritmos:

$\mathcal{B} \leftarrow$ **variação**$(\mathcal{A}, n)$
irá representar a aplicação de operadores de mutação e de cruzamento aos indivíduos do conjunto $\mathcal{A}$, resultando em n novos indivíduos que são armazenados no conjunto $\mathcal{B}$.

$\mathcal{B} \leftarrow$ **aleatório**(n)
irá representar a geração aleatória de n indivíduos, que serão armazenados no conjunto $\mathcal{B}$.

$\mathcal{B} \leftarrow$ **não-dominado**$(\mathcal{A})$
irá indicar a operação de identificar os elementos não-dominados pertencentes ao conjunto $\mathcal{A}$, atribuindo o resultado ao conjunto $\mathcal{B}$.

$\mathcal{B} \leftarrow$ **torneio**$(\mathcal{A}, \mathbf{R}, I_*, \delta_*, n)$
indicará a realização de um torneio binário com reposição[5], realizado com os elementos do conjunto $\mathcal{A}$, utilizando como função de aptidão o ranqueamento $\mathbf{R}(I_*, \delta_*, \mathcal{A})$, resultando na escolha de n indivíduos que são armazenados no conjunto $\mathcal{B}$.

Algoritmo 1 Pseudocódigo do NSGA-II

1: $\mathcal{X} \leftarrow$ aleatório(N)
2: **enquanto** critério de parada não satisfeito **faça**
3: $\quad\bar{\mathcal{X}} \leftarrow$ variação$(\mathcal{X}, N)$
4: $\quad\mathcal{P} \leftarrow \bar{\mathcal{X}} \cup \mathcal{X}$
5: $\quad\mathcal{X} \leftarrow$ seleciona$(I_f, \delta_a, \mathbf{R}, \mathcal{P}, N)$
6: **fim enquanto**

Algoritmo 2 Pseudocódigo do SPEA2

1: $\mathcal{X} \leftarrow$ aleatório(N)
2: $\mathcal{Q} \leftarrow \emptyset$
3: **enquanto** critério de parada não satisfeito **faça**
4: $\quad\mathcal{P} \leftarrow \mathcal{X} \cup \mathcal{Q}$
5: $\quad\mathcal{Q} \leftarrow$ não-dominado$(\mathcal{P})$
6: $\quad$**se** $|\mathcal{Q}| > q$ **então**
7: $\quad\quad\mathcal{Q} \leftarrow$ truncagem$(\mathcal{Q}, q)$
8: $\quad$**senão se** $|\mathcal{Q}| < q$ **então**
9: $\quad\quad\mathcal{Q} \leftarrow$ seleciona$(I_p, \delta_k, \mathbf{R}, \mathcal{Q}, q)$
10: $\quad$**fim se**
11: $\quad\mathcal{X} \leftarrow$ torneio$(\mathcal{Q}, \mathbf{R}, I_p, \delta_k, N)$
12: $\quad\mathcal{X} \leftarrow$ variação$(\mathcal{X}, N)$
13: **fim enquanto**

Além disso, a notação $|\mathcal{A}|$ indica a cardinalidade do conjunto $\mathcal{A}$.

É importante mencionar que o algoritmo NSGA-II e o algoritmo SPEA2 incluem em sua especificação, além das macro-operações a serem realizadas sobre os conjuntos descritas nesses pseudo-códigos, também a especificação de estruturas de dados eficientes, bem como de algoritmos detalhados particularmente eficazes para a realização dessas macro-operações. Isso quer dizer que, embora todo algoritmo que implemente os pseudo-códigos descritos no Algoritmo 1 e no Algoritmo 2 vá certamente produzir sequências de populações e resultados compatíveis com o NSGA-II e o SPEA2, não necessariamente isso será feito com a mesma eficiência computacional dos algoritmos originais.

3. Desenvolvimentos Recentes e Futuros

A computação evolutiva multiobjetivo foi, ao longo da década dos 2000, um tema no qual foi concentrado intenso esforço de pesquisa. Uma grande variedade de detalhes a respeito do funcionamento dos algoritmos dessa classe vêm sendo investigada, e encontra-se descrita em extensa literatura, constituída de quase seis mil artigos, publicados nos últimos dez anos, que se encontram hoje indexados[6]. Nesta seção, ao invés de procurar apresentar uma lista abrangente de temas que vêm sendo abordados, faremos uma discussão de apenas quatro assuntos, que se destacam por concentrarem o trabalho de muitos grupos de pesquisadores, em vários locais diferentes, e que parecem, neste momento, ter potencial para causar mudanças nas estruturas consideradas padrão na área de algoritmos evolutivos multiobjetivo: a hibridização com *mecanismos de busca local*, o desenvolvimento de algo-

[5] Em um torneio binário com reposição, dois indivíduos são escolhidos a cada vez, aleatoriamente. Esses dois indivíduos são comparados, utilizando uma função de aptidão, sendo o melhor dos dois escolhido. Os dois indivíduos são retornados ao conjunto original, e o processo se repete, até que n escolhas tenham sido feitas.

[6] Consulta realizada na base Scopus, em 14/06/2011, indicou 5936 artigos em 10 anos.

ritmos baseados no *indicador de hipervolume*, a introdução de ferramentas para *indicação progressiva de preferências*, e a questão teórica do tratamento de *muitos objetivos*.

Mecanismos de Busca Local

Sabe-se hoje que, em geral, os algoritmos evolutivos irão requerer algum mecanismo de busca local, em adição aos operadores evolutivos básicos, para realizar eficientemente as tarefas de determinação de mínimos de funções. Isto ocorre porque os operadores evolutivos tradicionais são construídos a partir de movimentos aleatorizados, que geram novos pontos que serão aproveitados se forem melhores que os anteriores. Esse tipo de movimentação é importante para a realização de buscas globais, principalmente no caso de funções multimodais. No entanto, mais cedo ou mais tarde, o processo de busca termina por chegar em pontos a partir dos quais seria possível atingir o ponto de ótimo por meio da escolha da direção do movimento como aquela em que a função decresce mais rapidamente. Quando o processo de otimização atinge regiões em que essa busca é possível, fazer a busca utilizando a informação local sobre o decrescimento da função torna-se muito mais eficiente que a busca aleatorizada típica dos operadores evolutivos. Isso vale tanto para algoritmos para problemas de variáveis discretas quanto para problemas de variáveis contínuas. Essa hibridização de algoritmos evolutivos com técnicas de busca determinística local poderia ser realizada em dois estágios: primeiro se faz a busca aleatorizada e, a partir de um certo ponto, inicia-se uma busca por meio de algoritmos determinísticos. No entanto, há uma tendência hoje para a construção de algoritmos que empregam operadores de busca local como um operador constituinte do algoritmo evolutivo, que é executado ao longo das iterações do AE. Esse tipo de esquema tem sido denominado *Algoritmo Memético* (Moscato, 1999; Hu et al., 2003).

O primeiro algoritmo memético multiobjetivo foi apresentado por (Ishibuchi e Murata, 1998). Outro algoritmo memético importante foi proposto por (Knowles e Corne, 2000b). O algoritmo denominado *Memetic Pareto Archive Evolutionary Strategy* (M-PAES) associa um esquema evolutivo de busca global, baseada em operadores de mutação e de recombinação, com o algoritmo PAES (Knowles e Corne, 1999), que realiza buscas locais. Uma revisão sobre algoritmos meméticos multiobjetivo é apresentada em (Knowles e Corne, 2004).

Neste momento, podem ser identificados três tipos de estratégias para a realização de buscas locais em algoritmos evolutivos multiobjetivo:

- Durante a execução do algoritmo, são escolhidos alguns pontos do arquivo corrente, e são aplicadas operações de busca local sobre esses pontos, na tentativa de gerar novos pontos que dominem os anteriores, ou ainda pontos sem relação de dominância com os anteriores que sirvam para melhorar a amostragem da estimativa do conjunto Pareto-ótimo. (Lara et al., 2010) apresentam um tipo de operador que procura executar essas operações de maneira automática. (Sindhya et al., 2011), por outro lado, apresentam estratégias baseadas em algoritmos clássicos de programação não-linear para realizar essa tarefa.

- Durante a execução do algoritmo evolutivo, vão sendo construídos modelos das funções-objetivo do problema multiobjetivo. Esses modelos são então empregados para a geração de estimativas melhoradas de soluções Pareto-ótimas. (Wanner et al., 2008) mostram uma estratégia baseada em modelos de aproximação quadrática para realizar essa operação.

- Outra alternativa é a construção de modelos do próprio conjunto Pareto-ótimo, como forma de articular a informação a ser utilizada na busca local. Os autores (Gaspar-Cunha e Vieira, 2004) e (Loshchilov et al., 2010) mostram alternativas de realização desta estratégia.

Indicador de Hipervolume

Neste capítulo foi visto que a tarefa de avaliar a qualidade das soluções de algoritmos de otimização multiobjetivo é bem mais complexa que no caso mono-objetivo. No caso multiobjetivo, essa tarefa diz

respeito a avaliar a qualidade de um conjunto de pontos, que deve constituir uma amostragem representativa do conjunto Pareto-ótimo. Não basta avaliar isoladamente o quão cada ponto se aproxima do conjunto Pareto-ótimo exato, mas também de julgar o quão bem representado está cada ponto desse conjunto exato, naquele conjunto-solução que procura representá-lo. Uma discussão abrangente sobre essa questão de avaliar os conjuntos-solução de algoritmos de otimização multiobjetivo foi apresentada por (Zitzler et al., 2003).

Uma maneira engenhosa de executar essa tarefa foi proposta por (Zitzler, 1999), baseada na medição do tamanho da região dominada pelo conjunto-solução a ser avaliado. O indicador que resulta dessa medição, chamado de *indicador de hipervolume*[7], é ilustrado na figura 17.10. O cálculo

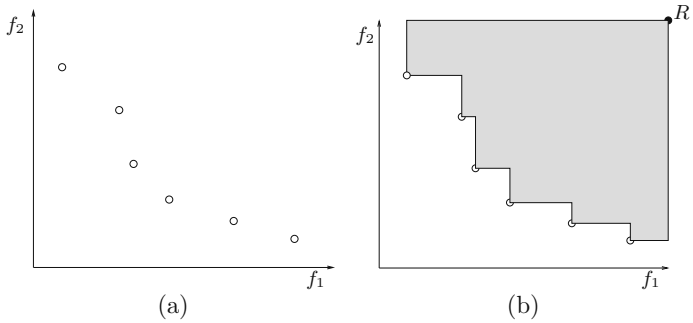


Figura 17.10: Indicador de hipervolume. (a) Conjunto de pontos a ser avaliado, no espaço de objetivos. (b) O valor do indicador de hipervolume corresponde ao volume da região constituída pelos pontos que simultaneamente dominam um ponto de referência R e são dominados por pelo menos um ponto do conjunto a ser avaliado.

desse indicador é feito da seguinte forma:

- Toma-se o conjunto de pontos a ser avaliado, no espaço de objetivos.
- Acrescenta-se um ponto de referência R, dominado por todos os pontos do conjunto a ser avaliado.
- Calcula-se o volume da região constituída pelos pontos que simultaneamente dominam R e são dominados por pelo menos um ponto do conjunto a ser avaliado.

No caso de duas dimensões (dois objetivos), esse volume se reduz à área da união dos retângulos, com lados paralelos aos eixos coordenados, que têm um vértice em R e o vértice oposto em um ponto do conjunto.

Uma propriedade desejável para um indicador que se proponha a avaliar conjuntos de soluções fornecidas por algoritmos de otimização multiobjetivo seria a chamada *Pareto-conformidade*[8]. Sejam A e B dois conjuntos sendo avaliados, e $I(A)$ e $I(B)$ os valores de um indicador I calculado sobre esses conjuntos. A propriedade de Pareto-conformidade é expressa como (Zitzler et al., 2007):

$$A \preceq B \text{ e } B \not\preceq A \Rightarrow I(A) > I(B) \tag{17.6}$$

A relação $A \preceq B$ significa que para todo ponto b_i pertencente a B, existe pelo menos um ponto a_j pertencente a A tal que $a_j \preceq b_i$. Já a relação $B \not\preceq A$ significa que existe pelo menos um ponto a_j pertencente a A tal que nenhum ponto b_i pertencente a B satisfaz a relação $b_i \preceq a_j$. Caso A e B sejam

[7] Na literatura, esse indicador também é chamado de *métrica S*, (ou *S-metric*, em inglês), seguindo a nomenclatura originalmente empregada na referência em que o indicador foi proposto.

[8] Do inglês: *Pareto-compliance*.

relacionados dessa forma, então um indicador $I(\cdot)$ Pareto-conforme deverá retornar valores escalares tais que $I(A) > I(B)$.

O indicador de hipervolume é a única métrica para avaliar conjuntos de estimativas do conjunto Pareto-ótimo hoje conhecida que possui a propriedade de Pareto-conformidade (Zitzler et al., 2007). Por esse motivo, ao longo dos últimos 10 anos, o indicador de hipervolume veio se tornando uma métrica relativamente consensual, quando se trata de avaliar o desempenho de algoritmos evolutivos de otimização multiobjetivo.

Dois tipos de questões relacionadas com o indicador de hipervolume têm merecido significativos esforços de pesquisa nos anos recentes. O primeiro diz respeito à dificuldade computacional para o cálculo desse indicador. Embora no caso de apenas duas dimensões esse cálculo seja bastante trivial, quando o número de dimensões cresce o problema se torna muito mais complexo. Têm sido feitos esforços tanto para a construção de algoritmos rápidos para o cálculo do indicador em dimensões elevadas (Beume et al., 2009; Emmerich e Fonseca, 2011; While et al., 2011) quanto para a obtenção de versões estocásticas ou heurísticas desse cálculo que aproximem o valor do indicador, sem calculá-lo de forma exata (Bringmann e Friedrich, 2009; Bader, 2009; Ishibuchi et al., 2010). O segundo tipo de questão que vem sendo estudado diz respeito à construção de algoritmos evolutivos que utilizem diretamente o indicador de hipervolume, como critério empregado no mecanismo de seleção (Zitzler e Kunzli, 2004; Emmerich et al., 2005; Bader, 2009). Diferentes estratégias podem ser empregadas para que, ao se substituírem algumas soluções de um conjunto por outras, o conjunto resultante tenha um indicador de hipervolume com um valor maior que o inicial. A fórmula mais simples seria a exclusão dos pontos com menor contribuição individual para o indicador, e a inclusão dos novos pontos que individualmente causem o maior incremento do mesmo. Mecanismos mais sofisticados, no entanto, procuram verificar a contribuição conjunta dos pontos, que é diferente da soma das contribuições individuais, para a escolha dos pontos a serem introduzidos ou excluídos.

Indicação de Preferências Progressiva

Uma questão que se manifesta em parte significativa dos problemas de otimização multiobjetivo é a de que o custo computacional para a geração de um conjunto de soluções que represente a fronteira Pareto-ótima pode ser muito elevado, seja pelo fato de cada avaliação das funções objetivo ser muito elevado, seja pela necessidade de que seja gerado um número muito grande de soluções para que se constitua um conjunto representativo do conjunto Pareto-ótimo do problema.

Nessas situações, torna-se conveniente que a exploração do espaço de soluções, ao invés de conduzir a soluções que cubram todo o conjunto Pareto-ótimo, promova a geração de soluções que representem apenas um sub-conjunto do conjunto Pareto-ótimo, que inclua a solução que será finalmente a preferida pelo tomador de decisão, e mais algumas soluções na vizinhança desta. Esse procedimento, claramente, simplifica o problema computacional em questão, permitindo a geração de um número muito menor de soluções Pareto-ótimas, e requerendo que as soluções geradas sejam representativas desse conjunto em um sentido apenas local.

Para que isto se torne possível, é necessário que o algoritmo disponha da informação a respeito da estrutura de preferências do tomador de decisão (decisor). Frequentemente, essa estrutura de preferências não se encontra disponível a priori, antes que se inicie o processo de otimização. Nesse caso, para que a busca possa ser dirigida para a região onde se encontra a solução preferida pelo tomador de decisão, será necessário que o algoritmo de otimização promova alguma interação com esse tomador de decisão ao longo do processo de otimização. Isto é particularmente conveniente, muitas vezes, porque a informação produzida pela execução do algoritmo de otimização, com a avaliação de um certo número de soluções em regiões diferentes do espaço de variáveis de decisão, pode ser empregada como informação capaz de subsidiar o tomador de decisão. Ao aprender sobre a estrutura do problema, o tomador de decisão forma sua opinião, e pode assim indicar suas preferências. Isso

ocorre progressivamente porque ao longo do processo de otimização ocorrem várias interações com o tomador de decisão, à medida em que a população do algoritmo evolutivo se aproxima de uma determinada região do conjunto Pareto-ótimo, sendo guiada pelas preferências indicadas por este tomador de decisão.

Um expressivo número de trabalhos foi publicado, de poucos anos para cá, a respeito de esquemas de indicação progressiva de preferências em algoritmos evolutivos. A título de exemplo, citamos aqui os trabalhos de (Battiti e Passerini, 2010), de (Deb et al., 2010), de (Karahan e Koksalan, 2010) e de (Kim et al., 2011).

Problemas com Muitos Objetivos

No início dos anos 2000, depois que os métodos evolutivos de otimização multiobjetivo passaram a ser utilizados em um número crescente de aplicações, uma dificuldade inicialmente não identificada passou a preocupar os pesquisadores da área: a perda de eficiência dos métodos evolutivos à medida em que crescia o número de objetivos a serem otimizados simultaneamente. Os problemas com tal dificuldade passaram a ser denominados *problemas com muitos objetivos*[9].

A natureza geométrica dos problemas com muitos objetivos encontra-se ilustrada na figura 17.11. Nessa figura, a origem do sistema de coordenadas do espaço de objetivos encontra-se deslocada para

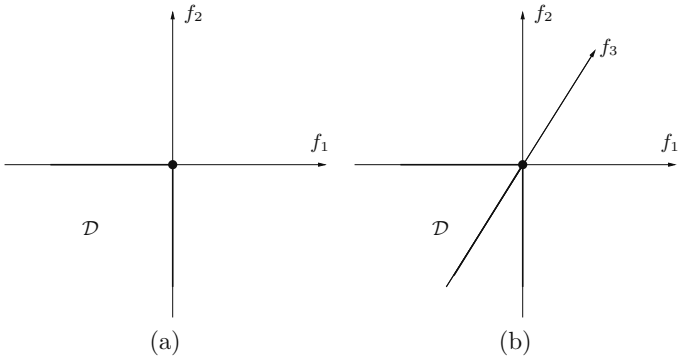


Figura 17.11: Geometria do problema com muitos objetivos. (a) Problema com dois objetivos. (b) Problema com três objetivos.

sobre um ponto-solução corrente, que irá sofrer uma mutação, tanto (a) para um caso com dois objetivos, como (b) para um caso com três objetivos. Em ambos os casos, a região indicada como $\mathcal{D}$ corresponde à região dos pontos que dominam o ponto corrente. Assim, se a mutação levar a um novo ponto em $\mathcal{D}$, ela será bem sucedida em melhorar o ponto atual. Do contrário, não conduzirá a melhorias na solução atual. O aspecto relevante nesta análise é que, no caso de dois objetivos, a região $\mathcal{D}$ corresponde a um quadrante do espaço (o quadrante de coordenadas negativas), ou seja, a uma fração de $\frac{1}{4}$ das direções no espaço de objetivos. Já no caso de três objetivos, $\mathcal{D}$ corresponde a um octante do espaço (o octante de coordenadas negativas), ou a uma fração de $\frac{1}{8}$ das direções no espaço de objetivos. Para um número de objetivos m, a fração correspondente a $\mathcal{D}$ será de $\frac{1}{2^m}$.

Embora o mapeamento entre o espaço de variáveis de decisão e o espaço de objetivos seja não-linear, o que significa que essas proporções não serão as mesmas no espaço de variáveis de decisão (no qual efetivamente será processada a mutação da solução corrente), esse efeito de rápida redução do volume da região na qual ocorre melhoria da solução, comparado ao volume da região em que não ocorre melhoria, se transpõe também para esse espaço. Como consequência, a probabilidade de uma mutação efetivada em uma direção inteiramente aleatória levar a uma melhoria de solução pode

[9] Do inglês: *many-objective problems*.

ser muito pequena, para problemas com um grande número de objetivos, ficando cada vez menor à medida em que aumenta a dimensão do espaço de objetivos. Isso explica a perda da eficiência dos algoritmos evolutivos quando o número de objetivos cresce.

Vários trabalhos publicados nos últimos anos têm procurado compreender, de maneira mais precisa, a natureza das dificuldades relacionadas com os problemas com muitos objetivos. Tem sido tentada a proposição de técnicas diferentes de seleção, de atribuição de aptidão, ou ainda de busca local, ou ainda técnicas de redução da dimensão do espaço de objetivos, para tratar esse tipo de problema. Citamos aqui as referências (Purshouse e Fleming, 2007; Adra e Fleming, 2011; Bader e Zitzler, 2011; Lopez-Jaimes et al., 2011; Pasia et al., 2011; Purshouse et al., 2011; Schutze et al., 2011), que em conjunto apresentam um panorama da questão.

CAPÍTULO 18

Algoritmos Genéticos em Problemas de Classificação

Márcio P. Basgalupp [*] *André Rossi* [**]

Ana C. Lorena [**] *André C. P. L. F. de Carvalho* [**]

[*] *Instituto de Ciência e Tecnologia*
Universidade Federal de São Paulo

[**] *Instituto de Ciências Matemáticas e de Computação*
Universidade de São Paulo - São Carlos

Muitas técnicas de Aprendizado de Máquina (AM) utilizam conhecimentos da Inteligência Artificial e da Estatística para construção de modelos capazes de adquirir conhecimento a partir de um conjunto de dados. Os exemplos desse conjunto são chamados de dados de treinamento e a partir desses faz-se a inferência indutiva, que pode gerar hipóteses verdadeiras ou não (Alpaydin, 2004; Monard e Baranauskas, 2003). Todo algoritmo de aprendizado, ou algoritmo de indução, ou simplesmente indutor, possui um viés (*bias*) indutivo, que é a preferência de uma hipótese sobre outra, que não são igualmente prováveis. Os exemplos de um conjunto de dados são formados por atributos e cada atributo especifica uma característica particular para esse conjunto. O aprendizado indutivo pode ser dividido em aprendizado supervisionado e não-supervisionado. Atualmente o aprendizado semi-supervisionado também tem atraído considerável atenção da comunidade de AM (Chapelle et al., 2006).

No aprendizado supervisionado, cada exemplo apresentado ao algoritmo de aprendizado possui um atributo especial que especifica o rótulo da classe real à qual este exemplo pertence. Se os rótulos das classes forem discretos, o problema é conhecido como classificação; se forem contínuos, como regressão ou aproximação de funções.

No aprendizado não-supervisionado ou agrupamento, o algoritmo de aprendizado não tem conhecimento dos rótulos das classes reais. Dessa maneira, o algoritmo agrupa os exemplos por meio de extração de padrões nos valores de seus atributos.

No aprendizado semi-supervisionado, exemplos em que os rótulos das classes são conhecidos e também exemplos em que os rótulos das classes não são conhecidos são apresentados ao algoritmo de aprendizado. O algoritmo utiliza, assim, tanto os exemplos rotulados como os não rotulados durante o aprendizado.

Neste capítulo serão discutidas algumas alternativas para o uso de computação evolutiva, especificamente algoritmos genéticos, no projeto e combinação de modelos de classificação. Segundo (Ye, 2003), o objetivo da classificação é construir um modelo conciso de distribuição do atributo classe (ou alvo) em função demais atributos, denominados atributos preditivos. O resultado desse modelo pode ser utilizado para atribuir valores a exemplos onde somente seus atributos preditivos são conhecidos.

Em um problema de classificação, os dados de entrada podem ser descritos pelo par ordenado (X, y), em que X é um vetor que representa os atributos preditivos, $X = (x_1, x_2, x_3, ..., x_n)$, e y é o rótulo da classe à qual esse exemplo pertence. Na Tabela 18.1 tem-se um conjunto de dados para classificação do estado de saúde de um paciente. Nessa tabela, cada linha representa um exemplo do conjunto de dados e cada coluna um atributo desse exemplo. O atributo *Diagnóstico* é especial, pois possui o rótulo da classe para cada exemplo, ou seja, doente ou saudável.

Tabela 18.1: Conjunto de dados para o diagnóstico da saúde de pacientes.

Exemplo	**Febre**	**Enjôo**	**Manchas**	**Dor**	**Diagnóstico**
T1	sim	sim	pequenas	sim	doente
T2	não	não	grandes	não	saudável
T3	sim	sim	pequenas	não	saudável
T4	sim	não	grandes	sim	doente
T5	sim	não	pequenas	sim	saudável
T6	não	não	grandes	sim	doente

Técnicas de AM têm sido utilizadas em problemas de classificação e cada uma utiliza um algoritmo de aprendizado para construir um modelo (classificador) que relaciona os atributos e os rótulos das classes. Um ponto importante dos algoritmos de aprendizado é construir modelos que possuam boa capacidade de generalização, ou seja, consigam predizer, com alta taxa de acerto, rótulos das classes para exemplos que não foram apresentados anteriormente (Tan et al., 2005).

Outra característica dos classificadores a ser observada diz respeito à interpretabilidade do conhecimento adquirido. Os classificadores do tipo caixa-preta são aqueles que possuem uma representação interna que, geralmente, não se consegue interpretar facilmente, ou seja, é difícil conhecer as características do problema que o levaram a uma determinada dedução. Um exemplo de classificador desse tipo são as Redes Neurais Artificiais (RNs). Os classificadores orientados a conhecimento, como as Árvores de Decisão (ADs) e sistemas *fuzzy*, criam estruturas simbólicas que normalmente são mais compreensíveis do que os classificadores do tipo caixa-preta (Monard e Baranauskas, 2003).

Na Figura 18.1 é ilustrado um diagrama do processo de indução de um classificador e posteriormente a sua utilização. Primeiro o conjunto de treinamento, no qual os rótulos das classes dos exemplos são conhecidos, é utilizado por um algoritmo de aprendizado para construir um modelo. Após a construção, esse classificador pode ser aplicado para predizer os rótulos das classes para exem-

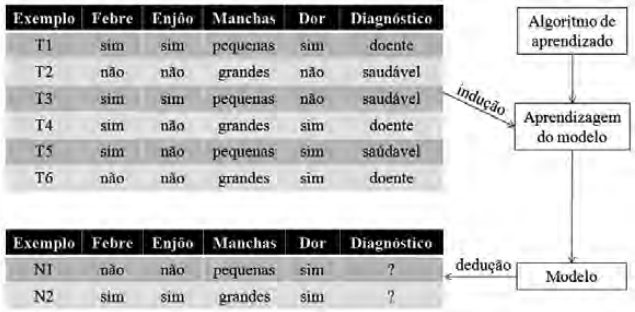


Figura 18.1: Diagrama do processo de indução de um classificador e sua utilização na dedução de novos exemplos.

plos do conjunto de teste.

No contexto de problemas de classificação, os algoritmos genéticos têm tido um papel importante em diversas aplicações. Dentre elas, destacam-se: (i) ajuste de parâmetros dos algoritmos de indução de classificadores; (ii) indução de árvores de decisão; e (iii) decomposição de problemas multiclasses.

O restante deste capítulo está organizado como segue. A Seção 1 discute como os algoritmos genéticos podem ser utilizados para ajustar parâmetros de algoritmos de indução de classificadores. Ao descrever o algoritmo LEGAL-Tree, a Seção 2 mostra como os algoritmos genéticos são facilmente aplicados na indução de modelos de classificação baseados em árvores de decisão. A solução para decomposição de problemas multiclasses por meio de algoritmos genéticos é descrita na Seção 3. Por fim, a Seção 4 apresenta as considerações finais.

1. Ajuste de Parâmetros de Classificadores por AGs

Grande parte dos algoritmos de AM possuem parâmetros cujos valores devem ser especificados pelo usuário. Esses valores para os parâmetros livres, em geral, influenciam diretamente no desempenho de modelos induzidos (Kohavi e John, 1995), o que pode ser entendido como uma deficiência das técnicas de AM. O objetivo do processo de ajuste desses parâmetros pode ser visto como encontrar os melhores valores dos parâmetros livres para um determinado conjunto de dados (Hutter e Hamadi, 2005). O ajuste de parâmetros, cujo intuito é obter melhor desempenho e maior robustez dessas técnicas, é uma tarefa subjetiva e pode consumir muito tempo.

Para definir um conjunto de valores, as atuais técnicas de ajuste normalmente consideram a interação entre o viés (*bias*) do algoritmo de indução (Mitchell, 1982) e o conjunto de treinamento disponível. Técnicas comuns, como a busca exaustiva, são intratáveis quando há mais de dois parâmetros a serem ajustados (Chapelle et al., 2002). Heurísticas podem ser aplicadas com relativo sucesso para uma variedade de conjuntos de dados. Porém, a melhor estratégia é obter valores dos parâmetros que funcionem bem para um conjunto de dados particular (Kohavi e John, 1995). A abordagem mais comum para definir os valores dos parâmetros livres é por tentativa e erro, a qual é altamente subjetiva. Além disso, a busca pelos melhores valores para os parâmetros, geralmente, envolve a otimização por um grande espaço de busca, o que torna esse problema muito custoso computacionalmente.

Por essa razão, técnicas alternativas de otimização têm sido propostas para ajustar de forma automática os parâmetros livres de algoritmos de AM. Algumas dessas técnicas, como os AGs, têm obtido resultados promissores.

Ajuste de Parâmetros de Máquinas de Vetores de Suporte

As Máquinas de Vetores de Suporte (SVMs, do inglês, Support Vector Machines) são baseadas na Teoria do Aprendizado Estatístico (Vapnik e Chervonenkis, 1971), mais especificamente, na Teoria de Minimização do Risco Estrutural (Vapnik, 1995). Os vetores de suporte utilizados pelas SVMs são exemplos que estão próximos da superfície de decisão e, portanto, são os mais difíceis de serem classificados (Haykin, 1999). São esses exemplos que influenciam diretamente na localização da superfície de decisão. Para o problema de classificação, o princípio das SVMs está em encontrar um hiperplano ótimo que separa satisfatoriamente os dados de entrada. O hiperplano ótimo é definido como aquele para o qual a margem de separação entre as classes é maximizada (Haykin, 1999).

O desempenho das SVMs é diretamente influenciado pelo parâmetro *custo* e pela escolha da função *kernel* e dos valores de seus parâmetros (Chapelle et al., 2002). O parâmetro *custo* controla o equilíbrio entre a complexidade do modelo e o número de exemplos não-separáveis (Haykin, 1999). Essa dependência dos valores dos parâmetros não é uma exclusividade das SVMs, e muitos algoritmos de aprendizado dependem de uma escolha adequada para encontrar um bom modelo. O conhecimento sobre o domínio pode auxiliar na escolha da função *kernel* apropriada, reduzindo o problema de seleção de modelo para o ajuste de parâmetros (Cristianini e Shawe-Taylor, 2000).

Para o ajuste de parâmetros de SVMs, métodos de otimização baseados no gradiente são muito utilizados. Porém, segundo (Imbault e Lebart, 2004), esses métodos não resolvem totalmente o problema, a menos que um ponto inicial seja conhecido. Nesse trabalho foi mostrado que o problema de ajuste de parâmetros apresenta mínimo local e foram comparados métodos clássicos de ajuste que utilizam busca local com AGs e *recozimento simulado*, que são métodos de otimização global. Um fator importante dos métodos de otimização global é que estes são desenvolvidos para evitar mínimos locais. Os resultados mostraram que os dois métodos citados obtiveram soluções próximas da ótima de forma mais robusta e eficiente.

Algoritmos Genéticos também foram utilizados em (Lorena e Carvalho, 2006) para ajustar os parâmetros de SVMs multiclasses com *kernel* gaussiano. Nesse estudo foram obtidas maiores taxas de acerto utilizando AGs se comparado aos resultados obtidos utilizando valores fixos para os parâmetros. Esse resultado ocorreu para as quatro bases de dados testadas. Em (Huang e Wang, 2006) e (Souza e Carvalho, 2005) foram utilizados AGs para selecionar características (atributos) de conjuntos de dados e ajustar parâmetros de SVMs simultaneamente. No primeiro caso, foram realizados testes com vários conjuntos de dados e a abordagem baseada em AGs teve boa taxa de acerto se comparada com a técnica *grid search*. No segundo caso, foi utilizado um conjunto de dados de expressão gênica e os resultados obtidos foram equivalentes a outros encontrados na literatura.

Em (Souza et al., 2006), a técnica de otimização por enxame de partículas (PSO) (Kennedy e Eberhart, 1995) foi utilizada para ajustar parâmetros de SVMs multiclasses com *kernel* gaussiano. Foram testadas quatro bases de dados e os resultados obtidos foram comparados com os de outras técnicas de ajuste. Essas técnicas foram denominadas *grid search*, que usa a busca exaustiva, *Naive*, que usa os mesmos valores padrão da biblioteca LIBSVM (Chang e Lin, 2001) para todas as SVMs binárias, e *Global*, que usa um conjunto de validação para estimar o erro de generalização. O PSO conseguiu os menores erros de classificação apenas para uma base, enquanto a técnica *grid search* obteve os melhores resultados para duas bases e a técnica Global para a base restante. Apesar disso, os resultados obtidos pela PSO ficaram próximos aos melhores alcançados e, segundo os autores, não foi possível determinar o melhor método para todas as bases de dados testadas.

Quatro algoritmos bioinspirados, dentre eles um AG, foram usados em (Rossi e Carvalho, 2008) para ajustar os parâmetros de SVMs com *kernel* gaussiano. Os modelos obtidos foram aplicados para classificar quatro bases de dados de análise de expressão gênica e foram comparados com a técnica *grid search* e com os valores padrão da biblioteca LIBSVM. Os resultados obtidos entre todos os algoritmos e técnicas foram semelhantes. Para uma das bases de dados o AG não conseguiu bons resultados, pois ficou preso a mínimos locais. Os valores padrão mostraram-se robustos, conseguindo

melhores resultados para duas bases testadas.

Ajuste de Parâmetros de Redes Neurais

As Redes Neurais Artificiais (RNs) são inspiradas no cérebro e no sistema nervoso e são compostas por unidades de processamento simples, denominados neurônios ou nós, dispostos em uma ou mais camadas e interligados por meio de conexões. Os pesos associados a cada conexão entre os neurônios (sinapse) são responsáveis pelo armazenamento do conhecimento. Os neurônios artificiais são unidades de processamento de informação que realizam um trabalho relativamente simples: recebem entradas de outros neurônios ou do meio externo e usam essas entradas para computar um sinal de saída que é propagado para outras unidades ou para o meio externo. Um algoritmo de aprendizado de RNs deve ser capaz de atribuir pesos a essas conexões durante o processo de treinamento, de maneira que a rede neural seja capaz de classificar corretamente o maior número de exemplos possíveis do conjunto de treinamento e, também, seja capaz de generalizar para novos exemplos.

Há uma grande variedade de arquiteturas e algoritmos de treinamento para RNs e, normalmente, um grande número de parâmetros precisam ser especificados pelo usuário. Em (Basheer e Hajmeer, 2000) os autores afirmam que bons valores para os parâmetros são encontrados, geralmente, por meio de tentativa e erro. Segundo os autores, a escolha de valores para os parâmetros do algoritmo de aprendizado *backpropagation* para RNs influencia na convergência do aprendizado e no desempenho geral da rede.

Algoritmos evolutivos são empregados para ajuste de parâmetros e treinamento de RNs desde o início da década de 90, como pode ser visto em (Miller et al., 1989; Dodd, 1990; Hintz e Spofford, 1990; Braun e Weisbrod, 1993), e ainda são utilizados nos dias atuais. Os trabalhos de (Leung et al., 2003) e (Tsai et al., 2006) propõem modificações nos AGs para a otimização automática e simultânea da topologia (estrutura) das RNs e dos parâmetros do algoritmo de treinamento. Os resultados obtidos pelo AG proposto em (Leung et al., 2003) superaram os obtidos pelo AG padrão, que utilizou cruzamento aritmético e mutação não-uniforme, na otimização de funções de *benchmark*. Posteriormente, duas aplicações foram testadas com as RNs para ilustrar o desempenho dos algoritmos propostos. Em ambas, o AG proposto superou o AG padrão, conseguindo uma rede de menor complexidade (menor número de conexões) e com maior acurácia. Em (Tsai et al., 2006) foi proposto um algoritmo híbrido, que combina AG com o método *Taguchi* (Phadke, 1995), o qual denominaram de HTGA (do inglês, *hybrid Taguchi-Genetic Algorithm*), para ajustar a estrutura e os parâmetros de RNs. Os autores utilizaram as mesmas aplicações testadas em (Leung et al., 2003) e compararam os resultados obtidos. O HTGA foi superior em relação à acurácia, enquanto que a complexidade da rede variou conforme a aplicação.

Um estudo comparativo entre sistemas evolutivos híbridos para geração e otimização da estrutura de RNs de múltiplas camadas foi realizado em (Castillo et al., 2007). Esse estudo usou um método chamado *G-Prop* para otimizar o número de neurônios ocultos e os pesos iniciais das RNs e outro método chamado *ev-QP* para buscar pelos parâmetros de um algoritmo de treinamento. Utilizou-se também um algoritmo co-evolutivo, chamado *co-evolutive*, para tratar dos parâmetros referente à arquitetura, aos pesos iniciais e aos parâmetros do algoritmo de treinamento. Os resultados mostraram que os algoritmos evolutivos apresentaram melhor habilidade de classificação do que o QuickProp (Fahlman, 1988). O primeiro método (GProp) obteve pequenas taxas de erro de classificação, porém o tempo de processamento foi um pouco superior se comparado a outros métodos. O segundo método (ev-QP) teve o menor tempo de processamento, mas produziu as maiores taxas de erro. O método co-evolutivo obteve os menores tempo de processamento e, ao mesmo tempo, melhorou a habilidade de classificação.

Um outro estudo foi realizado em (Rossi et al., 2008) comparando quatro algoritmos bioinspirados, entre eles um AG, para ajuste de parâmetros de RNs de múltiplas camadas. Para essa comparação foram usados quatro bases de dados de análise de expressão gênica. As melhores redes obtidas pelos

algoritmos usando um conjunto de dados de validação foram aplicados para novos dados. O AG apresentou resultados semelhantes aos outros algoritmos bioinspirados em termos de taxa de erro, apesar do menor espaço de busca analisado analisado pelo AG.

Em (Gao et al., 2006), foi proposto um algoritmo PSO modificado, chamado de *SPSO*, para ajustar simultaneamente a estrutura e os pesos das conexões de RNs. Os resultados obtidos foram comparados com o algoritmo *backpropagation* e com um AG desenvolvido para o mesmo propósito. As redes foram aplicadas para o problema de pontuação de crédito, sendo que o SPSO convergiu mais rapidamente e conseguiu maior acurácia do que o algoritmo *backpropagation*. O AG foi o método que obteve as menores taxas de acurácia.

Um Framework para Ajuste de Parâmetros de classificadores por AGs

Para o problema de ajuste de parâmetros de classificadores, os AGs podem ser empregados usando diferentes representações para os cromossomos, diferentes operadores genéticos (segundo a representação escolhida) e diferentes funções de aptidão. Nesta seção são apresentadas as opções comumente usadas na literatura.

Representação do AG

Grande parte das técnicas de classificação possuem parâmetros cujos valores são contínuos. Portanto, é mais natural representar os genes usando números contínuos ao invés da representação binária. O uso do alfabeto binário para representar valores no espaço contínuo pode produzir o efeito denominado penhasco de Hamming (*Hamming cliff*). Isso acontece quando a codificação binária de dois valores adjacentes difere em todos os bits. Por exemplo, os valores 31 e 32 são representados por 011111 e 100000, respectivamente (Herrera et al., 1998).

Com a utilização da codificação real para problemas de domínio contínuo, não há diferença entre a codificação e o espaço de busca (Herrera et al., 1998), pois cada gene representa uma variável do problema. Dessa forma, o tamanho do cromossomo tem o mesmo tamanho do vetor de soluções para o problema, que neste caso é o número de parâmetros a serem ajustados. Se apenas valores inteiros são permitidos para um parâmetro, a solução obtida pelo AG pode ser arredondada para o valor inteiro mais próximo.

Suponha que se queira ajustar o número de neurônios na camada oculta de uma RN, a taxa de aprendizado do algoritmo *backpropagation* e o termo momentum. Para isso, um cromossomo seria representado por três números reais. O primeiro valor refere-se ao número de neurônios na camada oculta (γ), e pode, por exemplo, assumir um valor dentro do intervalo $[2, 100]$. Como esse parâmetro deve ser inteiro, o valor encontrado pelo AG deve ser arredondado para o valor inteiro mais próximo. O segundo valor, refere-se ao parâmetro taxa de aprendizado (η), e pode assumir qualquer valor real, normalmente dentro do intervalo $[0.05, 1]$. O terceiro valor refere-se ao parâmetro termo *momentum* (μ) e pode assumir qualquer valor real, normalmente dentro do intervalo $[0, 1]$. Na Figura 18.2 é ilustrado um exemplo de um indivíduo com $\gamma = 36$, $\eta = 0.1$ e $\mu = 0.8$.

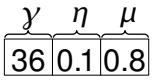


Figura 18.2: Representação de um cromossomo para ajuste de parâmetros de RNs.

Operadores genéticos

Para utilizar um AG com codificação real (AG_{CR}) na solução de problemas, foram desenvolvidos, desde a sua criação, diferentes operadores de cruzamento e mutação. Em (Herrera et al., 1998)

são analisados diferentes operadores para representação contínua. Segundo resultados obtidos neste trabalho, o operador de mutação não-uniforme (Michalewicz, 1992) e os operadores de cruzamento BLX-α (Eshelman e Schaffer, 1993) e *logical* FCB (Herrera et al., 1994) são os mais adequados para serem utilizados com o AG_{CR}. A seguir, o operador de cruzamento BLX-α e o operador de mutação não-uniforme são apresentados. Optou-se aqui por apresentar o BLX-α pela sua simplicidade se comparado ao *logical* FCB.

Sejam $C_1 = (c_1^1, \ldots, c_n^1)$ e $C_2 = (c_1^2, \ldots, c_n^2)$ dois cromossomos selecionados para aplicação do operador de cruzamento. Para o operador BLX-α, um descendente é gerado: $H = (h_1, \ldots, h_i, \ldots, h_n)$, onde h_i é um número aleatoriamente (uniformemente) escolhido no intervalo $[c_{min} - I \cdot \alpha, c_{max} + I \cdot \alpha]$, $c_{min} = minimo(c_i^1, c_i^2)$, $c_{max} = maximo(c_i^1, c_i^2)$, $I = c_{max} - c_{min}$ e α determina o balanceamento entre prospecção (*exploitation*) e exploração (*exploration*), e seu valor deve ser definido pelo usuário. Em (Herrera et al., 1998), diferentes valores para α foram testados e o melhor balanceamento foi alcançado quando $\alpha = 0.5$. A seguir será explicado o operador de mutação não-uniforme.

Seja $C = (c_1, ..., c_i, ..., c_n)$ um cromossomo e $c_i \in [a_i, b_i]$ um gene que foi selecionado aleatoriamente para mutação. A aplicação da mutação não-uniforme sobre esse gene resultará em um gene c_i', cujo valor é definido da seguinte forma. Seja t a geração em que o operador está sendo aplicado e t_{max} o número máximo de gerações. Então

$$c_i' = \begin{cases} c_i + \Delta(t, b_i - c_i) & se \quad \tau = 0 \\ c_i - \Delta(t, c_i - a_i) & se \quad \tau = 1 \end{cases}$$

em que τ é um número aleatório binário (0 ou 1) e

$$\Delta(t, y) = y \left(1 - r^{\left(1 - \frac{t}{t_{max}}\right)^b} \right),$$

sendo r um número aleatório pertencente ao intervalo $[0, 1]$ e b um parâmetro escolhido pelo usuário que determina o grau de dependência do número de gerações. Em (Herrera et al., 1998) é sugerido $b = 5$.

Função de aptidão

A função de aptidão dos AGs para o ajuste de parâmetros de classificadores é o desempenho (taxa de acerto ou erro, por exemplo) dos modelos induzidos para dados de um conjunto de validação separado para esse propósito. O método experimental apresentado a seguir é baseado em dois laços aninhados, evitando que dados de teste sejam usados durante a fase de ajuste de parâmetros.

Dos dois laços aninhados, o laço interno é usado para determinar a melhor combinação de valores para os parâmetros do classificador, ou seja, os valores de parâmetros em que foi obtido o melhor desempenho para o conjunto de dados de validação. O laço externo é usado para estimar o desempenho do classificador gerado com os melhores parâmetros encontrados no laço interno. Os parâmetros são ajustados para cada partição do laço externo, ou seja, a otimização dos classificadores para o conjunto de validação é feita independentemente para cada partição do laço externo. Neste texto usaremos a taxa de erro como exemplo de medida de aptidão para os AGs.

A confiança das estimativas obtidas, tanto para o laço interno como para o laço externo, são afetadas pela natureza aleatória dos exemplos usados para treinamento, validação e teste. Portanto, para reduzir a variância existente, a amostragem por validação cruzada é usada para os dois laços. O número de partições para o laço interno (N_P) pode ser de qualquer tamanho desejado. Geralmente, usa-se o número de partições para o laço externo N_D menos um ($N_P = N_D - 1$), como em (Statnikov, Tsamardinos, Dosbayev e Aliferis, 2005). No laço externo, uma, dentre as N_D partições, é usada como conjunto de teste. No laço interno, uma, dentre as N_P partições, é usada como conjunto de validação.

A Figura 18.3 mostra um exemplo do uso de validação cruzada para os dois laços aninhados, com $N_D = 4$ e $N_P = 3$. O conjunto de dados D, no laço externo, é dividido em quatro partições: $d1$, $d2$, $d3$ e $d4$. As partições de treinamento utilizadas no laço externo são usadas no laço interno para treinamento e teste. A taxa de erro médio obtida no laço interno é a estimativa para a taxa de erro de teste de uma partição do laço externo. No exemplo da Figura 18.3, a taxa de erro médio de validação, que é a aptidão de um cromossomo, é 10.3%. Esta taxa de erro é a estimativa para a partição de teste $d1$, em que a taxa de erro é 10%. A taxa de erro de teste para uma solução (combinação de valores para os parâmetros) para o conjunto de dados D é a média da taxa de erro de teste para cada uma das partições do laço externo. Esse valor é 10.9% no exemplo da figura citada. A taxa de erro de validação para o conjunto de dados D, é a média das N_D taxas de erro médio obtidas no laço interno.

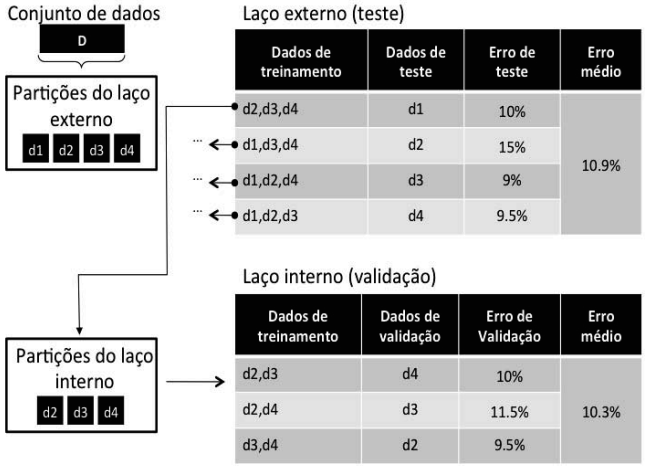


Figura 18.3: Método Experimental B.

A minimização da taxa de erro de validação é realizada pelo AG até que o critério de parada seja satisfeito. Como resposta, o AG fornece a melhor solução (valores para os parâmetros) obtida, ou seja, a solução com a menor taxa de erro (validação) médio obtida no laço interno. Esses valores para os parâmetros são usados para obter a estimativa do erro real (teste), utilizando as partições de teste no laço externo.

Suponha que exista apenas um parâmetro α a ser ajustado para um algoritmo de aprendizado A, e que α possa assumir m diferentes valores: $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_m$. O desempenho D_i de um classificador treinado pelo algoritmo de aprendizado A com o parâmetro α_i é estimado para $i = 1, \ldots, m$ no laço interno. O modelo final é construído treinando o algoritmo A com o parâmetro α_{melhor} no laço externo, em que $melhor = argmax(D_i)$, para $i = 1, 2, \ldots, m$. Para cada combinação de valores dos parâmetros, o algoritmo de aprendizado é executado $N_D \times N_P$ vezes. O Algoritmo 1 é usado para implementar este método experimental.

2. Indução de Árvores de Decisão com Algoritmos Genéticos

As Árvores de Decisão (ADs) são uma técnica muito poderosa e amplamente utilizada em problemas de classificação. Isso pode ser explicado por diversos fatores, dentre eles (Tan et al., 2005): (i) quando não muito grandes, são fáceis de interpretar em virtude da forma de representação do conhecimento adquirido - uma AD é uma representação gráfica e pode ser facilmente convertida em

Algoritmo 1 Método experimental para ajuste de parâmetros.

```
D ← conjunto de dados
l ← 1
enquanto l ≤ N_D faça
  conjunto_treino ← (N_D − 1) partições de D
  conjunto_teste ← partição restante de D
  i ← 1
  enquanto i ≤ m faça
    n ← 1
    enquanto n ≤ N_P faça
      conjunto_treino_validação ← (conjunto_treino − 1) partições
      conjunto_teste_validação ← partição restante de conjunto_treino
      C ← Treinar A com conjunto_treino_validação com parâmetro α_i
      P(n) ← Testar classificador C para o conjunto_teste_validação
      n ← n + 1
    fim enquanto
    D(i) ← média(P)
    i ← i + 1
  fim enquanto
  α_melhor ← argmax(D(i))
  M ← Treinar A com conjunto_treino usando α_melhor
  ρ(l) ← Testar classificador M para conjunto_teste
  l ← l + 1
fim enquanto
retorno média(ρ)
```

regras; (ii) robusta à presença de ruídos; (iii) baixo custo computacional tanto para indução como para utilização das árvores, mesmo para grandes conjuntos de dados; e (iv) lida bem com atributos redundantes e irrelevantes, os quais prejudicam o desempenho dos modelos.

Para ilustrar o funcionamento básico de uma árvore de decisão, pode ser considerado o problema de classificar clientes como bons ou maus pagadores. Supondo que um cliente solicite um crédito no banco, como verificar se o perfil do cliente indica se ele será um bom ou um mau pagador? A primeira pergunta que pode ser feia ao cliente é a sua renda. Se a renda for alta, a chance de ser um bom pagador deve ser um pouco maior. Também é importante saber o estado civil do cliente e a quantidade de filhos, se houver. Se o cliente for casado, também é importante a informação sobre a renda do cônjuge, o que pode ser decisivo na decisão.

O exemplo anterior ilustra uma forma de solucionar um problema de classificação por meio de perguntas sobre uma série de características de um objeto, que no caso é o cliente. A cada pergunta respondida, outra pode se realizada até que se chegue a uma conclusão sobre a classe que pertence o objeto. Essa série de perguntas e suas possíveis respostas podem ser organizadas na forma de uma árvore de decisão, a qual é uma estrutura hierárquica composta por nodos e arestas. A Figura 18.4 ilustra uma possível árvore decisão para o problema dos clientes, os quais podem ser classificados como bom pagador (SIM) ou mau pagador (NÃO). Segundo (Tan et al., 2005), a árvore possui três tipos de nodo:

- um **nodo raiz** que não possui nenhuma aresta de entrada e zero ou mais arestas de saída.
- **nodos internos**, cada qual com exatamente uma aresta de entrada e duas ou mais arestas de saída.

- nodos **folha** ou **terminal**, cada qual com uma única aresta de entrada e nenhuma de saída.

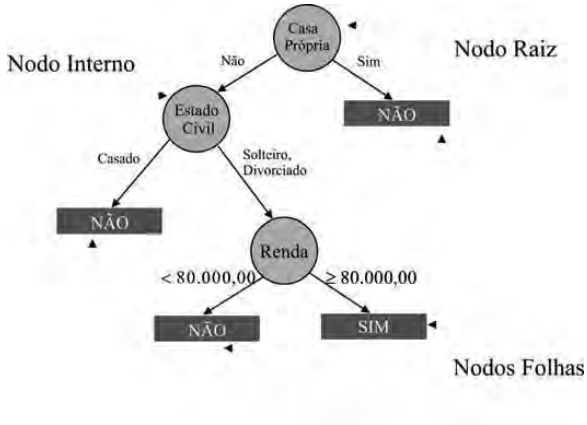


Figura 18.4: Uma árvore de decisão para o problema de classificação de clientes.

Assim, é possível utilizar uma árvore de decisão para classificar um novo cliente como bom ou mau pagador. Para isso, basta partir do nodo raiz da árvore e ir percorrendo-a, através das respostas aos testes dos nodos não-folha, até chegar em um nodo folha, o qual indica a classe correspondente do novo cliente. Além da obtenção da classe, a grande vantagem é que a trajetória percorrida até o nodo folha representa uma regra, facilitando a interpretabilidade do modelo pelo usuário.

Indução de árvore de decisão

Há muitas maneiras de uma árvore de decisão ser estruturada a partir de um conjunto de atributos. De forma exaustiva, o número de árvores de decisão possíveis cresce exponencialmente à medida que aumenta o número de atributos. Logo, fica impraticável encontrar exaustivamente a estrutura da árvore de decisão ótima para um determinado problema devido ao elevado custo computacional envolvido nessa busca. Nesse sentido, algoritmos baseados em heurísticas têm sido desenvolvidos para a indução de árvores de decisão. Mesmo que eles não garantam a solução ótima, apresentam resultados satisfatórios em tempo factível.

Geralmente, algoritmos de indução de AD usam estratégia gulosa, *top down* e com particionamento recursivo para a construção da árvore. No entanto, há pelo menos dois problemas relacionados a essas características: (i) estratégias gulosas geralmente produzem soluções ótimas locais ao invés de globais, (ii) particionamento recursivo degrada iterativamente a qualidade do *dataset*, pois quanto maior o número de vezes que o *dataset* é particionado, menores são os sub-conjuntos resultantes dessas partições e, conseqüentemente, tornam estatisticamente não significativos os resultados obtidos por quaisquer critérios de manipulação de dados (ex: cálculo de entropia). Além disso, o número de exemplos pertencentes a um nodo folha da árvore pode ser tão pequeno que torna também não estatisticamente significativa a decisão sobre a classe representada por esse nodo folha, contribuindo para o super-aprendizado (*overfitting*) do conjunto de treino (Tan et al., 2005).

Para lidar com essas dificuldades, diferentes abordagens foram sugeridas, mas que também apresentam problemas. Tais abordagens podem ser divididas em duas tendências básicas: (i) divisões múltiplas nos nodos internos; e (ii) geração de múltiplas árvores, combinando diferentes visões sobre o mesmo domínio. Abordagens baseadas em (i) resultam nas chamadas *Option Trees* (Buntine, 1993). *Option Trees* são difíceis de interpretar, comprometendo o que pode ser considerada a principal virtude das árvores de decisão. As abordagens baseadas em (ii) são mais interessantes, pois podem

fornecer uma única classificação a partir de diferentes árvores de decisão de acordo com um dado critério (ex: a classe resultante pela maioria das árvores). Uma vez que essa abordagem pode gerar melhores resultados quando comparada aos algoritmos que induzem uma única AD, em geral não resolve o problema de convergir para um ótimo local. Além disso, também degrada a interpretabilidade do modelo resultante, pois é composto por várias árvores de decisão, as quais podem ser totalmente diferentes umas das outras.

Métodos *ensemble* como *random forests*, *boosting* e *bagging* são as mais conhecidas abordagens baseadas em (ii). *Random forests* (Nadeau e Bengio, 2003) criam um conjunto inicial de árvores baseadas nos valores do conjunto independentes de vetores aleatórios, gerados por uma distribuição de probabilidades de um conjunto de treino. Um esquema de voto majoritário é usado para combinar os resultados das árvores. Algoritmos *boosting* (Quinlan, 1996) criam um conjunto inicial de árvores ao adaptativamente alterar a distribuição do conjunto de treino. Essas árvores são então combinadas considerando um sistema de votação ponderada. *Bagging* (Quinlan, 1996) quando aplicado em árvores de decisão pode ser considerado um caso particular de *random forests*. As árvores também são combinadas por uma regra de votação majoritária.

Todos esses métodos Ensemble têm um problema em comum, que é o esquema de combinar as árvores geradas. É de conhecimento geral que um *ensemble* de classificadores melhora a acurácia preditiva quando comparada a único classificador. Em contra partida, o uso de *ensembles* também tende a reduzir a interpretabilidade do modelo preditivo quando comparado a um único classificador. Um único modelo preditivo compreensível pode ser interpretado por um especialista, porém na prática é bastante complicado para um especialista interpretar uma combinação de modelos compreensíveis. Além de ser tediosa e demorada tal interpretação para o especialista, há também um problema fundamental: os modelos de classificação que compõem um *ensemble* geralmente estão sujeitos à inconsistências, as quais são necessárias para aumentar a acurácia preditiva do *ensemble*. Uma vez que cada modelo pode ser considerado como uma hipótese para explicar os padrões preditivos, isso significa que um *ensemble* não representa uma única hipótese coerente sobre os dados, mas sim um conjunto grande de hipóteses inconsistentes, as quais em geral podem confundir o especialista do domínio (Freitas et al., 2008).

Técnicas de algoritmos evolutivos também já foram exploradas no contexto de indução de árvores de decisão. Trabalhos como (Cantú-Paz e Kamath, 2000) e (Kretowski, 2004) usam tais técnicas para indução de Árvores de Decisão Oblíquas. Esse tipo de árvore, entretanto, difere das árvores de decisão tradicionais por considerar, em cada nodo interno, combinação linear de atributos, ao invés de um único atributo, para particionar o conjunto de exemplos. Como é de conhecimento geral, encontrar a melhor árvore de decisão oblíqua é um problema NP-completo, motivando os autores de (Cantú-Paz e Kamath, 2000) e (Kretowski, 2004) a proporem técnicas evolutivas para selecionar a melhor combinação de atributos em cada divisão, evitando a estratégia gulosa.

Uma das sub-áreas de AE, Programação Genética tem sido amplamente utilizada como método de indução de árvores de decisão, em trabalhos como (Koza, 1991; Zhao, 2007; Estrada-Gil et al., 2007; Bot e Langdon, 2000; Loveard e Ciesielski, 2001, 2002; Shirasaka et al., 1998; Zhao e Shirasaka, 1999; Tür e Güvenir, 1996). O trabalho de Koza (Koza, 1991) foi o pioneiro na indução de AD com PG, convertendo os atributos do problema das condições climáticas (Quinlan, 1986) em funções descritas em expressões S em LISP. Em (Zhao, 2007), é proposta uma ferramenta que permite ao usuário configurar diferentes parâmetros para gerar o melhor programa de computador para induzir árvores de classificação. Os autores de (Zhao, 2007) consideram o custo dos erros de classificação em uma abordagem multi-objetiva para definir uma árvore ótima. Em (Estrada-Gil et al., 2007), foi implementado um algoritmo de indução de árvores no contexto de bioinformática para detectar interações em variantes genéticas. Bot e Langdon (Bot e Langdon, 2000) propuseram uma solução para indução de árvores de classificação usando PG, em que um nodo intermediário é uma combinação linear de atributos. Em (Loveard e Ciesielski, 2001), os autores propuseram diferentes alternativas

para representar um problema de classificação multi-classes através de PG, e após estenderam seu trabalho (Loveard e Ciesielski, 2002) para lidar com atributos nominais. Em (Shirasaka et al., 1998) e (Zhao e Shirasaka, 1999), os autores propuseram o projeto de árvores de classificação binária usando PG, e em (Tür e Güvenir, 1996) é descrito um algoritmo de PG para indução de árvores considerando somente atributos binários.

Neste pronto, é importante discutir uma questão de terminologia. Em PG cada indivíduo da população é um programa de computador (contendo dados e operadores/funções aplicadas aos dados), e esse indivíduo geralmente é representado como uma estrutura de árvore. Idealmente, um programa de computador deveria ser genérico suficiente para processar qualquer instância do problema alvo (no nosso caso, o programa evoluído deveria ser hábil para induzir árvores de decisão para qualquer dataset de classificação para qualquer domínio de aplicação). Pode ser notado, entretanto, que nos algoritmos de PG citados cada indivíduo é uma árvore de decisão para um *dataset* a ser minerado, e não um programa de computador genérico responsável por induzir árvores de decisão para qualquer *dataset*. Uma vez que os trabalhos mencionados nomeiam seus respectivos algoritmos como PG, na realidade são AGs, pois enfatiza o fato de evoluir árvores de decisão (as quais não são programas de computador), isto é, a solução não é um algoritmo de indução (como o C4.5) mas sim a própria árvore de decisão.

Considerado os AGs, não há muitos trabalhos relacionados que os utilizam para indução de árvores de decisão. Em (Carvalho e Freitas, 2004), os autores combinam árvores de decisão e AG para lidar com problemas de pequenos disjuntos (basicamente, regras que cobrem poucos exemplos) em problemas de classificação. Essa abordagem busca incrementar a acurácia de um algoritmo *top-down* tradicional ($C4.5$) ao usar um AG para descobrir regras de pequenos disjuntos, uma vez que grandes disjuntos são classificados com sucesso pelo $C4.5$. Tal abordagem apresentou resultados interessantes com um ganho significativo de acurácia, porém mais custoso computacionalmente. Esse trabalho não tem por objetivo gerar uma árvore ótima, mas sim gerar uma árvore usando o $C4.5$ e então substituindo algumas regras extraídas das árvores (as regras de pequenos disjuntos) por regras evoluídas pelo AG.

A indução de árvores de decisão por meio de AG é apresentado em (Kretowski e Grzes, 2005) e (Z. Bandar e McLean, 1999). Em (Kretowski e Grzes, 2005), a função de *fitness* seleciona qual indivíduo tem a maior probabilidade de sobreviver para a próxima geração por dois conceitos: Q_{RCLass}, qualidade de classificação estimada no conjunto de treino, e tamanho da árvore, a qual é controlada por um parâmetro indicado pelo usuário. Embora essa abordagem seja considerada muito interessante, não está claro no artigo como os autores calculam a medida Q_{RClass}, nem como os autores aplicaram essa medida para comparar seu algoritmo com o $C4.5$. Em (Z. Bandar e McLean, 1999) cada árvore de decisão tem um número fixo de nodos que são representados como um vetor estático, e somente divisões binárias dos nodos são permitidas. Essa abordagem impõe a restrição de que atributos nominais com mais de dois valores possíveis devem ter seus conjuntos de valores reduzidos para dois via um agrupamento de valores, o qual é um processo não natural e pode degradar a acurácia em alguns casos. Embora o número de divisões igual a dois possa induzir o leitor a pensar que as árvores resultantes são mais simples, isso não é verdade pois as divisões binárias geram árvores com mais níveis de profundidade que, conseqüentemente, resultarão em regras mais difíceis de serem interpretadas.

Como alternativa aos algoritmos apresentados anteriormente, Basgalupp et al. propuseram em (Basgalupp, Barros, de Carvalho, Freitas e Ruiz, 2009) e (Basgalupp, de Carvalho, Barros, Ruiz e Freitas, 2009) o algoritmo LEGAL-Tree (LExicographic Genetic ALgorithm for decision Tree induction), um algoritmo genético baseado no critério multi-objetivo lexicográfico para indução de árvores de decisão precisas e interpretáveis. Esses trabalhos são pioneiros na utilização da avaliação lexicográfica para indução de árvores de decisão, considerando não somente a acurácia preditiva das árvores mas também a interpretabilidade das mesmas, a qual é medida pelo número de nodos da árvore. Os resultados obtidos foram satisfatórios, superando o algoritmo C4.5, bastante conhecido na literatura. A seguir, é feita uma descrição sobre cada detalhe de implementação do algoritmo LEGAL-Tree.

LEGAL-Tree

Representação da solução

Na grande maioria dos trabalhos que utilizam algoritmos genéticos, cada indivíduo é representado por *strings* binários ou *strings* numéricos. Em LEGAL-Tree, cada indivíduo é uma própria árvore de decisão, que é uma representação bastante lógica e intuitiva. Cada indivíduo é, portanto, um conjunto de nodos, que podem ser terminais (folhas) ou não-terminais (nodos de decisão). Cada nodo não-terminal representa um teste sobre um atributo preditivo e cada folha é rotulada com um valor do atributo classe. Um conjunto de arestas ligando cada nodo não-terminal com seus respectivos filhos também faz parte da representação da árvore. Há duas situações possíveis de relacionamento entre nodos:

- Relacionamento entre um nodo categórico e seus filhos: se um nodo x representa um atributo categórico, o nodo irá conter n arestas, em que n é o número total de categorias (valores) para o atributo;
- Relacionamento entre um nodo numérico e seus filhos: se um nodo x representa um atributo numérico, haverá uma divisão binária de acordo com um valor de *threshold*, o qual é determinado automaticamente pelo algoritmo.

Gerando a população inicial

Mesmo que construção totalmente aleatória de árvores seja a técnica mais comum para gerar populações iniciais em aplicações de AG, seria bem mais interessante incorporar conhecimento da tarefa de classificação para induzir as árvores de decisão. Assim, utilizando conhecimento sobre o significado das árvores de decisão no contexto da tarefa de classificação, espera-se derivar melhores soluções, ou pelo menos soluções tão boas quanto, em menos gerações. Nesse contexto, optou-se por utilizar uma estratégia de incorporar conhecimento específico no AG, a qual pode ser definida no pseudo-código ilustrado em Algoritmo 2.

Algoritmo 2 Geração dos *Decision Stumps*

```
dado x como o conjunto de treino
divida x em 10 partes diferentes
dado dsList como a lista de decision stumps
dado y_j representando a j^th parte de x
para y_i = 1 to 10 faça
  dado a_i como o i^th atributo do conjunto de treino
  para a_i = 1 até numeroDeAtributos faça
    se a_i é numérico então
      threshold = infoGain(a_i)
      dsList.add(new numericDS(a_i,threshold))
    senão
      dsList.add(new categoricalDS(a_i))
    fim se
  fim para
fim para
```

Um *decision stump* (DS) é o caso mais simples de árvores de decisão, o qual consiste de um único nodo de decisão (não-terminal) e duas folhas (Freund e Mason, 1999). Em LEGAL-Tree, esse conceito foi estendido para suportar atributos categóricos, em que cada aresta que representa uma categoria

será ligada a um nodo folha. Assim, poderá haver mais de duas folhas caso o nodo de decisão represente um atributo que possui mais de duas categorias possíveis. O pseudo-código ilustrado no Algoritmo 2 irá basicamente gerar um conjunto de *decision stumps* (na verdade, 10 X n *decision stumps*, em que n é o número de atributos preditivos do conjunto de treino). Ao dividir o conjunto de treino em 10 partições disjuntas, espera-se atingir um certo nível de heterogeneidade para os *decision stumps*, pois eles serão essenciais para a geração da população inicial, a qual deve ser mais diversificada possível. Tal processo de geração de *decision stumps* não deixa de ser aleatório, porém evita geração de soluções de baixa qualidade e não factíveis, que podem atrasar o processo de convergência. Assim, espera-se convergir para boas árvores de decisão em um menor número de gerações que em um processo de geração totalmente aleatório.

O passo final da geração da população inicial é combinar os diferentes *decision stumps* que foram criados. O usuário configura uma profundidade máxima para as árvores que formarão a população inicial, e o algoritmo seleciona os *decision stumps* aleatoriamente e então constrói as árvores de acordo com a profundidade também gerada aleatoriamente, variando de 1 até o valor máximo informado pelo usuário. A Figura 18.5 ilustra esse raciocínio, em que três *decision stumps* (A, B e C) são selecionados e usados para construir uma árvore completa de profundidade 2 (dois). Para o nodo raiz da árvore, foi selecionado o *decision stump* A, cujos filhos foram substituídos pelos *decision stumps* B e C, respectivamente. Observa-se que todas as árvores geradas na população inicial são árvores completas. Entretanto, as árvores que irão formar as próximas gerações da população não serão necessariamente completas em virtude da aplicação dos operadores genéticos, os quais afetarão suas estruturas.

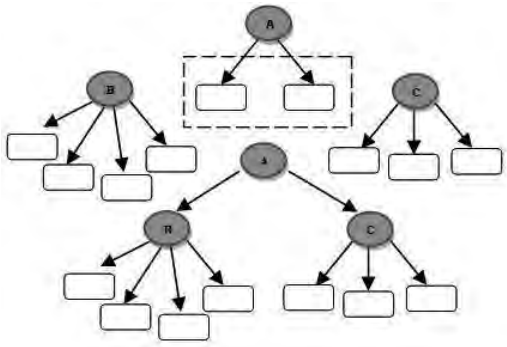


Figura 18.5: Combinando três decision stumps (A, B e C) para construção de um indivíduo de profundidade 2.

As duas principais vantagens desta abordagem para a geração da população inicial de árvores de decisão são que: (a) ao invés de aleatória, a escolha dos *thresholds* para atributos numéricos em nodos não-terminais é baseada no ganho de informação (Tan et al., 2005), utilizando conhecimento do conjunto de dados; e (b) a classe associada ao nodo folha é sempre a mais freqüente nos exemplos cobertos pelo nodo ao invés de uma classe escolhida aleatoriamente. Ambas as vantagens são resultado da utilização do conhecimento geral sobre a tarefa de classificação (ao invés de um conhecimento específico do domínio de aplicação) no AG, o qual tende a melhorar sua efetividade, como discutido previamente.

Tratamento de valores desconhecidos

Foi escolhida a seguinte estratégia para tratamento de valores desconhecidos:

- Atributos categóricos: se o nodo a ser analisado representa um atributo categórico, os valores ausentes são substituídos pela moda de todos os valores do atributo no conjunto de treino;

- Atributos numéricos: se o nodo a ser analisado representa um atributo numérico, os valores ausentes são substituídos pela média aritmética de todos os valores do atributo no conjunto de treino.

Função de *fitness* multi-objetiva baseada na abordagem lexicográfica

Uma questão crucial em mineração de dados é como avaliar a qualidade de um modelo candidato (Freitas, 2004). Considerando LEGAL-Tree, em que cada indivíduo é um candidato a modelo de classificação (árvore de decisão), a função de *fitness* é vital para determinar os melhores indivíduos de forma a evoluir a população e convergir para soluções próximas à(s) ótima(s).

É também de senso comum que em muitos domínios de aplicação o conhecimento descoberto por um algoritmo de mineração não deve ter somente uma boa acurácia (taxa de acerto) mas também deve ser compreensível pelo usuário, em que compreensão (interpretabilidade) é normalmente estimada pelo tamanho do classificador - classificadores menores são preferíveis, uma vez que outros critérios permaneçam iguais. Com isso, em LEGAL-Tree é proposta uma função de *fitness* multi-objetiva para cada indivíduo, isto é, considera mais que uma medida para avaliar a sua qualidade.

Nesse contexto, é apresentada a abordagem lexicográfica, que consiste basicamente em estabelecer diferentes prioridades a diferentes objetivos (medidas) e então focar na otimização desses objetivos de acordo com as respectivas ordens de prioridade (Freitas, 2004). Até o momento, LEGAL-Tree considera somente a acurácia e tamanho da árvore de decisão como objetivos, uma vez que são as medidas mais utilizadas para avaliação de árvores de decisão. Entretanto, esse conjunto de objetivos pode ser facilmente estendido para um número n de objetivos de acordo com o domínio de aplicação a ser considerado, tais como profundidade da árvore, custo dos atributos utilizados, desempenho, etc.

A abordagem convencional de fórmula ponderada sofre de diversos problemas, tais como o problema dos "números mágicos"(configurar os pesos na fórmula é um procedimento *ad-hoc*), misturar maças com laranjas (misturar critérios não comparáveis como acurácia e tamanho da árvore) e misturar diferentes unidades de medidas (realizar operações com diferentes escalas e induzir *bias* quando escolher um procedimento de normalização) (Freitas, 2004). A abordagem de Pareto também tem seus problemas, e o principal deles é a dificuldade de escolher a melhor solução a ser utilizada na prática, pois o resultado dessa abordagem não é apenas uma árvore e sim um conjunto de árvores não-dominadas. Uma alternativa seria combinar as árvores encontradas, porém seria considerado o caso de um *ensemble*, que apresentam problemas indesejáveis no contexto deste trabalho. Outro problema da abordagem de Pareto é a dificuldade de lidar com diferentes níveis de prioridade, isto é, quando um objetivo é significativamente mais importante que um outro. Em árvores de decisão, a acurácia é consideravelmente mais importante que tamanho da árvore, mas a abordagem de Pareto assume que ambos os objetivos são igualmente relevantes. A abordagem lexicográfica não sofre dos problemas mencionados, é conceitualmente simples, fácil de usar, implementar e estender para objetivos adicionais.

Seleção

Para selecionar os indivíduos que formação as próximas gerações da população, LEGAL-Tree usa o método do torneio, um método popular e bastante efetivo. É também implementada a seleção por elitismo, a qual mantém os melhores indivíduos nas próximas gerações. O número indivíduos que fazem parte da elite é um parâmetro que pode ser definido pelo usuário.

Cruzamento

LEGAL-Tree implementa a operação de cruzamento como segue. De acordo com um número gerado aleatoriamente entre 1 (nodo raiz) e n (número total de nodos da árvore), LEGAL-Tree realiza uma busca pré-fixada, visitando recursivamente o nodo raiz e então seus filhos da esquerda para a direita. Para nodos numéricos, o método de busca é equivalente à tradicional busca binária pré-fixada. Nos caso em que o nodo é categórico e possui mais de 2 filhos, o método de busca visita cada filho da esquerda para a direita, de acordo com um índice que identifica cada nodo filho. Após identificar os nodos que representam o número sorteado para ambas as as árvores, LEGAL-Tree troca as respectivas sub-árvores representadas pelos nodos sorteados, os quais representam os nodos raíz de cada sub-árvore.

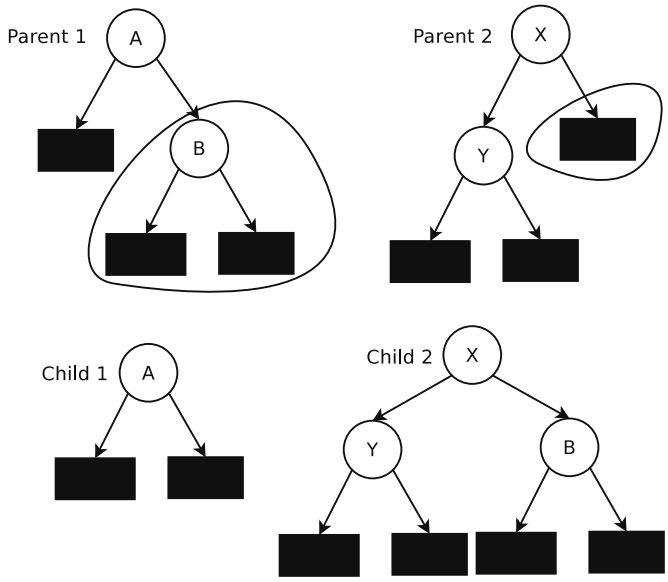


Figura 18.6: Cruzamento entre dois indivíduos, $Parent1$ e $Parent2$, formando dois novos indivíduos, $Child1$ e $Child2$.

Considere os dois indivíduos apresentados como "*Parents*"na Figura 18.6. Para $Parent1$, o nodo sorteado foi o C, enquanto que para o $Parent2$ foi sorteado o nodo M. Após identificar esses dois nodos pelo método de busca supracitado, a operação cruzamento irá criar dois indivíduos filhos ($Child1$ e $Child2$) que serão resultados da estrutura de um dos pais com uma parte do outro. $Child1$ mantém a estrutura de $Parent1$, mas herda o nodo M (e tudo o que estiver abaixo) de $Parent2$. Similarmente, $Child2$ mantém a estrutura de $Parent2$ e herda o nodo C de $Parent1$.

Ao trocar sub-árvores inteiras dos nodos sorteados ao invés de nodos específicos, espera-se evitar problemas como irregularidade de domínios, pois cada aresta representa um possível valor para o atributo específico de um nodo. Entretanto, a operação de cruzamento não evita a possibilidade de regras redundantes (mesmo atributo categórico representado por um nodo x sendo utilizado em outro nodo das sub-árvores de x) e inconsistências (*thresholds* que resultam em intervalos irregulares ao utilizar nodos numéricos). A Seção "Factibilidade das soluções"mostra detalhes de como LEGAL-Tree trata essas questões.

Mutação

LEGAL-Tree implementa duas estratégias diferentes para mutação dos indivíduos. A primeira consiste em trocar uma sub-árvore qualquer de um indivíduo por um nodo folha representando a classe mais freqüente para os exemplos cobertos por essa folha. A segunda estratégia utiliza os *decision stumps*, os quais foram criados durante a geração da população inicial. Essa estratégia substitui um nodo folha, também selecionado aleatoriamente do indivíduo, por um *decision stump* qualquer. A Figura 18.7 ilustra ambas as estratégias apresentas, as quais são denominadas $Mutation1$ e $Mutation2$, respectivamente.

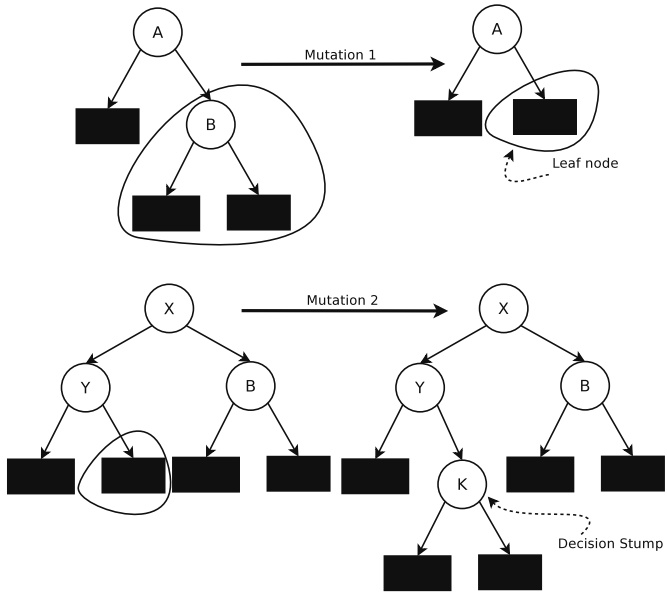


Figura 18.7: Duas estratégias, $Mutation1$ e $Mutation2$, para mutação dos indivíduos.

Tais estratégias podem aumentar ou diminuir o tamanho dos indivíduos, aumentando a diversidade da população e evitando convergências prematuras para ótimos locais. A taxa de mutação é um parâmetro configurado pelo usuário e geralmente possui um valor baixo.

Factibilidade das soluções

Após as operações de cruzamento e mutação e antes do ciclo do AG se repetir para as próximas gerações, há casos em que são gerados cenários inconsistentes. Por exemplo, considere que uma sub-árvore de uma árvore a foi substituída por uma sub-árvore de uma árvore b gerando um indivíduo filho durante o processo de cruzamento. Se a nova sub-árvore tem um nodo que representa um atributo já utilizado por um nodo ancestral, ações devem ser tomadas de modo a evitar regras redundantes ou *thresholds* que geram intervalos inconsistentes. Se o nodo é categórico e um ancestral representa o mesmo atributo, então caracteriza-se um caso de redundância que deve ser eliminado. Nesse caso, LEGAL-Tree substitui toda a sub-árvore por um nodo folha representando a classe mais freqüente. Por outro lado, se o nodo é numérico e seu *threshold* é inconsistente com algum nodo ancestral, então o *threshold* é ajustado para evitar caminhos que não contemplem nenhum exemplo.

Esses problemas são tratados por LEGAL-Tree por meio de filtros aplicados após as operações de cruzamento e mutação. Esses filtros são similares aos utilizados na geração da população inicial,

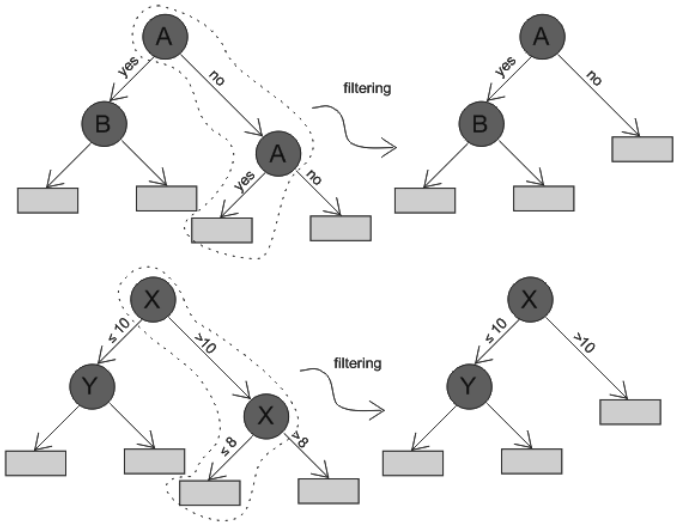


Figura 18.8: Dois tipos de inconsistência originados por repetição de atributos categórico (A) e numérico (X).

quando são evitados esses problemas durante a combinação dos *decision stumps* para a geração das árvores de decisão. A Figura 18.8 ilustra os dois casos de inconsistência supracitados.

Outro problema que pode ocorrer após as operações de cruzamento e mutação é quando um nodo folha deixa de representar a classe mais freqüente para os exemplos cobertos por ele. Essa inconveniência também é tratada por LEGAL-Tree pelos mesmos filtros, que recalculam o valor da classe mais freqüente para as folhas modificadas. O tratamento dessas questões foi considerado um grande passo na construção de LEGAL-Tree, aumentando a velocidade de convergência para soluções próximas à(s) ótima(s).

3. Decomposição de Problemas Multiclasses

Um problema de classificação com apenas duas classes é conhecido como problema de classificação binário. Um exemplo de um problema de classificação binário é o diagnóstico médico de uma doença em particular. Neste problema, o classificador induzido usa dados clínicos de um paciente para determinar se ele possui uma determinada doença. As classes representam a presença ou ausência da doença. Vários problemas reais, porém, envolvem a discriminação de mais que duas categorias ou classes. Exemplos incluem o reconhecimento de dígitos manuscritos (Knerr et al., 1992; LeCun et al., 1995), a distinção de múltiplos tipos de câncer (Statnikov, Aliferis, Tsamardinos, Hardin e Levy, 2005) e a categorização de textos (Berger, 1999; Ghani, 2000).

Várias técnicas de AM foram originalmente formuladas para a solução de problemas de classificação binários. Entre elas, pode-se mencionar SVMs (Cristianini e Shawe-Taylor, 2000; Lorena e Carvalho, 2008) e o algoritmo RIPPER (Cohen, 1995). Duas abordagens têm sido adotadas na literatura para generalizar essas técnicas a problemas multiclasses: (a) adaptação das operações internas de seus algoritmos de aprendizado; (b) combinação de classificadores binários por meio de uma decomposição do problema multiclasses em uma série de problemas de classificação binários.

A extensão de um algoritmo de aprendizado binário a uma versão multiclasses pode ser difícil

de ser realizada para algumas técnicas (Passerini et al., 2004). Para SVMs, em particular, Hsu and Lin (Hsu e Lin, 2002) observaram que a reformulação dessa técnica em versões multiclasses leva a algoritmos computacionalmente custosos. É comum recorrer-se então à alternativa de decompor o problema multiclasses em subproblemas binários, uma estratégia denominada decomposicional.

Segundo Moreira e Mayoraz (Moreira e Mayoraz, 1998), existe uma série de motivações para empregar estratégias decomposicionais na solução de problemas multiclasses. Além de haver técnicas cuja formulação é originalmente binária, alguns algoritmos não são adequados a problemas com um número elevado de classes ou apresentam dificuldades em lidar com grandes volumes de dados de treinamento. Mesmo que o algoritmo seja capaz de trabalhar com problemas de grande escala, o uso de um procedimento decomposicional pode reduzir a complexidade computacional envolvida na solução do problema total, por meio da divisão deste em subtarefas mais simples.

Pimenta (Pimenta, 2005) também aponta vantagens no uso da decomposição em problemas cujos erros de classificação das diferentes classes possuem pesos distintos. Pode-se assim gerar preditores binários de maneira a impor preferências para algumas classes. O autor menciona ainda o fato de que os subproblemas de classificação gerados em uma decomposição são independentes, podendo ser solucionados paralelamente.

O emprego de técnicas decomposicionais envolve dois passos. No primeiro, realiza-se a divisão do problema multiclasses em subproblemas binários, determinando assim os classificadores binários a serem gerados. A segunda etapa pode ser denominada agregação ou reconstrução e refere-se à forma como as saídas dos classificadores binários são combinadas na determinação da classe de um exemplo (Mayoraz e Moreira, 1997).

O *Framework* de Matrizes de Códigos

Várias estratégias podem ser empregadas na decomposição de um problema multiclasses em uma série de problemas de classificação binários. Essas decomposições podem ser genericamente representadas por meio do *framework* de matrizes de códigos descrito em (Allwein et al., 2000), em que as decomposições são representadas por uma matriz de códigos $\mathbf{M}$. As linhas dessa matriz contêm códigos que são atribuídos a cada classe. As colunas de $\mathbf{M}$ definem partições binárias das k classes e correspondem aos rótulos que essas classes assumem na geração dos classificadores binários. Tem-se então uma matriz de dimensão kxl, em que k é o número de classes do problema e l representa o número de classificadores binários utilizados na solução multiclasses. Na Figura 18.9 é apresentado um exemplo de matriz de códigos na qual o número de classes k é igual a 4 (quatro) e o número de classificadores binários l também é 4 (quatro). Abaixo dessa matriz de códigos são indicadas as partições binárias das classes realizadas por cada classificador binário representado em suas colunas.

Cada elemento da matriz $\mathbf{M}$ assume valores em $\{-1, 0, +1\}$. Um elemento m_{ij} com valor $+1$ indica que a classe correspondente à linha i assume rótulo positivo na indução do classificador f_j. O valor -1 designa um rótulo negativo e um valor 0 indica que os dados da classe i não participam do processo de indução do classificador f_j. Classificadores binários são então treinados de forma a aprender os rótulos representados nas colunas de $\mathbf{M}$.

Um novo dado $\mathbf{x}$ pode ser classificado avaliando-se as predições dos l classificadores, que geram um vetor $\mathbf{f}(\mathbf{x})$ de tamanho l na forma $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_l(\mathbf{x}))$. Esse vetor é então comparado às linhas de $\mathbf{M}$. O dado é atribuído à classe cuja linha de $\mathbf{M}$ é mais próxima de $\mathbf{f}(\mathbf{x})$ de acordo com alguma medida de distância. Esse processo de reconstrução também é referenciado como decodificação em alguns trabalhos (Passerini et al., 2004).

O número total de diferentes preditores binários para um problema com k classes é $0{,}5\big(3^k + 1\big) - 2^k$, considerando que $f = -f$, ou seja, que a inversão das classes positivas e negativas produz o mesmo classificador (Mayoraz e Moreira, 1997). Desses, $2^{k-1} - 1$ incluem todas as classes simultaneamente, ou seja, possuem somente rótulos $+1$ e -1, sem o elemento 0 (Mayoraz e Moreira, 1997). Para quatro

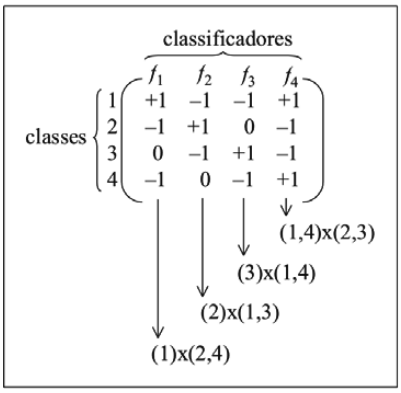


Figura 18.9: Exemplo de matriz de códigos para um problema com quatro classes

classes, tem-se nesse caso a matriz representada na Figura 18.10b. Por sua vez, a decomposição mais compacta de um problema com k classes pode ser realizada com o uso de $l = \lceil \log_2(k) \rceil$ classificadores binários (Mayoraz e Moreira, 1997). Um exemplo de matriz compacta para um problema com quatro classes é apresentado na Figura 18.10a.

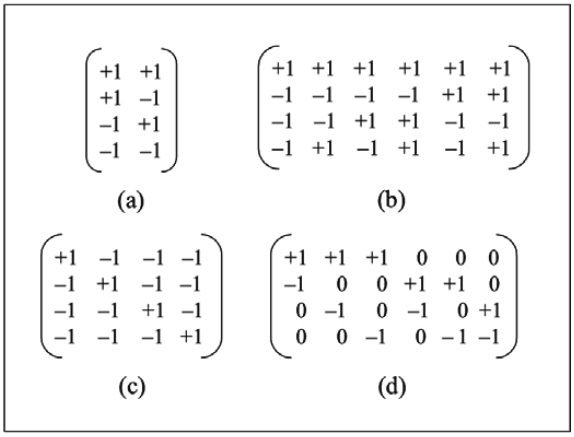


Figura 18.10: Matrizes de diferentes decomposições para um problema com quatro classes

Entre as estratégias decomposicionais mais comuns encontradas na literatura estão a um-contra-todos (*one-against-all* - OAA), a todos-contra-todos (*all-against-all*, também denominada *one-against-one* - OAO) (Hastie e Tibshirani, 1998) e a baseada em códigos de correção de erros (Dietterich e Bariki, 1995), que são descritas a seguir.

Na estratégia OAA, dado um problema com k classes, k classificadores binários $f_i(\mathbf{x})$ são gerados. Cada um deles é treinado de forma a distinguir uma classe i das demais. A representação dessa técnica é dada por uma matriz de dimensão kxk, na qual os elementos da diagonal possuem o valor $+1$ e os demais, o valor -1. Uma matriz do tipo OAA para um problema de quatro classes é apresentada na Figura 18.10c. Usualmente, na integração dos classificadores OAA, escolhe-se a classe correspondente ao classificador que produz a maior saída (Rifkin e Klautau, 2004).

A decomposição OAA pode apresentar desvantagens quando a proporção de exemplos de uma classe é muito pequena em relação a do conjunto formado pelos dados das outras classes. Esse tipo

de desbalanceamento pode dificultar a indução de um preditor que apresente bom desempenho no reconhecimento da classe considerada.

Na decomposição OAO, dadas k classes, $\frac{k(k-1)}{2}$ classificadores binários são gerados. Cada um deles é responsável por diferenciar um par de classes (i, j), em que $i \neq j$. A matriz de códigos nesse caso possui então dimensão $k\text{x}\frac{k(k-1)}{2}$ e cada coluna corresponde a um classificador binário para um par de classes. Em uma coluna representando o par (i, j), o valor do elemento correspondente à linha i é $+1$ e o valor do membro correspondente a j é igual a -1. Todos os outros elementos da coluna possuem o valor 0, indicando que os dados de outras classes não participam do processo de indução do classificador. Na Figura 18.10d tem-se uma matriz OAO para um problema com quatro classes.

Embora o número de classificadores gerados na decomposição OAO seja da ordem de k^2, o treinamento de cada um deles envolve dados de apenas duas classes. Com isso, mesmo com um número elevado de classes, o tempo total despendido na geração dos preditores geralmente não é grande.

A integração usual dos classificadores gerados na estratégia OAO é realizada por meio de um esquema de votação por maioria (Hastie e Tibshirani, 1998). Dado um novo exemplo $\mathbf{x}$, cada classificador fornece um voto em sua classe preferida. O resultado é então dado pela classe que recebeu mais votos. Um problema apontado na decomposição OAO é que a resposta de um preditor para um par de classes (i, j) na realidade não fornece informação alguma quando o exemplo não pertence às classes i ou j (Alpaydin e Mayoraz, 1999).

Em uma estratégia decomposicional alternativa, Dietterich e Bariki (Dietterich e Bariki, 1995) propuseram o uso de códigos de correção de erros para representar as k classes de um problema. Para tal, os autores sugerem que o valor do número de classificadores l seja maior que o número mínimo necessário para diferenciar cada classe unicamente. Os bits adicionais introduzem uma redundância na codificação das classes e têm a utilidade de prover ao sistema a capacidade de se recuperar de eventuais erros cometidos por alguns preditores no processo de classificação de um novo exemplo. Por esse motivo, essa técnica possui a denominação de decomposição por códigos de correção de erros (ECOC, do Inglês *Error Correcting Output Codes*). As matrizes de códigos nesse caso assumem valores em $\{-1, +1\}$ e utiliza-se a distância de *Hamming* na decodificação das saídas dos classificadores.

Com esse procedimento de correção de erros, Dietterich e Bariki (Dietterich e Bariki, 1995) sugerem que a solução de um problema multiclasses por uma técnica de AM seja vista como uma tarefa de comunicação. Dado um novo exemplo, a sua classe correta é transmitida por meio de um canal, o qual é constituído de seus atributos, dos dados de treinamento e do algoritmo de aprendizado. Devido aos erros que podem estar presentes nos atributos do exemplo, nos dados de treinamento utilizados ou em falhas no processo de aprendizado, a informação pode ser corrompida. Para prover ao sistema a capacidade de se recuperar desses erros de transmissão, a classe é codificada por um código de correção de erros e cada bit do mesmo é transmitido separadamente, ou seja, por execuções separadas do algoritmo de aprendizado.

Para possibilitar a correção de erros, Dietterich e Bariki (Dietterich e Bariki, 1995) sugerem que os códigos das classes contidos em $\mathbf{M}$ sejam bem separados segundo a distância de *Hamming*. Sendo d_l a distância mínima entre qualquer par de linhas de $\mathbf{M}$, então o classificador multiclasses final é capaz de corrigir ao menos $\left\lfloor \frac{d_l-1}{2} \right\rfloor$ bits incorretos em uma predição. Como segundo a função de decodificação de *Hamming* cada bit incorreto implica em se afastar em uma unidade do código da classe correta, cometendo $\left\lfloor \frac{d_l-1}{2} \right\rfloor$ erros, o código mais próximo ainda será o correto (Dietterich e Bariki, 1995). Segundo esse princípio, a codificação OAA é incapaz de se recuperar de qualquer erro, uma vez que $d_l = 2$.

Além disso, na construção de bons códigos de correção de erros, deve-se estimular também que os erros dos classificadores binários gerados sejam não correlacionados. Exige-se então a separação entre as colunas de $\mathbf{M}$, ou seja, a distância de *Hamming* entre cada par de colunas deve ser grande. Se no algoritmo de aprendizado a inversão das classes positivas e negativas produz o mesmo classificador

(ou seja, $f = -f$), então deve-se também fazer com que a distância de *Hamming* entre cada coluna e o complemento das outras seja grande.

Com base nessas observações, Dietterich e Bariki (Dietterich e Bariki, 1995) propuseram quatro técnicas para o desenvolvimento de matrizes de códigos com boa capacidade de correção de erros. A escolha de cada uma delas é determinada pelo número de classes do problema. Para $k \leqslant 7$, esses autores recomendam o uso de um código exaustivo, que consiste da combinação de $2^{k-1} - 1$ classificadores binários unicamente com rótulos $+1$ e -1, conforme ilustrado na Figura 18.10b para um problema com quatro classes. A distância d_l em uma matriz gerada pelo método exaustivo é de 2^{k-2}. Se $8 \leqslant k \leqslant 11$, é aplicado um método que seleciona colunas do código exaustivo. Para $k > 11$, tem-se duas opções: um método baseado no algoritmo *hill-climbing* e a geração de códigos BCH (Boser e Ray-Chaudhuri, 1960), os quais provêm da teoria de códigos de correção de erros em comunicação.

Uma crítica comum às estratégias OAA, OAO e ECOC é que todas realizam a decomposição do problema *a priori*, sem levar em consideração as propriedades e as características de cada aplicação. Essa foi a motivação de diversos trabalhos para a proposição de técnicas para encontrar matrizes de códigos adaptadas à solução de cada problema multiclasses, tais como (Lorena e Carvalho, 2007b,a; Lorena, 2006; Mayoraz e Moreira, 1997; Alpaydin e Mayoraz, 1999; Dekel e Singer, 2003; Ratsch et al., 2003), entre outros.

Na próxima seção discute-se como Algoritmos Genéticos podem ser empregados na construção de matrizes de códigos para problemas multiclasses.

Construindo Matrizes de Códigos para Problemas Multiclasses

A construção de matrizes de códigos para problemas de classificação multiclasses pode ser formulada como um problema de busca e otimização. Conforme mencionado na seção anterior, há $0.5\left(3^k + 1\right) - 2^k$ diferentes preditores binários para um problema com k classes. Um número combinatorial de combinações desses classificadores é possível, correspondendo a diferentes decomposições do problema multiclasses. Baseado nessa observação, alguns trabalhos empregaram Algoritmos Genéticos na construção de matrizes de códigos.

Verifica-se na literatura de comunicação o vasto uso de AGs para a obtenção de Códigos de Correção de Erros (*Error Correcting Codes* - ECC). Como exemplo, pode-se citar (Alba e Chicano, 2004; Alba et al., 2002; Simón et al., 2006; Wallis e Houghten, 2002; Dontas e De Jong, 1990). A construção de ECCs nesse contexto pode ser resumida a encontrar k códigos com l bits cada, que sejam capazes de corrigir um número máximo de erros pré-determinado. Esse é um problema de otimização NP-difícil (Alba e Chicano, 2004), em que geralmente se tem objetivos conflitantes: encontrar códigos com o menor tamanho possível (implicando em rápida transmissão) e maximizar d_m (para uma maior capacidade de correção de erros), o que sugere incluir mais redundância nos códigos (mais bits e, então, o uso de códigos maiores). Entretando, agrupando os k códigos em uma matriz, não se requer a separação entre colunas e também não é evitada a presença de colunas constantes (com o mesmo elemento em todas as linhas), dificultando o uso direto desses algoritmos para encontrat ECOCs para problemas multiclasses - ou seja, matrizes de códigos com propriedades de correção de erros segundo a abordagem proposta em (Dietterich e Bariki, 1995).

Pimenta (Pimenta, 2005) adaptou um algoritmo denominado GARA (*Genetic Algorithm with Repulsion Algorithm*) (Alba e Chicano, 2004) da teoria de telecomunicações para a geração de matrizes ECOC. Os cromossomos desse AG são cadeias binárias com kxl elementos, formadas pela concatenação dos códigos na matriz de códigos $\mathbf{M}$, como ilustrado na Figura 18.11. Um alfabeto binário é usado nas matrizes desse trabalho.

A função de avaliação da aptidão dos individuos neste AG considera o quanto os códigos em $\mathbf{M}$ estão separados no espaço de l dimensões. Há ainda um termo que penaliza a presença de colunas iguais e complementares nas matrizes obtidas, garantindo assim a separação entre as colunas da matriz.

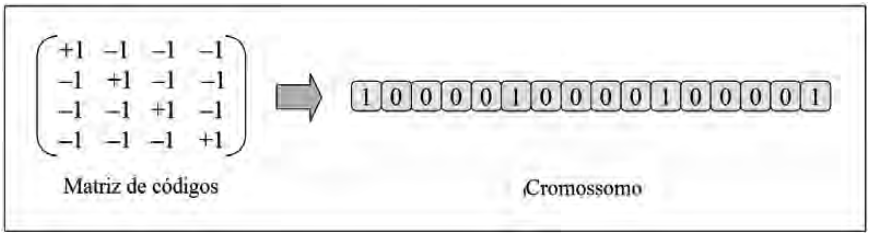


Figura 18.11: Exemplo de cromossomo na abordagem GARA (Alba e Chicano, 2004)

Empregou-se torneio binário na seleção de indivíduos para reprodução, e o operador de cruzamento utilizado foi o de um ponto. Para mutação, uma iteração do algoritmo RA (*Repulsion Algorithm*) foi utilizada como um procedimento de busca local. O algoritmo RA é baseado no comportamento físico de partículas com mesma carga em uma esfera. Nessa situação, as particícúlas movem pela esfera até que um equilíbrio seja atingido. Nessa abordagem, cada código de uma classe foi considerado como uma partícula carregada, posicionada no vértice de um hipercubo. Permite-se o movimento de códigos entre vértices vizinhos, o que equivale a inverter um bit dentro do código. O AG pára quando um número máximo de gerações é atingido ou quando uma matriz ótima, de acordo com a função de avaliação, é encontrada.

O algoritmo GARA foi comparado ao RA e a um algoritmo denominado *Persecution Algorithm* (PA), também proposto em (Pimenta, 2005). O PA adiciona e remove colunas de um ECOC inicial, buscando maximizar a sua qualidade, a qual é medida de acordo com sua capacidade de correção de erros. Em geral, o PA obteve melhores resultados para ECOCs, embora o GARA também tenha obtido bons ECOCs. A validade foi medida pelo critério de se evitar colunas iguais, complementares ou constantes nas matrizes de códigos obtidas.

Lorena e Carvalho (Lorena, 2006; Lorena e Carvalho, 2007b), por sua vez, empregaram AGs na obtenção de matrizes de códigos adaptadas a cada problema multiclasses considerado. Os AGs formulados lidam com três objetivos: (a) minimizar o erro da matriz quando esta é empregada na solução do problema multiclasses; (b) minimizar o número de colunas na matriz de códigos, representando a busca por decomposições mais simples, que empreguem menos classificadores binários; (c) evitar a presença de colunas equivalentes na matriz, colunas iguais ou complementares, as quais representam a repetição de um mesmo classificador binário na decomposição. O objetivo foi buscar uma matriz de códigos sem colunas equivalentes com um bom desempenho na solução do problema multiclasses e também com um número reduzido de classificadores binários. Duas variantes de AGs foram empregadas nesse problema multi-objetivo: uma lexicográfica, em que os múltiplos objetivos são ordenados segundo uma preferência, e outra baseada no algoritmo SPEA2 (*Strength Pareto Evolutionary Algorithm* 2) (Zitzler et al., 2002), baseado em uma técnica de otimização multiobjetivo.

Os cromossomos foram diretamente representados como matrizes de códigos, com uma codificação ternária. Assim, cada indivíduo corresponde a uma possível matriz de códigos $\mathbf{M}$ de tamanho kxl e com elementos no conjunto $\{-1, 0, +1\}$. Essa representação foi considerada mais intuitiva para representar as soluções para o problema em questão. Para determinar o número de classificadores binários nas matrizes, os indivíduos em uma mesma população continham diferentes números de colunas l. De acordo com essa estratégia, dois possíveis indivíduos para um problema com quatro classes podem ser visualizados na Figura 18.12. O usuário limita o número máximo de classificadores permitidos nas matrizes de códigos manipuladas pelo AG. As matrizes também devem ter no mínimo $\lceil \log_2 k \rceil$ classificadores binários, o que corresponde ao mínimo necessário para dividir k classes (Mayoraz e Moreira, 1997).

Na geração da população inicial, um teste de consistência foi aplicado a cada coluna das matrizes,

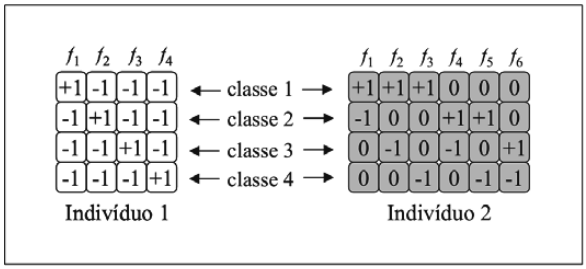


Figura 18.12: Ilustração de dois possíveis indivíduos para um problema com quatro classes

garantindo que elas possuam rótulos positivos e negativos, representando assim partições binárias válidas. Também é possível fornecer à população inicial os códigos das abordagens OAA, OAO e ECOC tradicionais.

Para avaliar cada indivíduo, os AGs consideram o poder preditivo do conjunto de classificadores binários, representado em sua matriz de códigos, na solução do problema multiclasses. Um erro em um conjunto de validação é calculado. Classificações desconhecidas, que ocorrem caso mais de uma linha da matriz de códigos apresente distância mínima em relação à cadeia de predições, também foram consideradas como erro. Essa medida de erro deve ser minimizada pelos AGs. O segundo objetivo de ambos AGs formulados é a minimização do número de classificadores binários nas matrizes. A versão lexicográfica favorece a minimização do erro, colocando a redução do número de classificadores binários em segunda ordem de importância. No SPEA2, esse valor foi considerado como um segundo objetivo a ser minimizado usando as relações de dominância de Pareto.

Durante a operação dos AGs, uma matriz de códigos gerada pode possuir colunas equivalentes em sua composição. Para inibir essa ocorrência, cada variante de AG empregou uma estratégia. Na versão lexicográfica, essa característica foi penalizada na função de avaliação dos indivíduos, enquanto que o algoritmo SPEA2 considerou a proporção de colunas equivalentes nos indivíduos como um terceiro objetivo a ser minimizado simultaneamente ao erro de validação e ao número de colunas das matrizes.

A ordenação de objetivos no AG lexicográfico foi realizada nos passos de elitismo e de seleção de indivíduos para reprodução. A cada ocorrência de empate entre duas matrizes, considerando sua aptidão em solucionar o problema multiclasses, o indivíduo com menor número de classificadores era escolhido.

Os AGs foram executados até que um número máximo de gerações fosse atingido. A seleção de indivíduos foi realizada segundo o torneio binário, e os operadores de cruzamento e mutação foram desenvolvidos considerando a representação dos indivíduos e as características do problema de busca de matrizes de códigos formulado. Um tipo de cruzamento empregado, por exemplo, considerou trocas de colunas entre dois indivíduos. Essa operação corresponde à troca de classificadores binários entre duas matrizes, motivado pelo fato de que um classificador binário pode ser mais efetivo em uma combinação multiclasses alternativa. No caso da mutação, empregou-se a alteração de elementos aleatórios nas matrizes e a adição e a remoção de colunas em indivíduos.

Como foram formulados diferentes operadores de cruzamento e mutação, utilizou-se um critério proposto em (Martí et al., 2005) na opção por cada um deles. Nesse caso, cada operador é selecionado probabilisticamente de acordo com o seu desempenho na obtenção de boas matrizes em gerações anteriores. Ou seja, operadores que produzem melhores soluções em gerações prévias têm maiores chances de serem aplicados e sua importância é então adaptada pelos AGs.

A cada execução do SPEA2, um conjunto de soluções é obtido. Para escolher uma solução em particular, foi considerada a distância a um ponto de referência. Esse ponto possui taxa de erro nula, um número mínimo de classificadores binários e não contém colunas equivalentes. Assim, a solução

com avaliações mais próximas a esse ponto é a escolhida.

Os AGs descritos foram avaliados em conjuntos de dados reais (da área de Bioinformática) e de *benchmarks*. Eles foram empregados na busca de matrizes de códigos com desempenho preditivo similar ou superior ao da decomposição OAA usando SVMs como classificadores base. O AG lexicográfico obteve matrizes de códigos com bom desempenho preditivo e com um número menor de classificadores binários que a abordagem OAA. Por outro lado, o SPEA2 não obteve sucesso nesse problema, pois, embora tenha obtido matrizes com menos classificadores binários, as taxas de acerto obtidas por elas não eram comparáveis às da matriz OAA. Além disso, o SPEA2 também encontrou matrizes com colunas equivalentes.

A busca por uma decomposição para um problema multiclasses também pode ser vista como a busca de um comitê (*ensemble*) de classificadores binários para a sua solução. Baseado nisso, Kuncheva (Kuncheva, 2005) propôs o uso de medidas de diversidade da literatura de comitês de classificadores para a geração de matrizes de códigos para problemas multiclasses. Nesse caso, as matrizes consideradas possuíam apenas elementos binários. A autora propôs uma medida conjunta que considera a diversidade das linhas e das colunas de uma dada matriz de códigos. Assim, AGs forma empregados para encontrar matrizes que maximizassem a medida de diversidade formulada. Como no GARA, cada cromossomo foi representado por uma cadeia formada pela concatenação dos códigos da matriz de códigos (Figura 18.11). Somente a mutação foi aplicada como operador genético, implementada por um procedimento de mudança de bits, e um número máximo de iterações foi utilizado como critério de parada. A avaliação experimental considerou a capacidade do AG em otimizar as medidas de diversidade formuladas, mas não foram incluídos experimentos em conjuntos de dados reais ou de *benchmarks*.

Empregando conceitos do trabalho de Kuncheva (Kuncheva, 2005), Shen e Tan (Shen e Tan, 2005) também usaram AGs na busca de matrizes de códigos. Eles adaptaram as medidas de diversidade do trabalho anterior considerando um alfabeto ternário para as matrizes de códigos e as empregaram na formulação de uma medida que considera margens de separação entre os códigos. Assim, procurou-se então maximizar essa margem de separação. A codificação dos cromossomos foi a mesma empregada anteriormente. Previamente ao cálculo da aptidão de um indivíduo, foram removidas as colunas equivalentes ou constantes de sua matriz de códigos. Contudo, o tamanho original l dos códigos foi mantido no cálculo das aptidões, resultando na penalização desse tipo de matriz. O AG pára quando os valores de aptidão não se alteram por um dado período de tempo ou após um número máximo de gerações. Empregou-se cruzamento de um ponto, mutação uniforme e a seleção por roleta. O AG proposto foi avaliado experimentalmente em dois conjuntos de dados para o diagnóstico de câncer. As matrizes de códigos do AG, usando SVMs lineares como classificadores base, foram superiores a outras matrizes de códigos, como OAA e OAO, assim como ao algoritmo k-vizinhos mais próximos (*k-nearest neighbor* - kNN) e Árvores de Decisão (ADs). Foram utilizados $l = \lceil 10 * \log_2(k) \rceil$ classificadores binários.

4. Conclusões

Algoritmos evolutivos têm sido cada vez mais utilizados para diversas finalidades, tanto na computação como em outras áreas de aplicação. Este capítulo descreveu três importantes aplicações de algoritmos genéticos: ajuste de parâmetros dos algoritmos de indução de classificadores; indução de árvores de decisão; e decomposição de problemas multiclasses.

Grande parte dos algoritmos de AM possuem parâmetros cujos valores devem ser especificados pelo usuário. Tais valores influenciam diretamente no desempenho de modelos induzidos. A tarefa de encontrar os melhores valores para esses parâmetros é um problema de otimização, e foi possível observar neste capítulo que os algoritmos genéticos são muito bem aplicados nesse contexto, melhorando o desempenho e robustez dos modelos.

Também foi apresentado o algoritmo genético multi-objetivo para indução de árvores de decisão chamado LEGAL-Tree. LEGAL-Tree apresentou algumas diferenças em relação a outros algoritmos evolutivos para o mesmo propósito, com destaque para o método de geração da população inicial e, principalmente, pela abordagem multi-objetiva proposta na função de *fitness*. Enquanto outros trabalhos lidam apenas com um objetivo ou tratam problemas multi-objetivos por meio das abordagens de fórmula-ponderada ou de Pareto, LEGAL-Tree propôs o uso da abordagem lexicográfica, em que diversos critérios podem ser avaliados de acordo com suas respectivas prioridades para o domínio do problema. Essa abordagem é relativamente simples de implementar e controlar, e também não sofre dos problemas sofridos por fórmula-ponderada e Pareto.

Por fim, esse capítulo também descreveu alguns trabalhos relacionados ao uso de AGs na obtenção de matrizes de códigos para problemas multiclasses. As abordagens ilustradas consideraram: (a) a obtenção de códigos com boas propriedades de correção de erros de classificação; (b) a adaptação dos códigos a cada problema multiclasses considerado; (c) a maximização da diversidade dos classificadores combinados na decomposição.

Referências Bibliográficas

Aarts, E. e Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley and Sons.

Abbass, H. A. (2002). An evolutionary artificial neural networks approach for breast cancer diagnosis, *Artificial Intelligence in Medicine* **25**: 265–281.

Abdinnour-Helm, S. e Hadley, S. (2000). Tabu search based heuristics for multi-floor facility layout, *International Journal of Production Research* **38**: 365–383.

Acid, S. e Campos, L. M. (1996). An algorithm for finding minimum d-separating sets in belief networks, *Proc. 12th Conf. Uncertainty in Artificial Intelligence*, pp. 3–10.

Ada, L. G. e Nossal, G. J. V. (1987). The clonal selection theory, *Scientific American* **257**(2): 50–57.

Adeli, H. e Cheng, N. T. (1994). Augmented Lagrangian genetic algorithm for structural optimization, *Journal of Aerospace Engineering* **7**(1): 104–118.

Adra, S. F. e Fleming, P. J. (2011). Diversity management in evolutionary many-objective optimization, *IEEE Transactions on Evolutionary Computation* **15**(2): 183–195.

Ahmed, R. e Sprent, J. (1999). Immunological memory, *The Immunologist* **7/1-2**: 23–26.

Ahn, B.-H. e Shin, J.-Y. (1991). Vehicle-routeing with time windows and time-varying congestion, *Journal of the Operational Research Society* **42**(5): 393–400.

Ahn, C., Ramakrishna, R. e Goldberg, D. (2006). Real-coded Bayesian optimization algorithm, *in* J. A. Lozano, P. Larrañaga, I. Inza e E. Bengoetxea (eds), *Towards a New Evolutionary Computation*, Springer, pp. 51–73.

Ahn, C. W., Ramakrishna, R. S. e Goldberg, D. E. (2004). Real-coded Bayesian optimization algorithm: Bringing the strength of BOA into the continuous world, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2004)*, pp. 840–851.

Ahuja, R. K., Ergun, O., Orlin, J. B. e Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques, *Discrete Applied Mathematics* **123**: 75–102.

Ahuja, R. K., Orlin, J. B. e Sharma, D. (2001). Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem, *Mathematical Programming* **91**: 71–97.

Aiex, R. M., Pardalos, P. M., Resende, M. G. C. e Toraldo, G. (2005). GRASP with path-relinking for three-index assignment, *INFORMS Journal on Computing* **17**: 224–247.

Aizawa, A. N. e Wah, B. W. (1993). Dynamic control of genetic algorithms in a noisy environment, *Proc. 5th Int. Conf. Genetic Algorithms*, pp. 48–55.

Aizawa, A. N. e Wah, B. W. (1994). Scheduling of genetic algorithms in a noisy environment, *Evolutionary Computation* **2**(2): 97–122.

Akaike, H. A. (1974). A new look at the statistical model identification, *IEEE Transactions on Automatic Control* **AC-19**: 716–723.

Alba, E. e Chicano, J. F. (2004). Solving the error correcting code problem with parallel hybrid heuristics, *Proc. 2004 ACM Symp. Applied Computing*, pp. 985–989.

Alba, E., Cotta, C., Chicano, F. e Nebro, A. J. (2002). Parallel evolutionary algorithms in telecommunications: two case studies, *Proc. Congresso Argentino de Ciências de la Computación*.

Ali, M. M. e Törn, A. (2000). Optimization of carbon and silicon cluster geometry for tersoff potential using differential evolution, *in* C. A. Floudas e P. M. Pardalos (eds), *Optimisation in Computational Chemistry and Molecular Biology*, Kluwer Academic, Dordrecht.

Allahverdi, A., Ng, C. T., Cheng, T. C. E. e Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs, *European Journal of Operational Research* **187**: 985–1032.

Allen, D., Cumano, A., Dildrop, R., Kocks, C., Rajewsky, K., Rajewsky, N., Roes, J., Sablitzky, F. e Siekevitz, M. (1987). Timing, genetic requirements and functional consequences of somatic hypermutation during B-cell development, *Immunology Reviews* **96**: 5–22.

Allwein, E. L., Shapire, R. E. e Singer, Y. (2000). Reducing multiclass to binary: a unifying approach for margin classifiers, *Proc. 17th Int. Conf. Machine Learning*, pp. 9–16.

Alpaydin, E. (2004). *Introduction to Machine Learning: Adaptive Computation and Machine Learning*, The MIT Press.

Alpaydin, E. e Mayoraz, E. (1999). Learning error-correcting output codes from data, *Proc. 9th Int. Conf. Neural Networks*, pp. 743–748.

Alves, M. J., Antunes, C. H. e Clímaco, J. (1997). An experimental comparison of MOLP interactive approaches based on a regional planning model, *Proc. 11th Int. Conf. Multiple Criteria Decision Making*, pp. 428–439.

Andrade, D. V. e Resende, M. G. C. (2006). A GRASP for PBX telephone migration scheduling, *Proc. 8th INFORMS Telecommunications Conf.*

Angelo, J. S. (2008). *Algoritmos baseados em colônia de formigas para otimização multiobjetivo*, Master's thesis, Laboratório Nacional de Computação Científica (LNCC).

Antunes, C. H., Alves, M. J., Silva, A. L. e Clímaco, J. N. (1992). An integrated MOLP method base package – a guided tour of TOMMIX, *Computers and Operations Research* **19**(7): 609–625.

Antunes, C. H., Craveirinha, J. e Clímaco, J. (1993). A multiple criteria model for new telecommunication service planning, *European Journal of Operational Research* **71**(3): 341–352.

Antunes, C. H., Craveirinha, J. e Clímaco, J. (1998). Planning the evolution to broadband access networks - a multicriteria approach, *European Journal of Operational Research* **109**(2): 530–540.

Antunes, C. H., Oliveira, C. e Clímaco, J. (2002). A study of the interactions between the energy system and the economy using TRIMAP, *in* D. Bouyssou (ed.), *Aiding Decisions with Multiple Criteria - Essays in Honor of Bernard Roy*, Kluwer Academic Publishers, pp. 407–427.

Applegate, D., Cook, D. e Rohe, A. (2003). Chained Lin-Kernighan for large traveling salesman problem, *INFORMS Journal on Computing* **15**(1): 82–92.

Araujo, M. C., Wanner, E. F., Guimarães, F. G. e Takahashi, R. H. C. (2009). Constrained optimization based on quadratic approximations in genetic algorithms, *in* E. Mezura-Montes (ed.), *Constraint-Handling in Evolutionary Optimization*, Studies in Computational Intelligence, Springer Berlin, pp. 193–217.

Archetti, C., Speranza, M. G. e Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem, *Transportation Science* **40**: 64–73.

Argoud, A. R. T. T., Gonçalves, E. V. F. e Tiberti, A. J. (2008). Algoritmo genético de agrupamento para formação de módulos de arranjo físico, **15**(2).

Armañanzas, R., Inza, I., Santana, R., Saeys, Y., Flores, J., Lozano, J., Peer, Y., Blanco, R., Robles, V., Bielza, C. e Larrañga, P. (2008). A review of estimation of distribution algorithms in bioinformatics, *BioData Mining* **1**(6): 1–12.

Augusto, D. A. e Barbosa, H. J. C. (2000). Symbolic regression via genetic programming, *Proc. 6th Brazilian Symp. Neural Networks (SBRN 00)*, p. 173.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Inc., NY.

Bäck, T., Fogel, D. e Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press.

Bader, J. M. (2009). *Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods*, PhD thesis, ETH, Zurich, Switzerland.

Bader, J. e Zitzler, E. (2011). HypE: An algorithm for fast hypervolume-based many-objective optimization, *Evolutionary Computation* **19**(1): 45–76.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm, *Proc. Int. Conf. Genetic Algorithms*, Massachusetts, USA, pp. 14–21.

Balas, E. e Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling, *Management Science* **2**(44): 262–275.

Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning, *Technical report*, Carnegie Mellon University, Pittsburgh, PA, USA.

Baluja, S. e Davies, S. (1997). Using optimal dependency-trees for combinational optimization, *Proc. 14th Int. Conf. Machine Learning*, pp. 30–38.

Bandyopadhyay, S., Saha, S., Maulik, U. e Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: AMOSA, *IEEE Transactions on Evolutionary Computation* **12**(3): 269–283.

Barán, B. e Schaerer, M. (2003). A multiobjective ant colony system for vehicle routing problem with time windows, *Proc. 21st IASTED Int. Conf.*, pp. 97–102.

Barone, L. e While, R. L. (2000). Adaptive learning for poker, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2000)*, pp. 566–573.

Basgalupp, M., de Carvalho, A., Barros, R., Ruiz, D. e Freitas, A. (2009). Lexicographic multi-objective evolutionary induction of decision trees, *International Journal of Bio-Inspired Computation* **1**(1/2): 105–117.

Basgalupp, M. P., Barros, R. C., de Carvalho, A. C. P. L. F., Freitas, A. A. e Ruiz, D. D. (2009). LEGAL-Tree: a lexicographic multi-objective genetic algorithm for decision tree induction, *Proc. 2009 ACM Symp. Applied Computing*, pp. 1085–1090.

Basheer, I. A. e Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application, *Journal of Microbiological Methods* **43**: 3–31.

Batista, L. S., Guimarães, F. G. e Ramírez, J. A. (2009). Differential mutation operator for the archive population of multi-objective evolutionary algorithms, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondheim, Noruega, pp. 1108–1115.

Battiti, R. (1996). Reactive search: Toward self-tuning heuristics, *in* V. Rayward-Smith, I. Osman, C. Reeves e G. Smith (eds), *Modern Heuristic Search Methods*, John Wiley & Sons, New York, chapter 4, pp. 61–83.

Battiti, R. e Passerini, A. (2010). Brain-computer evolutionary multiobjective optimization: A genetic algorithm adapting to the decision maker, *IEEE Transactions on Evolutionary Computation* **14**(5): 671–687.

Battiti, R. e Tecchiolli, G. (1994). The reactive tabu search, *ORSA Journal on Computing* **6**(2): 126–140.

Baum, E. B. (1986a). Iterated descent: A better algorithm for local search in combinatorial optimization problems, *Technical report*, California Institute of Technology.

Baum, E. B. (1986b). Towards practical 'neural' computation for combinatorial optimization problems, *AIP Conference Proceedings 151: Neural Networks for Computing*, Woodbury, NY, USA, pp. 53–58.

Baxter, J. (1981). Local optima avoidance in depot location, *Journal of the Operational Research Society* **32**: 815–819.

Bazaraa, M. S., Sherali, H. D. e Shetty, C. M. (1979). *Nonlinear Programming: Theory and Algorithms*, John Wiley and sons, Inc.

Bean, J. C. e Alouane, A. B. (1992). A dual genetic algorithm for bounded integer programs, *Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan* .

Bell, D. e Farquhar, P. (1986). Perspectives of utility theory, *Operations Research* **34**(1): 179–183.

Beltrán, J. D., Calderón, J. E., Cabrera, R. J., Pérez, J. A. M. e Moreno-Vega, J. M. (2004). GRASP/VNS hybrid for the strip packing problem, *Proc. Hybrid Metaheuristics (HM2004)*, pp. 79–90.

Benayoun, R., de Montgolfier, J., Tergny, J. e Larichev, O. (1971). Linear programming with multiple objective functions: step method (STEM), *Mathematical Programming* **1**: 366–375.

Bentley, P. (1999). *Evolutionary design by computers*, Morgan Kaufmann.

Berger, A. (1999). Error-correcting output coding for text classification, *Proc. Workshop on Machine Learning for Information Filtering (IJCAI 1999)*.

Bernardino, H. S., Barbosa, H. J. C., Lemonge, A. C. C. e Fonseca, L. G. (2009). On GA-AIS hybrids for constrained optimization problems in engineering, *in* E. Mezura-Montes (ed.), *Constraint-Handling in Evolutionary Optimization*, Studies in Computational Intelligence, Springer Berlin, pp. 167–192.

Bersini, H. e Varela, F. J. (1990). Hint for adaptive problem solving gleaned from immune networks, *Proc. Int. Conf. Parallel Problem Solving from Nature (PPSN 1990)*, pp. 343–354.

Beume, N., Fonseca, C. M., Lopez-Ibanez, M., Paquete, L. e Vahrenhold, J. (2009). On the complexity of computing the hypervolume indicator, *IEEE Transactions on Evolutionary Computation* **13**(5): 1075–1082.

Beyer, H.-G. (1993). Toward a theory of evolution strategies: Some asymptotical results from the (1,+ λ)-theory, *Evolutionary Computation* **1**(2): 165–188.

Beyer, H.-G. e Schwefel, H.-P. (2002). Evolution strategies - a comprehensive introduction, *Natural Computing* **1**: 3–52.

Bhattacharya, M. e Lu, G. (2003). DAFHEA: a dynamic approximate fitness-based hybrid EA for optimisation problems, *Proc. Congress on Evolutionary Computation (CEC 2003)*, pp. 1879–1886.

Bilchev, G. e Parmee, I. C. (1995). The ant colony metaphor for searching continuous design spaces, *Proc. AISB Workshop on Evolutionary Computation*, pp. 25–39.

Biles, J. A. (2002). GenJam: evolution of a jazz improviser, *Creative Evolutionary Systems*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 165–187.

Billera, L. J., Holmes, S. P. e Vogtmann, K. (2001). Geometry of the space of phylogenetic trees, *Advances in Applied Mathematics* **27**: 733–767.

Birattari, M., Stützle, T. e L. Paquete, K. V. (2002). A racing algorithm for configuring metaheuristics, *Proc. Genetic and Evolutionary Computation Conference (GECCO 2002)*, pp. 11–18.

Blackwell, T. (2007). Particle swarm optimization in dynamic environments, *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer, pp. 29–49.

Blackwell, T. e Branke, J. (2004). Multi-swarm optimization in dynamic environments, *Proc. Applications of Evolutionary Computing (EvoWorkshops 2004)*, pp. 489–500.

Blum, C. (2005). Ant colony optimization: Introduction and recent trends, *Physics of Life Reviews* **2**: 353–373.

Blum, C., Roli, A. e Dorigo, M. (2001). HC-ACO: The hyper-cube framework for ant colony optimization, *4th Metaheuristics Int. Conf. (MIC 2001)*, pp. 399–403.

Boese, K. D., Kahng, A. B. e Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations, *Operations Research Letters* **16**(2): 101–113.

Bolshakova, N. & Azuaje, F. (2002). Cluster validation techniques for genome expression data, Online.
URL: *https://www.cs.tcd.ie/publications/techreports/reports.02/TCD-CS-2002-33.pdf*

Booker, L., Fogel, D., Whitley, D. e Angeline, P. (1997). Recombination, in *Handbook of Evolutionary Computation* (Bäck et al., 1997), pp. C3.3:1–C3.3:27.

Boser, R. C. e Ray-Chaudhuri, D. K. (1960). On a class of error-correcting binary group codes, *Information and Control* **3**: 68–79.

Bosman, P. A. N. e Thierens, D. (2000). Expanding from discrete to continuous estimation of distribution algorithms: The IDEA, *Proc. Int. Conf. Parallel Problem Solving From Nature (PPSN 2000)*, pp. 767–776.

Bot, M. C. J. e Langdon, W. B. (2000). Application of genetic programming to induction of linear classification trees, *Proc. 3rd European Conf. Genetic Programming (EuroGP 2000)*, Edinburgh, pp. 247–258.

Box, G. E. P. e Muller, M. E. (1958). A note on the generation of random normal deviates, *Annals of Mathematical Statistics* **29**: 610–611.

Box, M. (1965). A new method of constrained optimization and a comparison with other methods, *Computation Journal* **8**: 42–52.

Boyd, S. P., Ghaoui, L. E., Feron, E. e Balakrishnan, V. (1997). *Linear Matrix Inequalities in System and Control Theory*, SIAM Studies in Applied Mathematics, SIAM.

Branke, J. (1998). Creating robust solutions by means of evolutionary algorithms, *Proc. 5th Int. Conf. Parallel Problem Solving from Nature (PPSN 1998)*, pp. 119–128.

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pp. 1875–1882.

Branke, J., Kaußler, T., Schmidt, C. e Schmeck, H. (2000). A multi-population approach to dynamic optimization problems, *In Adaptive Computing in Design and Manufacturing*, Springer, pp. 299–308.

Branke, J., Meisel, S. e Schmidt, C. (2008). Simulated annealing in the presence of noise, *Journal of Heuristics* **14**(6): 627–654.

Branke, J. e Schmidt, C. (2003). Selection in the presence of noise, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2003)*, Vol. 2723, pp. 766–777.

Branke, J. e Schmidt, C. (2004). Sequential sampling in noisy environments, *Proc. Int. Conf. Parallel Problem Solving from Nature (PPSN 2004)*, pp. 202–211.

Branke, J., Schmidt, C. e Schmec, H. (2001). Efficient fitness estimation in noisy environments, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2001)*, pp. 243–250.

Braun, H. e Weisbrod, J. (1993). Evolving neural feedforward networks, *Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms*, pp. 25–32.

Bresina, J. (1996). Heuristic-biased stochastic sampling, *Proc. 13th National Conf. Artificial Intelligence*, Portland, pp. 271–278.

Brest, J., Zamuda, A., Boskovic, B., Maucec, M. S. e Zumer, V. (2009). Dynamic optimization using self-adaptive differential evolution, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 415–422.

Brindle, A. (1981). *Genetic Algorithms for Function Optimization*, Phd thesis, University of Alberta.

Bringmann, K. e Friedrich, T. (2009). Approximating the least hypervolume contributor: NP-hard in general, but fast in practice, *Proc. 5th Int. Conf. Evolutionary Multicriterion Optimization (EMO 2009)*, Nantes, France, pp. 6–20.

Broomhead, D. S. e Lowe, D. (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems* **2**: 321–355.

Bullnheimer, B., Hartl, R. F. e Strauss, C. (1997). A new rank-based version of the ant system: A computational study, *Technical report*, Institute of Management Science - University of Viena/ Austria.

Bullnheimer, B., Hartl, R. F. e Strauss, C. (1999). A new rank-based version of the ant system: A computational study, *Central European Journal for Operations Research and Economics* **7**(1): 25–38.

Buntine, W. (1993). Learning classification trees, *in* D. J. Hand (ed.), *Artificial Intelligence Frontiers in Statistics*, Chapman & Hall,London, pp. 182–201.

Burnet, F. M. (1959). *The Clonal Selection Theory of Acquired Immunity*, Cambridge University Press.

Buscher, U. e Shen, L. (2009). An integrated tabu search algorithm for the lot streaming problem in job shops, *European Journal of Operational Research* **199**: 385–399.

Camponogara, E. e Talukdar, S. N. (1997). A genetic algorithm for constrained and multiobjective optimization, *Proc. 3rd Nordic Workshop on Genetic Algorithm and their Applications (3NWGA)*, Vaasa, Finland, pp. 49–62.

Cano, C., Blanco, A., Garcia, F. e Lopez, F. J. (2006). Evolutionary algorithms for finding interpretable patterns in gene expression data, *International Journal on Computer Science and Information System* **1**(2): 88–99.

Cantú-Paz, E. (2002). Feature subset selection by estimation of distribution algorithms, *Proc. Genetic and Evolutionary Computation Conference (GECCO 2002)*, pp. 303–310.

Cantú-Paz, E. (2004). Adaptive sampling for noisy problems, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2004)*, pp. 947–958.

Cantú-Paz, E. e Kamath, C. (2000). Using evolutionary algorithms to induce oblique decision trees, *Proc. Genetic and Evolutionary Computation Conference (GECCO 2000)*, Las Vegas, Nevada, USA, pp. 1053–1060.

Canuto, S., Resende, M. e Ribeiro, C. (2001). Local search with perturbations for the prize-collecting Steiner tree problem in graphs, *Networks* **38**: 50–58.

Capriles, P. V. S. Z., Fonseca, L. G., Barbosa, H. J. C. e Lemonge, A. C. C. (2006). An ant colony algorithm for continuous structural optimization problems, *27th Iberian Latin American Congress in Computational Methods in Engineering (XXVII CILAMCE)*, pp. 531–542.

Capriles, P. V. S. Z., Fonseca, L. G., Barbosa, H. J. C. e Lemonge, A. C. C. (2007). Rank-based ant colony algorithms for truss weight minimization with discrete variables, *Communications in Numerical Methods in Engineering* **23**(6): 553–576.

Cardoso, P., Jesus, M. e Márquez, A. (2003). MONACO - multi-objective network optimisation based on an ACO, *In Proc. of Encuentros de Geometria Computacional*.

Carlisle, A. e Dozier, G. (2000). Adapting particle swarm optimization to dynamic environments, *Proc. Int. Conf. Artificial Intelligence*.

Carrano, E. G. (2007). *Algoritmos evolucionários eficientes para otimização de redes*, PhD thesis, Universidade Federal de Minas Gerais, Brasil.

Carrano, E. G., Fonseca, C. M., Takahashi, R. H. C., Pimenta, L. C. A. e Neto, O. M. (2007). A preliminary comparison of tree encoding schemes for evolutionary algorithms, *Proc. IEEE Int. Conf. System, Man and Cybernetics*, Vancouver, Canada.

Carrano, E. G., Guimaraes, F. G., Takahashi, R. H. C., Neto, O. M. e Campelo, F. (2007). Electric distribution network expansion under load-evolution uncertainty using an immune system inspired algorithm, *IEEE Transactions on Power Systems* **22**: 851–861.

Carrano, E. G., Takahashi, R. H. C., Fonseca, C. M. e Neto, O. M. (2010). Nonlinear network optimization - An embedding vector space approach, *IEEE Transactions on Evolutionary Computation* **14**(2): 206–226.

Carvalho, D. R. e Freitas, A. A. (2004). A hybrid decision tree/genetic algorithm method for data mining, *Information Science* **163**(1-3): 13–35.

Castillo, P. A., Merelo, J. J., Arenas, M. G. e Romero, G. (2007). Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters, *Information Sciences* **177**(14): 2884–2905.

Castro, P. A. D. e Von Zuben, F. J. (2008). MOBAIS: A Bayesian artificial immune system for multi-objective optimization, *Proc. 7th Int. Conf. Artificial Immune Systems (ICARIS 2008)*, pp. 48–59.

Castro, P. A. D. e Von Zuben, F. J. (2009a). BAIS: A Bayesian artificial immune system for the effective handling of building blocks, *Information Sciences* **179**(10): 1426 – 1440.

Castro, P. A. D. e Von Zuben, F. J. (2009b). Multi-objective Bayesian artificial immune system: Empirical evaluation and comparative analyses, *Journal of Mathematical Modelling and Algorithms* **1**: 151–173.

Chakraborty, U. K. (ed.) (2008). *Advances in Differential Evolution*, Studies in Computational Intelligence, Springer.

Chang, C.-C. e Lin, C.-J. (2001). *LIBSVM: a Library for Support Vector Machines*.
URL: *http://www.csie.ntu.edu.tw/∼cjlin/libsvm/*

Chang, W., Yeh, W. e Huang, P. (2009). A hybrid immune-estimation distribution of algorithm for mining thyroid gland data, *Expert Systems with Applications* **37**(3).

Chankong, V. e Haimes, Y. Y. (1983). *Multiobjective Decision Making: Theory and Methodology*, Elsevier.

Chapelle, O., Schölkopf, B. e Zien, A. (2006). *Semi-Supervised Learning*, MIT Press.

Chapelle, O., Vapnik, V., Bousquet, O. e Mukherjee, S. (2002). Choosing multiple parameters for support vector machines, *Machine Learning* **46**(1-3): 131–159.

Charon, I. e Hudry, O. (1993). The noising method: A new method for combinatorial optimization, *Operations Research Letters* **14**: 133–137.

Charon, I. e Hudry, O. (2002). The noising methods: A survey, *in* C. Ribeiro e C. Ribeiro (eds), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, pp. 245–261.

Chen, J., Goldberg, D. E., Sastry, K. e Ho, S. (2002). Fitness inheritance in multi-objective optimization, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2002)*, pp. 319–326.

Chen, W., Shi, Y. e Teng, H. (2008). A generalized differential evolution combined with EDA for multi-objective optimization problems, *Proc. 4th Int. Conf. Intelligent Computing (ICIC 2008)*, pp. 140–147.

Chow, C. J. e Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees, *IEEE Transations on Information Theory* **14**(3): 462–467.

Christodoulakis, M., Iliopoulos, C. S., Park, K. e Sim, J. S. (2005). Implementing approximate regularities, *Mathematical and Computer Modelling* **42**: 855–866.

Clímaco, J. e Antunes, C. H. (1987). TRIMAP – an interactive tricriteria linear programming package, *Foundations of Control Engineering* **12**: 101–119.

Clímaco, J. e Antunes, C. H. (1989). Implementation of an user friendly software package - a guided tour of TRIMAP, *Mathematical and Computer Modelling* **12**: 1299–1309.

Clímaco, J. e Antunes, C. H. (1990). A comparison of microcomputer implemented interactive MOLP methods based on a case study, *in* C. B. e Costa (ed.), *Readings in Multiple Criteria Decision Aid*, Springer-Verlag, pp. 445–470.

Clímaco, J., Antunes, C. H. e Alves, M. J. (1997). From TRIMAP to SOMMIX - building effective interactive MOLP computational tools, *Proc. 12th Int. Conf. Multiple Criteria Decision Making*, pp. 285–296.

Clímaco, J., Antunes, C. H. e Alves, M. J. (2003). Programação linear multiobjectivo – do modelo de programação linear clássico à consideração explícita de várias funções objectivo, Imprensa da Universidade de Coimbra.

Clímaco, J., Antunes, C. H., Martins, A. e Almeida, A. (1995). A multiple objective linear programming model for power generation expansion planning, *Energy Research* **19**: 419–432.

Cobb, H. G. (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments, *Technical Report AIC-90-001*, Naval Research Laboratory, Washington, USA.

Codenotti, B., Manzini, G., Margara, L. e Resta, G. (1996). Perturbation: An efficient technique for the solution of very large instances of the TSP, *INFORMS Journal on Computing* **8**(2): 125–133.

Coelho, L. d. S., Souza, R. C. T. e Mariani, V. C. (2009). Improved differential evolution approach based on cultural algorithm and diversity measure applied to solve economic load dispatch problems, *Mathematics and Computers in Simulation* **79**(10): 3136 – 3147.

Coello Coello, C. A. (1999). A comprehensive survey of evolutionary-based multiobjective optimization technique, *Knowledge Information Systems* **3**: 269–308.

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art, *Computer Methods in Applied Mechanics and Engineering* **191**: 1254–1287.

Coello Coello, C. A. e Mezura-Montes, E. (2002). Handling constraints in genetic algorithms using dominance-based tournament, *Proc. 5th Int. Conf. Adaptative Computing Design Manufacture*, 5, Devon, UK, pp. 273–284.

Coello Coello, C. A., Van Veldhuizen, D. A. e Lamont, G. B. (2002). *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer.

Cohen, W. W. (1995). Fast effective rule induction, *Proc. 12th Int. Conf. on Machine Learning*, pp. 115–123.

Corana, A., Marchesi, M., Martini, C. e Ridella, S. (1987). Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm, *ACM Transactions on Mathematical Software* **13**(3): 262–280.

Cordeau, J. F., Gendreau, M., Laporte, G., Potvin, J. Y. e Semet, F. (2002). A guide to vehicle routing problem, *Journal of the Operational Research Society* **53**: 512–522.

Cordón, O., de Vianna, I. F., Herrera, F. e Moreno, L. (2000). A new ACO model integrating evolutionary computation concepts: The best-worst ant system, *Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*, IRIDIA – Université Libre de Bruxelles/ Belgium, pp. 22–29.

Costa, D. e Silver, E. A. (1998). Tabu search when noise is present: An illustration in the context of cause and effect analysis, *Journal of Heuristics* **4**(1): 5–23.

Costa, E. e Simoes, A. (2008). *Inteligência Artificial: Fundamentos e Aplicações (2a edição)*, FCA - Editora de Informática.

Cotta, C., Alba, E., Sagarna, R. e Larrañaga, P. (2001). Adjusting weights in artificial neural networks using evolutionary algorithms, *in* P. Larrañaga e J. Lozano (eds), *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, chapter 18, pp. 357–373.

Cover, T. M. e Hart, P. E. (1967). Nearest neighbor pattern classification, *IEEE Transactions on Information Theory* **13**: 21–27.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L. e Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*, Springer.

Cristianini, N. e Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and other Kernel-Based Learning Methods*, Cambridge University Press, New York, NY, USA.

Cutello, V. & Nicosia, G. (2004). *Recent Developments in Biologically Inspired Computing*, Idea Group Pub., chapter VI, pp. 104–146.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals, and Systems* **2**(4): 303–314.

Czyzak, P. e Jaszkiewicz, A. (1998). Pareto simulated annealing: A metaheuristic technique for multiple-objective combinatorial optimization, *Journal of Multi-Criteria Decision Analysis* **7**(1): 34–47.

Darwen, P. J. (2000). Computationally intensive and noisy tasks: Co-evolutionary learning and temporal difference learning on backgammon, *Proc. IEEE Congress on Evolutionary Computation (CEC 2000)*, pp. 872–879.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, John Murray.

Das, I. (2000). Robustness optimization for constrained nonlinear programming problem, *Engineering Optimization* **32**(5): 585–618.

Dasgupta, D. e Michalewicz, Z. (1997). *Evolutionary Algorithms in Engineering Applications*, Springer.

Davis, L. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, NY.

De Bonet, J. S., Isbell, C. L. e Viola Jr., P. (1997). MIMIC: Finding optima by estimating probability densities, *Advances in Neural Information Processing Systems*, p. 424.

de Castro, L. N. (2001). *Engenharia Imunológica: Desenvolvimento e Aplicação de Ferramentas Computacionais Inspiradas em Sistemas Imunológicos Artificiais*, PhD thesis, DCA FEEC/UNICAMP, Campinas/SP, Brazil.

de Castro, L. N. (2006). *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*, Chapman & Hall/CRC.

de Castro, L. N. (2007). Fundamentals of natural computing: An overview, *Physics of Life Reviews* **4(1)**: 1–36.

de Castro, L. N. e Timmis, J. I. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag.

de Castro, L. N. e Von Zuben, F. J. (2001). aiNet: An artificial immune network for data analysis, *in* R. A. S. In Hussein A. Abbass e C. S. Newton (eds), *Data Mining: A Heuristic Approach*, Idea Group Publishing, USA, pp. 231–259.

de Castro, L. N. e Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle, *IEEE Transactions on Evolutionary Computation* **6(3)**: 239-251.

de Castro, L. N., Von Zuben, F. J. e Knidel, H. (eds) (2007). *Proc. 6th Int. Conf. Artificial Immune Systems*, Springer, Santos, Brazil.

de França, F., Gomes, L. C. T., de Castro, L. N. e Von Zuben, F. J. (2006). Handling time-varying TSP instances, *Proc. IEEE Congress on Evolutionary Computation (CEC 2006)*, pp. 2830–2837.

de França, F. O. e Von Zuben, F. J. (2009). A dynamic artificial immune algorithm applied to challenging benchmarking problems, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 423–430.

de França, F. O., Von Zuben, F. J. e de Castro, L. N. (2005). An artificial immune network for multimodal function optimization on dynamic environments, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2005)*, pp. 289–296.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD thesis, University of Michigan.

De Jong, K. A. (1993). Genetic algorithms are NOT function optimizers, *Proc. Workshop on Foundations of Genetic Algorithms (FOGA 1993)*, pp. 5–17.

de la Penã, M. (2004). Heuristics and metaheuristics approaches used to solve the rural postman problem: A comparative case study, *Proc. 4th Int. ICSC Symp. Engineering of Intelligent Systems (EIS 2004)*.

de Souza, M., Duhamel, C. e Ribeiro, C. (2003). A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy, *in* M. G. C. Resende e J. P. de Sousa (eds), *Metaheuristics: Computer Decision-Making*, Kluwer Academic Publishers, pp. 627–658.

de Werra, D. e Hertz, A. (1989). Tabu search techniques: A tutorial and an application to neural networks, *OR Spektrum* **11**: 131–141.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms, *Computer Methods in Applied Mechanics and Engineering* **186**: 311–338.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, Inc., New York, NY, USA.

Deb, K. (2008). Scope of stationary multi-objective evolutionary optimization: a case study on a hydro-thermal power dispatch problem, *Journal of Global Optimization* **41**(4): 479–515.

Deb, K. e Goldberg, D. E. (1992). Analysing deception in trap functions, *Proc. 2nd Workshop on Foundations of Genetic Algorithms (FOGA 1992)*, Vail, Colorado, USA, pp. 93–108.

Deb, K. e Gupta, H. (2005). Searching for robust Pareto-optimal solutions in multi-objective optimization, *Proc. 3rd Int. Conf. Evolutionary Multi-Criterion Optimization (EMO 2005)*, Guanajuato, México, pp. 150–164.

Deb, K., Gupta, S., Daum, D., Branke, J., Mall, A. e Padmanabhan, D. (2009). Reliability-based optimization using evolutionary algorithms, *IEEE Transactions on Evolutionary Computation* **13**(5).

Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, *Proc. 6th Int. Conf. Parallel Problem Solving from Nature (PPSN 2000)*.

Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transations on Evolutionary Computation* **6**(2): 182–197.

Deb, K., Pratap, A. e Meyarivan, T. (2001). Constrained test problems for multi-objective evolutionary optimization, *Proc. 1st Int. Conf. Evolutionary Multi-Criterion Optimization (EMO 2001)*, pp. 284–298.

Deb, K., Sinha, A., Korhonen, P. J. e Wallenius, J. (2010). An interactive evolutionary multiobjective optimization method based on progressively approximated value functions, *IEEE Transactions on Evolutionary Computation* **14**(5): 723–739.

Dekel, O. e Singer, Y. (2003). Multiclass learning by probabilistic embeddings, *Advances in Neural Information Processing Systems*, pp. 945–952.

delaOssa, L., Gamez, J. A. e Puerta, J. M. (2009). Learning weighted linguistic fuzzy rules by using specifically-tailored hybrid estimation of distribution algorithms, *International Journal of Approximation Reasoning* **50**(3): 541–560.

Dellaert, N., Jeunet, J. e Jonard, N. (2000). A genetic algorithm to solve the general multi-level lot-sizing problem with time-varying costs, *International Journal of Production Economics* **68**(3): 241–257.

Delmaire, H., Díaz, J., Fernández, E. e Ortega, M. (1999). Reactive GRASP and Tabu Search based heuristics for the single source capacitated plant location problem, *INFOR* **37**: 194–225.

Deneubourg, J. L., Aron, S., Goss, S. e Pasteels, J. M. (1990). The self-organizing exploratory pattern of the argentine ant, *Journal of Insect Behavior* **3**(2): 159–162.

Deza, M. e Huang, T. (1998). Metrics on permutations, a survey, *Journal of Combinatorics, Information and System Science* **23**: 173–185.

Di Caro, G. e Dorigo, M. (1997). AntNet: A mobile agents approach to adaptive routing, *Technical Report 97–12*, IRIDIA - Université Libre de Bruxelles/ Belgium.

Díaz, J. A. e Fernández (2001). A tabu search heuristic for the generalized assignment problem, *European Journal of Operational Research* **132**: 22–38.

Dietterich, T. G. e Bariki, G. (1995). Solving multiclass learning problems via error-correcting output codes, *Journal of Artificial Intelligence Research* **2**: 263–286.

Dodd, N. (1990). Optimisation of network structure using genetic techniques, *Proc. Int. Joint Conf. Neural Networks (IJCNN 1990)*, pp. 965–970.

Dontas, K. e De Jong, K. A. (1990). Discovery of maximal distance codes using genetic algorithms, *Proc. 2nd IEEE Int. Conf. Tools for Artificial Intelligence*, pp. 905–811.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms (in Italian)*, PhD thesis, Dipartimento di Elettronica e Informazione - Politecnico di Milano/ Italia.

Dorigo, M., Birattari, M. e Stützle, T. (2006). Ant colony optimization: Artificial ants as a computational intelligence technique, *IEEE Computational Intelligence Magazine* **11**: 28–39.

Dorigo, M. e Gambardella, L. M. (1997a). Ant colonies for the traveling salesman problem, *IEEE Transactions on Evolutionary Computation,* **1**(1): 53–66.

Dorigo, M. e Gambardella, L. M. (1997b). Ant Colony System: A cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* **1**(1): 53–66.

Dorigo, M., Maniezzo, V. e Colomi, A. (1996). The ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics: Part B* **26**(1): 29–41.

Dorigo, M., Maniezzo, V. e Colorni, A. (1991). Positive feedback as a search strategy, *Technical Report 91–012*, Dipartimento di Elettronica e Informazione - Politecnico di Milano/ Italia.

Dorigo, M. e Stützle, T. (2004). *Ant Colony Optimization*, MIT Press.

Dréo, J. e Siarry, P. (2004). Continuous interacting ant colony algorithm based on dense heterarchy, *Future Generation Computer Systems* **20**(5): 841–856.

Drummond, L., Vianna, L. S., Silva, M. B. e Ochi, L. S. (2002). Distributed parallel metaheuristics based on GRASP and VNS for solving the traveling purchaser problem, *Proc. 9th Int. Conf. Parallel and Distributed Systems (ICPADS 2002)*, pp. 257–266.

Duarte, A. R., Ribeiro, C. C., Urrutia, S. e Haeusler, E. H. (2006). Referee assignment in sports leagues, *Proc. 6th Int. Conf. Practice and Theory of Automated Timetabling (PATAT 2006)* pp. 158–173.

Duarte, A., Ribeiro, C. C. e Urrutia, S. (2007). A hybrid ILS heuristic to the referee assignment problem with an embedded MIP strategy, *Proc. 4th Int. Conf. Hybrid Metaheuristics (MH 2007)*, pp. 82–95.

Ducheyne, E. I., De Baets, B. e De Wulf, R. R. (2008). Fitness inheritance in multiple objective evolutionary algorithms: A test bench and real-world evaluation, *Applied Soft Computing* **8**(1): 337–349.

Ehrgott, M. (2000). *Multicriteria Optimization*, Springer, Berlin, Germany.

Ehrgott, M. e Gandibleux, X. (2000). A survey and annotated bibliography of multiobjective combinatorial optimization, *OR Spectrum* **22**(4): 425–460.

Eiben, A. e Smith, J. (2003). *Introduction to Evolutionary Computing*, Springer.

El-Beltagy, M. A. e Keane, A. J. (1999). Metamodeling techniques for evolutionary optimization of computationally expensive problems: promises and limitations, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 1999)*, Orlando, USA, pp. 196–203.

Emmerich, M., Beume, N. e Naujoks, B. (2005). An EMO algorithm using the hypervolume measure as selection criterion, *Proc. 3rd Int. Conf. Evolutionary Multicriterion Optimization (EMO 2005)*, pp. 62–76.

Emmerich, M. T. M. e Fonseca, C. M. (2011). Computing hypervolume contributions in low dimensions: Asymptotically optimal algorithm and complexity results, *Proc. 6th Int. Conf. Evolutionary Multicriterion Optimization (EMO 2011)*, Ouro Preto, Brazil, pp. 121–135.

Engquist, B. e Runborg, O. (2002). Wavelet-based numerical homogenization with applications, *Lecture Notes in Computational Science and Engineering*, Springer, pp. 97–148.

Eshelman, L. J. e Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata, *Proc. Workshop on Foundations of Genetic Algorithms (FOGA 1993)*, San Mateo, pp. 187–202.

Esponda, F., Forrest, S. e Helman, P. (2004). A formal framework for positive and negative detection schemes, *IEEE Transations on Systems, Man and Cybernetics* **34(1)**: 357-373.

Estrada-Gil, J. K., Fernandez-Lopez, J. C., Hernandez-Lemus, E., Silva-Zolezzi, I., Hidalgo-Miranda, A., Jimenez-Sanchez, G. e Vallejo-Clemente, E. E. (2007). GPDTI: A genetic programming decision tree induction method to find epistatic effects in common complex diseases, *Bioinformatics* **23**(13): i167–174.

Everitt, B. S., Landau, S. e Leese, M. (2001). *Cluster Analysis*, Arnold, London.

Fahlman, S. E. (1988). Faster-learning variations on back-propagation: An empirical study, *Proc. 1988 Connectionist Models Summer School*, Los Altos, CA.

Fan, H.-Y., Dulikravich, G. S. e Han, Z.-X. (2005). Aerodynamic data modeling using support vector machines, *Inverse Problems in Science and Engineering* **13**(3): 261–278.

Farina, M., Deb, K. e Amato, P. (2004). Dynamic multiobjective optimization problems: test cases, approximations, and applications, *IEEE Transactions on Evolutionary Computation* **8**(5): 425–442.

Farmer, J. D., Kauffman, S. A., Packard, N. H. e Perelson, A. S. (1987). Adaptive dynamic networks as models for the immune system and autocatalytic sets, *Annals of the New York Academy of Sciences* **504**: 118–131.

Farmer, J. D., Packard, N. H. e Perelson, A. S. (1986). The immune system, adaptation, and machine learning, *Physica* **22D**: 187-204.

Feo, T. e Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters* **8**: 67–71.

Feo, T. e Resende, M. (1995). Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6**: 109–133.

Feoktistov, V. (2006). *Differential Evolution: In Search of Solutions*, Springer.

Fernandes, C. M., Lima, C. e Rosa, A. C. (2008). UMDAs for dynamic optimization problems, *Proc. 10th Annual Conf. Genetic and Evolutionary Computation (GECCO 2008)*, New York, NY, USA, pp. 399–406.

Fernandez-Marquez, Luis, J., Arcos e Lluis, J. (2009). An evaporation mechanism for dynamic and noisy multimodal optimization, *Proc. 11th Annual Conf. Genetic and Evolutionary Computation (GECCO 2009)*, pp. 17–24.

Festa, P., Pardalos, P., Pitsoulis, L. e Resende, M. (2006). GRASP with path-relinking for the weighted MAXSAT problem, *ACM Journal of Experimental Algorithmics* **11**. article 2.4: 1-16.

Festa, P., Pardalos, P., Resende, M. e Ribeiro, C. (2002). Randomized heuristics for the MAX-CUT problem, *Optimization Methods and Software* **7**: 1033–1058.

Festa, P. e Resende, M. (2002). GRASP: An annotated bibliography, *in* C. Ribeiro e P. Hansen (eds), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, pp. 325–367.

Festa, P. e Resende, M. (2009a). An annotated bibliography of GRASP, Part I: Algorithms, *International Transactions in Operational Research* **16**: 1–24.

Festa, P. e Resende, M. (2009b). An annotated bibliography of GRASP, Part II: Applications, *International Transactions in Operational Research* **16**: 131–172.

Fitzpatrick, J. M. e Grefenstette, J. J. (1988). Genetic algorithms in noisy environments, *Machine Learning* **3**(2-3): 101–120.

Fleischmann, B., Gnutzmann, S. e Sandvoβ, E. (2004). Dynamic vehicle routing based on online traffic information, *Transportation Science* **38**(4): 420–433.

Fleming, P. J. e Purshouse, R. (2002). Evolutionary algorithms in control systems engineering: a survey, *Control Engineering Practice* **10**: 1223–1241.

Fleurent, C. e Glover, F. (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory, *INFORMS Journal on Computing* **11**: 198–204.

Floreano, D. e Nolfi, S. (2004). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, Bradford Books.

Fogel, L., Owens, A. e Walsh, M. (1966). *Artificial Intelligence through Simulated Evolution*, John Wiley, New York.

Fonseca, C. M. e Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization, *Proc. 5th Int. Conf. Genetic Algorithms*.

Fonseca, C. M. e Fleming, P. J. (1995). An overview of evolutionary algorithms in multiobjective optimization, *Evolutionary Computation* **3**(1): 1–16.

Forouraghi, B. (2000). A genetic algorithm for multiobjective robust design, *Applied Intelligence* **12**(3): 151–161.

Forrest, S., Javornik, B., Smith, R. E. e Perelson, A. S. (1993). Using genetic algorithms to explore pattern recognition in the immune system, *Evolutionary Computation* **1(3)**: 191–211.

Forrest, S., Perelson, A., Allen, L. e Cherukuri, R. (1994). Self-nonself discrimination in a computer, *Proc. IEEE Symp. Research in Security and Privacy*, pp. 202–212.

França, P. M. (2009). Busca tabu. Disponível em http://www.densis.fee.unicamp.br/ franca/EA043/TranspaCap-4a.pdf.

Freitas, A. A. (2004). A critical review of multi-objective optimization in data mining: a position paper, *SIGKDD Exploration Newsletter* **6**(2): 77–86.

Freitas, A. A., Wieser, D. C. e Apweiler, R. (2008). On the importance of comprehensible classification models for protein function prediction, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **7**(1): 172–182.

Freund, Y. e Mason, L. (1999). The alternating decision tree learning algorithm, *Proc. 16th Int. Conf. Machine Learning*, pp. 124–133.

Galeano, J. C., V.-S. A. e González, F. A. (2005). A comparative analysis of artificial immune network models, *Proc. 6th Int. Conf. Genetic Algorithms*, pp. 344–350.

Galic, E. e Höhfeld, M. (1996). Improving the generalization performance of multi-layer-perceptrons with population-based incremental learning, *Proc. 4th Int. Conf. Parallel Problem Solving from Nature (PPSN 1996)*, London, UK, pp. 740–750.

Galton, A. (1999). Two topologies for discrete space, *Proc. Leeds Spatial Reasoning Workshop*, Leeds, UK.

Galton, A. (2000). Continuous motion in discrete spaces, *Proc. Int. Conf. Principles of Knowledge Representation and Reasoning*, San Francisco, USA.

Galton, A. (2003). A generalized topological view of motion in discrete space, *Theoretical Computer Science* **305**: 114–134.

Gambardella, L. M. e Dorigo, M. (1995). Ant–Q: A reinformcement learning approach to the traveling salesman problem, *Proc. 12th Int. Conf. Machinge Learning*, pp. 252–260.

Gamperle, G., Mueller, S. D. e Koumoutsakos, P. (2002). A parameter study for differential evolution, *Proc. of the WSEAS Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, Athens, pp. 293–298.

Gao, L., Zhou, C., Gao, H.-B. e Shi, Y.-R. (2006). Credit scoring model based on neural network with particle swarm optimization, *Proc. 2nd Int. Conf. Advances in Natural Computation*, pp. 76–79.

García-Martínez, C., Cordón, O. e Herrera, F. (2007). A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP, *European Journal of Operational Research* **180**: 116–148.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, USA.

Gaspar-Cunha, A. (2000). *Modeling and Optimisation of Single Screw Extrusion*, PhD thesis, University of Minho, Guimarães, Portugal.

Gaspar-Cunha, A. e Covas, J. A. (2004). RPSGAe - a multiobjective genetic algorithm with elitism: Application to polymer extrusion, *in* X. Gandibleux, M. Sevaux, K. Sörensen e V. T'kindt (eds), *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems, Springer, pp. 221–249.

Gaspar-Cunha, A. e Vieira, A. S. (2004). A hybrid multi-objective evolutionary algorithm using an inverse neural network, *Hybrid Metaheuristics (HM 2004) Workshop at ECAI 2004*, pp. 25–30.

Gendreau, M. (2002). Recent advances in tabu search, *in* C. C. Ribeiro e P. Hansen (eds), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, pp. 369–377.

Gendreau, M. (2003). An introduction to tabu search, *in* F. Glover e G. A. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, chapter 2, pp. 37–54.

Gendreau, M., Guertin, F., Potvin, J.-Y. e Taillard, E. D. (1999). Parallel tabu search for real-time vehicle routing and dispatching, *Transportation Science* **33**: 381–390.

Gendreau, M., Hertz, A. e Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem, *Management Science* **40**: 1276–1290.

Gendreau, M., Laporte, G. e Séguin, R. (1996). A tabu search heuristic for the vehicle routing problem with stochastic demands and customers, *Operations Research* **44**: 469–477.

Gendreau, M., Soriano, P. e Salvail, L. (1993). Solving the maximum clique problem using a tabu search approach, *Annals of Operations Research* **41**(4): 385–403.

Geng, Y., Li, Y. e Lim, A. (2005). A very large-scale neighborhood search approach to capacitated warehouse routing problem, *Proc. 17th IEEE Int. Conf. Tools with Artificial Intelligence (ICTAI 2005)*.

Geoffrion, A. (1968). Proper efficiency and the theory of vector maximization, *Journal of Mathematical Analysis and Applications* **22**: 618–630.

Ghani, R. (2000). Using error correcting output codes for text classification, *Proc. 17th Int. Conf. Machine Learning*, pp. 303–310.

Giel, O. e Lehre, P. K. (2010). On the effect of populations in evolutionary multi-objective optimisation, *Evolutionary Computation* **18**(3): 335–356.

Gilks, W. R., Richardson, S. e Spiegelhalter, D. (1995). *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*, 1 edn, Chapman & Hall/CRC.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints, *Decision Sciences* **8**: 156–166.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research* **5**: 553–549.

Glover, F. (1989). Tabu search – part I, *ORSA Journal of Computing* **1**(3): 190–206.

Glover, F. (1990). Tabu search – part II, *ORSA Journal of Computing* **2**(1): 4–32.

Glover, F. (1996). Tabu search and adaptive memory programing – Advances, applications and challenges, *in* R. Barr, R. Helgason e J. Kennington (eds), *Interfaces in Computer Science and Operations Research*, Kluwer, pp. 1–75.

Glover, F. (2000). Multi-start and strategic oscillation methods – Principles to exploit adaptive memory, *in* M. Laguna e J. Gonzáles-Velarde (eds), *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, Kluwer, pp. 1–24.

Glover, F. e Laguna, M. (1993). Tabu search, *in* C. R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Advanced Topics in Computer Science Series, Blackwell Scientific Publications, London, chapter 3, pp. 70–150.

Glover, F. e Laguna, M. (1997). *Tabu Search*, Kluwer Academic Publishers, Boston.

Glover, F., Laguna, M. e Martí, R. (2000). Fundamentals of scatter search and path relinking, *Control and Cybernetics* **29**(3): 652–684.

Glover, F., Taillard, E. e de Werra, D. (1993). A user's guide to tabu search, *Annals of Operations Research* **41**(1): 1–28.

Glover, F. W. e Kochenberger, G. A. (2003). *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*, Springer.

Goemans, M. e Williamson, D. (1996). The primal dual method for approximation algorithms and its application to network design problems, *in* D. Hochbaum (ed.), *Approximation algorithms for NP-hard problems*, PWS Publishing Co., pp. 144–191.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

Goldberg, D. E. e Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms, *Proc. Workshop Foundations of Genetic Algorithms (FOGA 1991)*, pp. 69–93.

Goldberg, D. E., Korb, G. e Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results, *Complex Systems* **3**(5): 493–530.

Goldberg, D. E. e Smith, R. E. (1987). Nonstationary function optimization using genetic algorithm with dominance and diploidy, *Proc. 2nd Int. Conf. Genetic Algorithms and their Application (ICGA 1987)*, pp. 59–68.

Gong, W. e Cai, Z. (2008). A multiobjective differential evolution algorithm for constrained optimization, *Proc. IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 181–188.

Goss, S., Aron, S., Deneubourg, J. L. e Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant, *Naturvissenschaften* **76**: 579–581.

Grassé, P. P. (1959). La reconstruction du nid et les coordinations inter-individuelles chez *bellicosi-termes natalensis* et *cubitermes sp.*. la theorie de la stigmergie: Essai d'interpretation des termites constructeurs, *Insectes Sociaux* **6**(1): 41–80.

Greenwood, P. E. e Nikulin, M. S. (1996). *A guide to chi-squared testing*, Wiley-Interscience.

Grefenstette, J. (1992). Genetic algorithms for changing environments, *Proc. Int. Conf. Parallel Problem Solving from Nature (PPSN 2)*, pp. 137–144.

Grünwald, P. (2000). Model selection based on minimum description length, *Journal of Mathematical Psychology* **44**: 133–152.

Guntsch, M. e Middendorf, M. (2001). Pheromone modification strategies for ant algorithms applied to dynamic TSP, *Proc. EvoWorkshops on Applications of Evolutionary Computing*, pp. 213–222.

Guntsch, M., Middendorf, M. e Schmeck, H. (2001). An ant colony optimization approach to dynamic TSP, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2001)*, pp. 860–867.

Guo, Q., Zhang, H. e Iliopoulos, C. S. (2008). Generalized approximate regularities in strings, *International Journal of Computer Mathematics* **85**: 155–168.

Gutjahr, W. J. (2003). A converging ACO algorithm for stochastic combinatorial optimization, *Proc. 2nd Int. Symp. Stochastic Algorithms: Foundations and Applications (SAGA 2003)*, pp. 10–25.

Gutjahr, W. J., Pflug, G. C. e Ug, G. C. P. (1996). Simulated annealing for noisy cost functions, *Journal of Global Optimization* **8**: 1–13.

Gwiazda, T. D. (2006). *Genetic Algorithms Reference: crossover for single-objective numerical optimization problems*, Vol. 1, Tomasz Gwiazda.

Hadad, B. S. e Eick, C. F. (1997). Supporting polyploidy in genetic algorithms using dominance vectors, *Proc. 6th Int. Conf. Evolutionary Programming (EP 1997)*, pp. 223–234.

Hamming, R. W. (1950). Error detecting and error correcting codes, *Bell System Technology Journal* **29**: 147–160.

Hamming, R. W. (1980). *Coding and Information Theory*, Prentice Hall.

Han, J. e Kamber, M. (2000). *Data Mining: Concepts and Techniques*, Morgan Kaufmann.

Hansen, N. e Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies, *Evolutionary Computation* **9**(2): 159–195.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming, *Proc. Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

Hansen, P., Jaumard, J. e Mladenović, N. (2000). Variable neighborhood search for maximum weight satisfiability problem, *Technical report*, Le cahiers du GERAD, Group for Research in Decision Analysis.

Hansen, P. e Mladenović, N. (1997). Variable neighborhood search for the p-median, *Location Science* **5**: 207–226.

Hansen, P. e Mladenović, N. (1999). An introduction to variable neighbourhood search, *in* S. Voss, S. Martello, I. Osman e C. Roucairol (eds), *Metaheuristics: Advances and Trends in Local Search Procedures for Optimization*, Kluwer, pp. 433–458.

Hansen, P. e Mladenović, N. (2001). J-means: A new local search heuristic for minimum sum-of-squares clustering, *Pattern Recognition* **34**: 405–413.

Harik, G. R., Lobo, F. G. e Goldberg, D. E. (1999). The compact genetic algorithm, *IEEE Transations on Evolutionary Computation* **3**(4): 287–297.

Harik, G. R., Lobo, F. G. e Sastry, K. (2006). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA), *in* M. Pelikan, K. Sastry e E. Cantú-Paz (eds), *Scalable Optimization via Probabilistic Modeling*, Springer, pp. 39–61.

Hart, J. e Shogan, A. (1987). Semi-greedy heuristics: An empirical study, *Operations Research Letters* **6**: 107–114.

Hastie, T. e Tibshirani, R. (1998). Classification by pairwise coupling, *The Annals of Statistics* **2**: 451–471.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ. 2nd edition.

Heckerman, D., Geiger, D. e Chickering, D. (1995). Learning Bayesian networks: The combination of knowledge and statistical data, *Machine Learning* **20**: 197–243.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic, *European Journal of Operational Research* **126**(1): 106–130.

Hendtlass, T. (2007). Fitness estimation and the particle swarm optimisation algorithm, *Proc. IEEE Congress on Evolutionary Computation (CEC 2007)*, pp. 4266–4272.

Henrion, M. (1986). Propagating uncertainty in Bayesian networks by probabilistic logic sampling, *Proc. Annual Conf. Uncertainty in Artificial Intelligence (UAI 1986)*, pp. 149–164.

Herrera, F., Herrera-Viedma, E., Lozano, M. e Verdegay, J. L. (1994). Fuzzy tools to improve genetic algorithms, *Proc. 2nd European Congress on Intelligent Techniques and Soft Computing*, pp. 1532–1539.

Herrera, F., Lozano, M. e Verdegay, J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis, *Artificial Intelligence Review* **12**(4): 265–319.

Hertz, A. e de Werra, D. (1990). The tabu search metaheuristic: how we used it, *Annals of Mathematics and Artificial Intelligence* **1**: 111–121.

Hightower, R. R., Forrest, S. e Perelson, A. S. (1995). The evolution of emergent organization in immune system gene libraries, *Proc. 6th Int. Conf. Genetic Algorithms*, pp. 344–350.

Hinterding, R. (1999). Representation, constraint satisfaction and the knapsack problem, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pp. 1286–1292.

Hintz, K. J. e Spofford, J. J. (1990). Evolving a neural network, *Proc. IEEE Int. Symp. Intelligent Control*, pp. 479–484.

Hodgkin, P. D. (1998). Role of cross-reactivity in the develpment of antibody responses, *The Immunologist* **6**: 223–226.

Hoffmann, G. W. (1986). A neural network model based on the analogy with the immune system, *Journal of Theoretical Biology* **122**: 33–67.

Hofmeyr, S. A. (1997). An overview of the immune system, Online.
URL: *http://www.cs.unm.edu/ steveah/imm-html/immune-system.html.*

Hofmeyr, S. A. e Forrest, S. (2000). Architecture for an artificial immune system, *Evolutionary Computation* **7(1)**: 45–68.

Holland, J. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions, *Evolutionary Computation* **8**(4): 373–391.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, Massachusetts.

Homaifar, A., Lai, S. H. Y. e Qi, X. (1994). Constrained optimization via genetic algorithms, *Simulation* **62**(4): 242–254.

Hoos, H. e Stützle, T. (2004). *Stochastic Local Search – Foundations and Applications*, Morgan Kaufmann Publishers, San Francisco, CA.

Hruschka, E. R., de Castro, L. N. e Campello, R. J. G. B. (2004). Evolutionary algorithms for clustering gene-expression data, *Proc. 4th IEEE Int. Conf. Data Mining (ICDM 2004)*, pp. 403–406.

Hsu, C.-W. e Lin, C.-J. (2002). A comparison of methods for multi-class support vector machines, *IEEE Transations on Neural Networks* **13**(2): 415–425.

Hu, T. C. (1974). Optimum communication spanning trees, *SIAM Journal of Computing* **3**: 188–195.

Hu, X., Huang, Z. e Wang, Z. (2003). Hybridization of the multi-objective evolutionary algorithms and the gradient-based algorithms, *Proc. IEEE Congress on Evolutionary Computation (CEC 2003)*, pp. 870–877.

Huang, C.-L. e Wang, C.-J. (2006). A GA-based feature selection and parameters optimization for support vector machines, *Expert Systems with Applications* **31**(2): 231–240.

Hüsken, M., Jin, Y. e Sendhoff, B. (2005). Structure optimization of neural networks for evolutionary design optimization, *Soft Computing Journal* **9**(1): 21–28.

Hutter, F. e Hamadi, Y. (2005). Parameter adjustment based on performance prediction: Towards an instance-aware problem solver, *Technical report*, Microsoft Research, Redmond, WA.

Ilonen, J., Kamarainen, J.-K. e Lampinen, J. A. (2003). Differential evolution training algorithm for feed-forward neural networks, *Neural Processing Letters* **17**(1): 93–105.

Imbault, F. e Lebart, K. (2004). A stochastic optimization approach for parameter tuning of support vector machines, *Proc. 17th Int. Conf. on Pattern Recognition*, pp. 597–600.

Ingolfsson, A., Amanul Haque, M. e Umnikov, A. (2002). Accounting for time-varying queueing effects in workforce scheduling, *European Journal of Operational Research* **139**(3): 585–597.

Inza, I., Larrañaga, P., Etxeberria, R. e Sierra, B. (2000). Feature subset selection by Bayesian network-based optimization, *Artificial Intelligence* **123**: 157–184.

Iredi, S., Merkle, D. e Middendorf, M. (2001). Bi-criterion optimization with multi colony ant algorithms, *Proc. 1st Int. Conf. Evolutionary Multi-Criterion Optimization (EMO 2001)*, pp. 359–372.

Ishibuchi, H. e Murata, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling, *IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews* **28**(3): 392–403.

Ishibuchi, H., Tsukamoto, N., Sakane, Y. e Nojima, Y. (2010). Indicator-based evolutionary algorithm with hypervolume approximation by achievement scalarizing functions, *Proc. 12th Annual Genetic and Evolutionary Computation Conf. (GECCO 2010)*, Portland, USA, pp. 527–534.

Jacob, C., Pilat, M. L., Timmis, J. e Bentley, P. J. (eds) (2005). *Proc. 4th Int. Conf. Artificial Immune Systems*, Vol. 3627, Springer.

Jain, A. K. e Dubes, R. C. (1988). *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ.

Jain, A., Murty, M. N. e Flynn, P. J. (1999). Data clustering: A review, *ACM Computing Surveys* **31**(3): 264–323.

Jaisankar, S. e Raghurama Rao, S. V. (2007). Diffusion regulation for Euler solvers, *Journal of Computer Physics* **221**(2): 577–599.

Janeway, C. A. (1992). The immune system evolved to discriminate infectious nonself from noninfectious self, *Immunology Today* **13(1)**: 11–16.

Janeway, C. A., Travers, P., Walport, M. e Capra, J. D. (2000). *Immunobiology: The Immune System in Health and Disease*, Garland Publishing.

Jaros, J. e Schwarz, J. (2007). Parallel BMDA with probability model migration, *Proc. IEEE Congress on Evolutionary Computation (CEC 2007)*, pp. 1059–1066.

Jaskowski, W. e Kotlowski, W. (2008). On selecting the best individual in noisy environments, *Proc. 10th Annual Genetic and Evolutionary Computation Conf. (GECCO 2008)*, New York, NY, USA, pp. 961–968.

Jatmiko, W., Sekiyama, K. e Fukuda, T. (2006). A PSO-based mobile sensor network for odor source localization in dynamic environment: Theory, simulation and measurement, *Proc. IEEE Congress on Evolutionary Computation (CEC 2006)*, pp. 1036–1043.

Jensen, F. V. (1996). *An Introduction to Bayesian Networks*, UCL Press, London.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Jensen, M. (2004). Generating robust and flexible job shop schedules using genetic algorithms, *IEEE Transactions on Evolutionary Computation* **7**(3): 275–288.

Jeong, S. e Murayama, M. (2004). Efficient optimization design method using kriging model, *42nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, USA.

Jerne, N. K. (1974). Towards a network theory of the immune system, *Annals of Immunology* **125C**: 373–389.

Jiménez, F., Verdegay, J. e Gomeéz-Skarmeta, A. F. (1999). Evolutionary techniques for constrained multiobjective optimization problems, *Proc. Workshop on Multi-Criterion Optimization Using Evolutionary Methods held at Genetic and Evolutionary Computation Conference (GECCO -1999)*, ACM, pp. 115–116.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing Journal* **9**(1): 3–12.

Jin, Y. e Branke, J. (2005). Evolutionary optimization in uncertain environments – a survey, *IEEE Transactions on Evolutionary Computation* **9**(3): 303–317.

Jin, Y. e Sendhoff, B. (2003). Tradeoff between performance and robustness: An evolutionary multiobjective approach, *Proc. 2nd Int. Conf. Evolutionary Multi-Criterion Optimization (EMO 2003)*, pp. 237–251.

Jin, Y. e Sendhoff, B. (2004). Reducing fitness evaluations using clustering techniques and neural network ensembles, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2004)*, pp. 688–699.

Johanson, B. e Poli, R. (1998). GP-music: An interactive genetic programming system for music generation with automated fitness raters, *Proc. 3rd Annual Conf. Genetic Programming (GP 1998)*, pp. 181–186.

Johnson, D. (1990). Local optimization and the traveling salesman problem, *Proc. 17th Colloquium on Automata*, pp. 446–461.

Johnson, D. S. e McGeoch, L. A. (1997). The travelling salesman problem: A case study in local optimization, *in* E. H. L. Aarts e J. K. Lenstra (eds), *Local Search in Combinatorial Optimization*, John Wiley & Sons, Chichester, UK, pp. 215–310.

Johnson, D. S., Papadimitriou, C. e Yannakakis, M. (1988). How easy is local search?, *Journal of Computer and System Sciences* **37**(1): 79–100.

Johnston, R. (2004). *Applications of Evolutionary Computation in Chemistry*, Structure and Bonding, Springer.

Joines, J. e Houck, C. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs, *Proc. 1st IEEE Int. Conf. Evolutionary Computation (ICEC 1994)*, Orlando, FL, pp. 579–584.

Jonhson, J. M. e Rahmat-Samii, Y. (1997). Genetic algorithms in engineering electromagnetics, *IEEE Antennas and Propagation Magazine* **39**(4): 7–21.

Junqueira, C., de França, F., Attux, R., Panazio, C., de Castro, L., Von Zuben, F. e Romano, J. (2006). Immune-inspired dynamic optimization for blind spatial equalization in undermodeled channels, *Proc. IEEE Congress on Evolutionary Computation (CEC 2006)*, pp. 2896–2903.

Junqueira, C., de França, F. O., Attux, R. R. F., Suyama, R., de Castro, L. N., Von Zuben, F. J. e Romano, J. M. T. (2005). A proposal for blind FIR equalization of time-varying channels, *Proc. IEEE Workshop on Machine Learning for Signal Processing*, pp. 9–14.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems, *Transactions of the ASME–Journal of Basic Engineering* **82**(Series D): 35–45.

Karahan, I. e Koksalan, M. (2010). A territory defining multiobjective evolutionary algorithms and preference incorporation, *IEEE Transactions on Evolutionary Computation* **14**(4): 636–664.

Kasanen, E., Ostermark, R. e Zeleny, M. (1991). Gestalt system of holistic graphics: new management support view of MCDM, *Computers and Operations Research* **18**: 233–239.

Kaufman, L. e Rousseeuw, P. J. (1990). *Finding groups in data: an introduction to cluster analysis*, John Wiley & Sons, New York.

Kay, S. M. (1993). *Fundamentals of statistical signal processing : Estimation theory*, Signal Processing Series, Prentice Hall.

Keeney, R. e Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Values Tradeoffs*, Wiley, New York, USA.

Kelsey, J. e Timmis, J. (2003). Immune inspired somatic contiguous hypermutation for function optimisation, *Proc. Genetic and Evolutionary Computation Conference (GECCO 2003)* pp. 207–218.

Kennedy, J. (1997). The particle swarm: social adaptation of knowledge, *Proc. IEEE Int. Conf. Evolutionary Computation (ICEC 1997)*, Indianapolis, USA, pp. 303–308.

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization, *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, pp. 1942–1948.

Kepler, T. B. e Perelson, A. S. (1993). Somatic hypermutation in B cells: An optimal control treatment, *Journal of Theoretical Biology* **164**: 37-64.

Khan, N., Goldberg, D. E. e Pelikan, M. (2002). Multi-objective Bayesian optimization algorithm, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2002)*, p. 684.

Kikuchi, R. (1951). A theory of cooperative phenomena, *Physical Review* **81**(6): 988.

Kim, H.-K., Park, J.-K. C. K.-Y. e Lowther, D. A. (2007). Differential evolution strategy for constrained global optimization and application to practical engineering problems, *IEEE Transactions on Magnetics* **43**(4): 1565–1568.

Kim, H.-S. e Cho, S.-B. (2001). An efficient genetic algorithm with less fitness evaluation by clustering, *Proc. IEEE Congress on Evolutionary Computation (CEC 2001)*, pp. 887–894.

Kim, J.-H., Han, J.-H., Kim, Y.-H., Choi, S.-H. e Kim, E.-S. (2011). Preference-based solution selection algorithm for evolutionary multiobjective optimization, *IEEE Transactions on Evolutionary Computation* . DOI: 10.1109/TEVC.2010.2098412.

Kirkpatrick Jr., S., Gellat, C. D. e Vechi, Z. M. P. (1983). Optimization by simulated annealing, *Science* **220**: 671–680.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies, *Journal of Statistical Physics* **34**(5–6): 975–986.

Kirkpatrick, S., Gelatt, Jr., C. e Vecchi, M. (1983). Optimization by simulated annealing, *Science* **220**(4598): 671–680.

Kirkpatrick, S., Gelatt-Junior., C. D. e Vecchi, M. (1982). Optimizations by simulated annealing, *Technical Report RC 9355*, IBM.

Klee, V. e Minty, G. (1972). How good is the simplex algorithm?, *in* O. Shisha (ed.), *Inequalities III*, Academic Press, New York, pp. 159–175.

Klein, J. e Horejsi, V. (1990). *Immunology*, Blackwell Scientific Publications.

Knerr, S., Personnaz, L. e Dreyfus, G. (1992). Handwritten digit recognition by neural networks with single-layer training, *IEEE Transactions on Neural Networks* **3**(6): 962–968.

Knowles, J. e Corne, D. (1999). The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pp. 105–110.

Knowles, J. e Corne, D. (2004). Memetic algorithms for multiobjective optimization: issues, methods and prospects, *in* W. E. Hart, N. Krasnogor e J. E. Smith (eds), *Recent Advances in Memetic Algorithms*, Vol. 166 of *Studies in Fuzziness and Soft Computing*, Springer, pp. 313–352.

Knowles, J. D. e Corne, D. W. (2000a). Approximating the nondominated front using the Pareto archived evolution strategy, *Evolutionary Computation* **8**(2): 149–172.

Knowles, J. D. e Corne, D. W. (2000b). M-PAES: a memetic algorithm for multiobjective optimization, *Proc. IEEE Congress on Evolutionary Computation (CEC 2000)*, pp. 325–332.

Kohavi, R. e John, G. H. (1995). Automatic parameter selection by minimizing estimated error, *Proc. 12th Int. Conf. Machine Learning*, San Francisco, CA, pp. 304–312.

Kohavi, R. e John, G. H. (1997). Wrappers for feature subset selection, *Artificial Intelligence* **97**(1-2): 273–324.

Kong, M. e Tian, P. (2006). A direct application of ant colony optimization to function optimization problem in continuous domain, *Proc. 5th Int. Workshop Ant Colony, Optimization and Swarm Intelligence (ANTS 2006)*, pp. 324–331.

Korhonen, P. (1987). VIG - a visual interactive support system for multiple criteria decision making, *Belgian Journal of Operations Research* **27**(1): 3–15.

Korhonen, P. e Laakso, J. (1986a). Solving generalized goal programming problems using a visual interactive approach, *European Journal of Operational Research* **26**(3): 355–363.

Korhonen, P. e Laakso, J. (1986b). A visual interactive method for solving the multiple criteria problem, *European Journal of Operational Research* **24**(2): 277–287.

Korhonen, P. e Wallenius, J. (1988). A Pareto race, *Naval Research Logistics* **35**: 615–623.

Korosec, P. e Silc, J. (2009). The differential ant-stigmergy algorithm applied to dynamic optimization problems, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 407–414.

Koza, J. R. (1991). Concept formation and decision tree induction using the genetic programming paradigm, *Proc. 1st Int. Conf. Parallel Problem Solving from Nature (PPSN 1991)*, pp. 124–128.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA.

Koziel, S. e Michalewicz, Z. (1998). A decoder-based evolutionary algorithm for constrained parameter optimization problems, *Proc. 5th Int. Conf. Parallel Problem Solving from Nature (PPSN 1998)*, Springer, Amsterdam, pp. 231–240.

Koziel, S. e Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings and constrained parameter optimization, *Evolutionary Computation* **7**(1): 19–44.

Kretowski, M. (2004). An evolutionary algorithm for oblique decision tree induction, *Proc. Int. Conf. Artificial Intelligence and Soft Computing (ICAISC 2004)*, pp. 432–437.

Kretowski, M. e Grzes, M. (2005). Global learning of decision trees by an evolutionary algorithm, *Information Processing and Security Systems*, Springer, pp. 401–410.

Krusienski, D. e Jenkins, W. (2004). The application of particle swarm optimization to adaptive IIR phase equalization, *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP 2004)*, pp. 693–696.

Kuhn, H. W. (2004). The Hungarian method for the assignment problem, *Naval Research Logistics* **52**: 7–21.

Kumar, A., Keane, A. J., Nair, P. B. e Shahpar, S. (2006). Robust design of compressor fan blades against erosion, *ASME Journal of Mechanical Design* **128**(4): 864–873.

Kuncheva, L. I. (2005). Using diversity measures for generating error-correcting output codes in classifier ensembles, *Pattern Recognition Letters* **26**: 83–90.

Kuncheva, L. I. e Hadjitodorov, S. T. (2004). Using diversity in cluster ensembles, *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pp. 1214–1219.

Lafage, C. e Lang, J. (2005). Propositional distances and compact preference representation, *European Journal of Operational Research* **160**: 741–761.

Laguna, M. (1994). A guide to implement tabu search, *Investigación Operativa* **4**: 5–25.

Laguna, M. e González-Velarde, J. (1991). A search heuristic for just-in-time scheduling in parallel machines, *Journal of Intelligent Manufacturing* **2**: 253–260.

Laguna, M. e Martí, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization, *INFORMS Journal on Computing* **11**: 44–52.

Lam, W. e Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the MDL principle, *Computational Intelligence* **10**(4): 269–293.

Lampinen, J. A. (2002). A constraint handling approach for the differential evolution algorithm, *Proc. IEEE Congress on Evolutionary Computation (CEC 2002)*, Honolulu, Hawaii, pp. 1468–1473.

Lampinen, J. A. e Zelinka, I. (1999). Mechanical engineering design optimization by differential evolution, *in* D. Corne, M. Dorigo e F. Glover (eds), *New Ideas in Optimisation*, Advanced Topics in Computer Science, McGraw-Hill Inc., London, UK, pp. 127–146.

Lara, A., Sanchez, G., Coello-Coello, C. A. e Schutze, O. (2010). HCS: A new local search strategy for memetic multiobjective evolutionary algorithms, *IEEE Transactions on Evolutionary Computation* **14**(1): 112–132.

Larrañaga, P., Etxeberria, R., Lozano, J. A. e Peña, J. M. (2000a). Combinatorial optimization by learning and simulation of Bayesian networks, *Proc. 16th Conf. Uncertainty in Artificial Intelligence*, pp. 343–352.

Larrañaga, P., Etxeberria, R., Lozano, J. A. e Peña, J. M. (2000b). Optimization in continuous domains by learning and simulation of Gaussian networks, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2000), Workshop in Optimization by Building and Using Probabilistic Models*, pp. 201–204.

Larrañaga, P. e Lozano, J. (2002). *Estimation of Distribution Algorithms: A new tool for evolutionary computation*, Kluwer Academic Pub.

Laumanns, M. e Ocenasek, J. (2002). Bayesian optimization algorithms for multi-objective optimization, *Proc. Int. Conf. Parallel Problem Solving From Nature (PPSN 2002)*, pp. 298–307.

LeCun, Y. A., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Müller, E., Säckinger, E., Simard, P. Y. e Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition, *Proc. Int. Conf. Artificial Neural Networks (ICANN 1995)*, Nanterre, France, pp. 53–60.

Lee, D. S., Gonzalez, L. F., Srinivas, K. e Periaux, J. (2008). Robust evolutionary algorithms for UAV/UCAV aerodynamic and RCS design optimisation, *Computers & Fluids* **37**(5): 547–564.

Lemonge, A. C. C. e Barbosa, H. J. C. (2004). An adaptive penalty scheme for genetic algorithms in structural optimization, *International Journal for Numerical Methods in Engineering* **59**(5): 703–736.

Leon, J. V., Wu, D. S. e Storer, R. H. (1994). Robustness measures and robust scheduling for job shops, *IIE Transactions* **26**(5): 32–43.

Leung, F. H. F., Lam, H. K., Ling, S. H. e Tam, P. K. S. (2003). Tuning of the structure and parameters of a neural network using an improved genetic algorithm, *IEEE Transactions on Neural Networks* **14**(1): 79–88.

Lewandowski, A. e Wierzbicki, A. (1988). Aspiration based decision analysis and support. part i: Theoretical and methodological backgrounds, *Working paper WP-88-03*, IIASA, Laxenburg, Austria.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems, *OR Spectrum* **30**: 167–190.

Li, C. e Yang, S. (2009). A clustering particle swarm optimizer for dynamic optimization, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 439–446.

Li, H. e Zhang, Q. (2008). Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, *IEEE Transations on Evolutionary Computation* **13**(2): 284–302.

Li, H., Zhang, Q., Tsang, E. e Ford, J. (2004). Hybrid estimation of distribution algorithm for multiobjective knapsack problem, *Proc. 4th European Conf. Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004)*, pp. 145–154.

Li, X., Branke, J. e Blackwell, T. (2006). Particle swarm with speciation and adaptation in a dynamic environment, *Proc. 8th Annual Genetic and Evolutionary Computation Conference (GECCO 2006)*, pp. 51–58.

Li, Y. J. e Wu, T. J. (2003). An adaptive ant colony system algorithm for continuous-space optimization problems, *Journal of Zhejiang University Science* **4**(1): 40–46.

Li, Z., Guo, S., Wang, F. e Lim, A. (2004). Improved GRASP with tabu search for vehicle routing with both time window and limited number of vehicles, *Proc. 17th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2004)*, pp. 552–561.

Liang, J. e Suganthan, P. (2005). Dynamic multi-swarm particle swarm optimizer, *Proc. IEEE Swarm Intelligence Symposium (SIS 2005)*, pp. 124–129.

Liang, K.-H., Yao, X. e Newton, C. (1999). Combining landscape approximation and local search in global optimization, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pp. 1514–1520.

Liang, K.-H., Yao, X. e Newton, C. (2000). Evolutionary search of approximated n-dimensional landscapes, *International Journal of Knowledge-based Intelligent Engineering Systems* **4**: 172–183.

Lim, A. e Wang, F. (2004). A smoothed dynamic tabu search embedded GRASP for m-VRPTW, *Proc. 16th IEEE Int. Conf. Tools with Artificial Intelligence (ICTAI 2004)*, pp. 704–708.

Lima, E. L. (1995). *Curso de Análise*, Projeto Euclides - IMPA, Rio de Janeiro.

Lin, S. e Kerninghan, B. (1973). An effective heuristic algorithm for the traveling salesman problem, *Operations Research* (21): 498–516.

Lippmann, R. P. (1987). An introduction to computing with neural networks, *IEEE Acoustics, Speech and Signal Processing Magazine* **4**(2): 4–22.

Liu, X., Pardalos, P., Rajasekaran, S. e Resende, M. (2000). A GRASP for frequency assignment in mobile radio networks, *in* B. Badrinath, F. Hsu, P. Pardalos e S. Rajasejaran (eds), *Mobile Networks and Computing*, Vol. 52 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, pp. 195–201.

Liu, X., Wu, Y. e Ye, J. (2008). An improved estimation of distribution algorithm in dynamic environments, *Proc. 4th Int. Conf. Natural Computation (ICNC 2008)*, Los Alamitos, CA, USA, pp. 269–272.

Lobo, F. G., Lima, C. F. e Michalewicz, Z. (eds) (2007). *Parameter Setting in Evolutionary Algorithms*, Vol. 54 of *Studies in Computational Intelligence*, Springer.

Lopez-Jaimes, A., Coello-Coello, C. A., Aguirre, H. e Tanaka, K. (2011). Adaptive objective space partitioning using conflict information for many-objective optimization, *Proc. 6th Int. Conf. Evolutionary Multicriterion Optimization (EMO 2011)*, Ouro Preto, Brazil, pp. 151–165.

Lorena, A. C. (2006). *Investigação de Estratégias para a Geração de Máquinas de Vetores de Suporte Multiclasses*, PhD thesis, ICMC-USP. Disponível em: `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-26052006-111406/`.

Lorena, A. C. e Carvalho, A. C. P. L. F. (2006). Multiclass SVM design and parameter selection with genetic algorithms, *Proc. 9th Brazilian Symp. Neural Networks*, p. 23.

Lorena, A. C. e Carvalho, A. C. P. L. F. (2007a). Design of directed acyclic graph multiclass structures, *Neural Network World* **17**(6): 657–673.

Lorena, A. C. e Carvalho, A. C. P. L. F. (2007b). Evolutionary design of multiclass support vector machines, *Journal of Intelligent and Fuzzy Systems* **8**(5): 445–454.

Lorena, A. C. e Carvalho, A. C. P. L. F. (2008). Estratégias para a combinação de classificadores binários em soluções multiclasses, *Revista de Informática Teórica e Aplicada* **XV**: 65–86.

Loshchilov, I., Schoenauer, M. e Sebag, M. (2010). A Pareto-compliant surrogate approach for multiobjective optimization, *Proc. 12th Annual Genetic and Evolutionary Computation Conf. (GECCO 2010)*, Portland, USA, pp. 1979–1982.

Loughlin, D. H. e Ranjithan, S. R. (1999). Chance-constrained genetic algorithms, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 1999)*, Orlando, Florida, USA, pp. 369–376.

Louis, S. e Xu, Z. (1996). Genetic algorithms for open shop scheduling and re-scheduling, *Proc. 11th Int. Conf. Computers and their Applications (ISCA 1996)*, pp. 99–102.

Lourenço, H. R., Martin, O. e Stützle, T. (2002). Iterated local search, *in* F. Glover e G. Kochenberger (eds), *Handbook of Metaheuristics*, Vol. 57 of *International Series in Operations Research & Management Science*, Kluwer Academic Publishers, Norwell, MA, pp. 321–353.

Lourenço, H. R. e Zwijnenburg, M. (1996). Combining the larg-step optimization with tabu search: Application to the job-shop scheduling problem, *in* J.P.Kelly (ed.), *Meta-heuristics: Theory & Applications*, Kluwer Academic Publishers, Norwell, MA, pp. 219–236.

Loveard, T. e Ciesielski, V. (2001). Representing classification problems in genetic programming, *Proc. IEEE Congress on Evolutionary Computation (CEC 2001)*, pp. 1070–1077.

Loveard, T. e Ciesielski, V. (2002). Employing nominal attributes in classification using genetic programming, *Proc. 4th Asia-Pacific Conf. Simulated Evolution and Learning (SEAL 2002)*, pp. 487–491.

Loyer, B. e Jézéquel, L. (2009). Robust design of a passive linear quarter car suspension system using a multi-objective evolutionary algorithm and analytical robustness indexes, *Vehicle System Dynamics* **47**(10): 1253–1270.

Lozano, M., Herrera, F., Krasnogor, N. e Molina, D. (2004). Real-coded memetic algorithms with crossover hill-climbing, *Evolutionary Computation* **12**(3): 273–302.

Luenberger, D. G. (1969). *Optimization by Vector Space Methods*, John Willey & Sons.

Luenberger, D. G. (1984). *Linear and Nonlinear Programming*, Addison-Wesley.

Luo, B. e Zheng, J. (2008). A new methodology for searching robust Pareto optimal solutions with MOEAs, *Proc. IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 580–586.

Machwe, A. e Parmee, I. (2007). Towards an interactive, generative design system: Integrating a build and evolve approach with machine learning for complex freeform design, *Proc. EvoWorkshops 2007*, pp. 449–458.

Magoulas, G. D., Plagianakos, V. P. e Vrahatis, M. N. (2001). Hybrid methods using evolutionary algorithms for on-line training, *Proc. Int. Joint Conf. Neural Networks (IJCNN 2001)*, Washington, DC, USA, pp. 2218–2223.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, *INFORMS Journal on Computing* **11**(4): 358–369.

Maniezzo, V. e Carbonaro, A. (2000). An ANTS heuristic for the frequency assignment problem, *Future Generation Computer Systems* **16**: 927–935.

Markon, S., Markon, O., Arnold, D. V., Bäck, T., Beyer, H.-G. e Beielstein, T. (2001). Thresholding - a selection operator for noisy ES, *Proc. IEEE Congress on Evolutionary Computation (CEC 2001)*, pp. 465–472.

Martí, L., García, J., Berlanga, A. e Molina, J. M. (2008). Model building algorithms for multiobjective EDAs: Directions for improvement, *Proc. IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 2848–2855.

Martí, R., Laguna, M. e Campos, V. (2005). Scatter search vs. genetic algorithms: An experimental evaluation with permutation problems, *in* C. Rego e B. Alidaee (eds), *Metaheuristic Optimization Via Adaptive Memory and Evolution: Tabu Search and Scatter Search*, Kluwer Academic Publishers, pp. 263–282.

Martin, O. e Otto, S. W. (1996). Combining simulated annealing with local search heuristics, *Annals of Operations Research* **63**: 57–75.

Martin, O., Otto, S. W. e Felten, E. W. (1991a). Large-step Markov chains for the traveling salesman problem, *Complex Systems* **5**: 299–326.

Martin, O., Otto, S. W. e Felten, E. W. (1991b). Large-step Markov chains for the traveling salesman problem, *INFORMS Journal on Computing* **8**(1): 1–15.

Martins, A. G., Coelho, D., Antunes, C. H. e Clímaco, J. (1996). A multiple objective linear programming approach to power generation planning with demand-side management (DSM), *International Transactions in Operational Research* **3**(3-4): 305–317.

Martins, S., Pardalos, P., Resende, M. e Ribeiro, C. (1999). Greedy randomized adaptive search procedures for the Steiner problem in graphs, *in* P. Pardalos, S. Rajasejaran e J. Rolim (eds), *Randomization Methods in Algorithmic Design*, Vol. 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, pp. 133–145.

Mason, D. (1998). Antigen cross-reactivity: Essential in the function of TCRs, *The Immunologist* **6**: 220–222.

Masters, T. e Land, W. (1997). A new training algorithm for the general regression neural network, *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pp. 1990–1994.

Mateus, G., Resende, M. e Silva, R. (2009). GRASP with path-relinking for the generalized quadratic assignment problem, *Technical report*, AT&T Labs Research, Shannon Laboratory, Florham Park, New Jersey.

Matzinger, P. (1994). Tolerance, danger and the extended family, *Annual Reviews of Immunology* **12**: 991–1045.

Matzinger, P. (2002). The danger model: A renewed sense of self, *Science* **296**(5566): 301–305.

Mayoraz, E. e Moreira, M. (1997). On the decomposition of polychotomies into dichotomies, *Proc. 14th Int. Conf. Machine Learning (ICML 1997)*, Nashville, TN, pp. 219–226.

McKay, M. D., Beckman, R. J. e Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, *Technometrics* **42**(1): 55–61.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. e Teller, E. (1953). Equation of state calculations by fast computing machines, *Journal of Chemical Physics* **21**: 1087–1092.

Mezura-Montes, E. e Coello Coello, C. A. (2005). A simple multimembered evolution strategy to solve constrained optimization problems, *IEEE Transactions on Evolutionary Computation* **9**(1): 1–17.

Mezura-Montes, E., Velazquez-Reyes, J. e Coello Coello, C. A. (2006). A comparative study of differential evolution variants for global optimization, *Proc. 8th Annual Genetic and Evolutionary Computation Conf. (GECCO 2006)*, pp. 485–492.

Mühlenbein, H. e Mahnig, T. (1999). The factorized distribution algorithm for additively decomposed functions, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pp. 752–759.

Mühlenbein, H. e Paass, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters, *Proc. 4th Int. Conf. Parallel Problem Solving from Nature (PPSN 1996)*, London, UK, pp. 178–187.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer.

Michalewicz, Z. (1995a). Genetic algorithms, numerical optimization and constraints, *Proc. 6th Int. Conf. Genetic Algorithms*, San Francisco, CA, pp. 151–158.

Michalewicz, Z. (1995b). A survey of constraint handling techniques in evolutionary computation methods, *Proc. 4th Annual Conf. Evolutionary Programming*, pp. 135–155.

Michalewicz, Z. e Attia, N. F. (1994). Evolutionary optimization of constrained problems, *Proc. 3rd Annual Conf. Evolutionary Programming*, Singapore, pp. 98–108.

Michalewicz, Z. e Janikow, C. (1991). Handling constraints in genetic algorithms, *Proc. 4th Int. Conf. Genetic Algorithms*, Los Altos, CA, pp. 151–157.

Michalewicz, Z. e Schoenauer, M. (1996a). Evolutionary algorithms for constrained parameter optimization problem, *Evolutionary Computation* **4**(1): 1–32.

Michalewicz, Z. e Schoenauer, M. (1996b). Evolutionary computation at the edge of feasibility, *Proc. 4th Int. Conf. Parallel Problem Solving from Nature (PPSN 1996)*, pp. 245–254.

Michalski, K. A. (2001). Electromagnetic imaging of elliptical-cylindrical conductors and tunnels using a differential evolution algorithm, *Microwave and Optical Technology Letters* **28**(3): 164–169.

Miller, B. L. (1997). Noise, sampling, and efficient genetic algorithms, *Technical report*, University of Illinois at Urbana-Champaign.

Miller, B. L., Miller, B. L., Goldberg, D. E. e Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise, *Evolutionary Computation* **4**: 113–131.

Miller, G. F., Todd, P. M. e Hegde, S. U. (1989). Designing neural networks using genetic algorithms, *Proc. 3rd Int. Conf. Genetic Algorithms*, San Francisco, CA, USA, pp. 379–384.

Miller, J. F. e Smith, S. L. (2006). Redundancy and computational efficiency in Cartesian genetic programming, *IEEE Transactions on Evolutionary Computation* **10**(2): 167–174.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA.

Mitchell, T. M. (1982). Generalization as search, *Artificial Intelligence* **18**: 203–266.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search, *Computers & Operations Research* **24**: 1097–1100.

Moioli, R. C., Vargas, P. A. e Husbands, P. (2009). A multiple hormone approach to the homeostatic control of conflicting behaviours in an autonomous mobile robot, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 47–54.

Monard, M. C. e Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina, *in* S. O. Rezende (ed.), *Sistemas Inteligentes: Fundamentos e Aplicações*, Editora Manole Ltda, chapter 4, pp. 89–114.

Monmarché, N., Venturini, G. e Slimane, M. (2000). On how *Pachycondyla apicalis* ants suggest a new search algorithm, *Future Generation Computer Systems* **16**: 937–946.

Montana, D. J. (1994). Strongly typed genetic programming, *Technical Report #7866*, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA.

Moore, R. (1966). *Dimensional Analysis*, Prentice Hall.

Moraglio, A., Kim, Y., Yoon, Y. e Moon, B. (2007). Geometric crossovers for multiway graph partitioning, *Evolutionary Computation* **14**: 445–474.

Moraglio, A. e Poli, R. (2004). Topological interpretation of crossover, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2004)*, Seattle, USA.

Moraglio, A. e Poli, R. (2005). Topological crossover for the permutation representation, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2005)*, Washington, USA.

Moraglio, A., Poli, R. e Seehuus, R. (2006). Geometrical crossover for biological sequences, *European Conf. Genetic Programming*, Budapest, Hungary.

Moraglio, A. e Togelius, J. (2009). Inertial geometric particle swarm optimization, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 1973–1980.

Moreira, M. e Mayoraz, E. (1998). Improved pairwise coupling classification with correcting classifiers, *Proc. 10th European Conf. Machine Learning*, London, UK, pp. 160–171.

Mori, N., Kita, H. e Nishikawa, Y. (1996). Adaptation to a changing environment by means of the thermodynamical genetic algorithm, *Proc. 4th Int. Conf. Parallel Problem Solving from Nature (PPSN 1996)*, pp. 513–522.

Mori, N., Kita, H. e Nishikawa, Y. (1998). Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm, *Proc. 5th Int. Conf. Parallel Problem Solving from Nature (PPSN 1998)*, London, UK, pp. 149–158.

Moroni, A., Von Zuben, F. J. e Manzolli, J. (2002). ArTbitration: Human-Machine Interaction in Artistic Domains, *Leonardo* **35**(2): 185–188.

Morrison, R. e De Jong, K. (1999). Memory enhanced evolutionary algorithms for changing optimization problems, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, pp. 2047–2053.

Moscato, P. (1999). Memetic algorithms: A short introduction, *in* D. Corne, M. Dorigo e F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, pp. 219–234.

Moura, A. e Oliveira, J. (2005). A GRASP approach to the container-loading problem, *IEEE Intelligent Systems* **20**: 50–57.

Nadeau, C. e Bengio, Y. (2003). Inference for the generalization error, *Machine Learning* **52**(3): 239–281.

Nascimento, M. C. V., Resende, M. G. C. e Toledo, F. M. B. (2010). GRASP with path-relinking for the multi-plant capacitated plot sizing problem, *European Journal of Operational Research* **200**(3): 747–754.

Neal, M. (2003). Meta-stable memory in an artificial immune network, *Proc. 2nd Int. Conf. Artificial Immune Systems*, pp. 168–180.

Newman, D. J., Hettich, S., Blake, C. L. e Merz, C. J. (1998). UCI repository of machine learning databases, [On Line].
URL: *http://www.ics.uci.edu/ mlearn/MLRepository.html*

Ng, K. P. e Wong, K. C. (1995). A new diploid scheme and dominance change mechanism for non-stationary function optimization, *Proc. 6th Int. Conf. Genetic Algorithms*, pp. 159–166.

Nicosia, G., Cutello, V., Bentley, P. J. e Timmis, J. (eds) (2004). *Proc. 3rd Int. Conf. Artificial Immune Systems*, Vol. 3239, Springer.

Nocedal, J. e Wright, S. (1999). *Numerical Optimization*, Springer-Verlag, New York.

Nossal, G. J. V. (1993). Life, death and the immune system, *Scientific American* **269**(3): 21–30.

Ocenasek, J., Cantú-Paz, E., Pelikan, M. e Schwarz, J. (2006). Design of parallel estimation of distribution algorithms, *in* M. Pelikan, K. Sastry e E. CantúPaz (eds), *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Springer.

Ochi, L., Silva, M. e Drummond, L. (2001). GRASP and VNS for solving traveling purchaser problem, *Proc. 4th Metaheuristics Int. Conf. (MIC 2001)*, pp. 489–494.

Oliveira, C. A., Pardalos, P. M. e Resende, M. G. C. (2004). GRASP with path-relinking for the quadratic assignment problem, *Proc. 3rd Workshop on Efficient and Experimental Algorithms (WEA 2004)*, pp. 356–368.

Oltean, M., Grosan, C., Diosan, L. e Mihaila, C. (2009). Genetic programming with linear representation: a survey, *International Journal on Artificial Intelligence Tools* **18**(2): 197–238.

O'Neill, M. e Ryan, C. (2001). Grammatical evolution, *IEEE Transactions on Evolutionary Computation* **5**(4): 349–358.

Ono, S. e Nakayama, S. (2009). Multi-objective particle swarm optimization for robust optimization and its hybridization with gradient search, *Proc. IEEE Int. Conf. E-Commerce Technology*, pp. 1629–1636.

Onwubolu, G. C. e Davendra, D. (eds) (2009). *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, Studies in Computational Intelligence, Springer.

Osman, I. H. (1993). The vehicle routing problem, *Annals of Operations Research* **41**: 421–451.

Ostermeier, A., Gawelczyk, A. e Hansen, N. (1994). Step-size adaptation based on non-local use of selection information, *Proc. 3rd Int. Conf. Parallel Problem Solving from Nature (PPSN 1994)*, pp. 189–198.

Paenke, I., Branke, J. e Jin, Y. (2006). Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation, *IEEE Transactions on Evolutionary Computation* **10**(4): 405–420.

Pahner, U. e Hameyer, K. (2000). Adaptive coupling of differential evolution and multiquadrics approximation for the tuning of the optimization process, *IEEE Transactions on Magnetics* **36**(4): 1047–1051.

Palacios, P., Pelta, D. e Blanco, A. (2006). Obtaining biclusters in microarrays with population-based heuristics, *Proc. Applications of Evolutionary Computing (EvoWorkshops 2006)* pp. 115–126.

Palmer, C. C. e Kershenbaum, A. (1994a). Representing trees in genetic algorithms, *Proc. IEEE Int. Conf. Evolutionary Computation*, Bristol, USA, pp. 379–384.

Palmer, C. e Kershenbaum, A. (1994b). Representing trees in genetic algorithms, *Proc. 1st IEEE Int. Conf. Evolutionary Computation (ICEC 1994)*, Piscataway, NY, pp. 379–384.

Papadimitriou, C. H. e Steiglitz, K. (1982). *Combinatorial Optimization – Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ.

Papadimitriou, C. H. e Yannakakis, M. (1991). Optimization, approximation, and complexity classes, *Journal of Computer System Science* **43**: 425–440.

Paquete, L. e Stützle, T. (2002). An experimental investigation of iterated local search for coloring graphs, *Proc. Applications of Evolutionary Computing (EvoWorkshops 2002)*, pp. 191–208.

Park, J. e Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks, *Neural Computation* **3**: 246–257.

Parmee, I. C., Johnson, M. e Burt, S. (1994). Techniques to aid global search in engineering design, *Proc. 7th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 1994)*, pp. 377–385.

Pasia, J. M., Aguirre, H. e Tanaka, K. (2011). Improved random one-bit climbers with adaptive e-ranking and tabu moves for many-objective optimization, *Proc. 6th Int. Conf. Evolutionary Multicriterion Optimization (EMO 2011)*, Ouro Preto, Brazil, pp. 182–196.

Passerini, A., Pontil, M. e Frasconi, P. (2004). New results on error correcting output codes of kernel machines, *IEEE Transactions on Neural Networks* **15**: 45–54.

Paul, T. K. e Iba, H. (2003). Reinforcement learning estimation of distribution algorithm, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2003)*, pp. 1259–1270.

Peña, J., Lozano, J. A. e Larrañaga, P. (2004). Unsupervised learning of Bayesian networks via estimation of distribution algorithms: an application to gene expression data clustering, *International Journal of Uncertainty Fuzziness and Knowledge Based Systems* **12**: 63–82.

Peña, J. M., Robles, V., Larrañaga, P., Herves, V., Rosales, F. e Pérez, M. S. (2004). GA-EDA: hybrid evolutionary algorithm using genetic and Estimation of Distribution Algorithms, *Proc. 17th Int. Conf. Innovations in Applied Artificial Intelligence (IEA/AIE 2004)*, pp. 361–371.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Peconick, G., Wanner, E. F. e Takahashi, R. H. C. (2007). Projection-based local search operator for multiple equality constraint within genetic algorithms, *Proc. IEEE Congress on Evolutionary Computation (CEC 2007)*, Singapore, pp. 3043–3049.

Pedrycz, W. e Gomide, F. (2007). *Fuzzy Systems Engineering: toward human-centric computing*, Wiley-IEEE Press.

Pelikan, M. (2005a). *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*, Studies in Fuzziness and Soft Computing, Springer.

Pelikan, M. (2005b). *Probabilistic Model-Building Genetic Algorithms*, Springer.

Pelikan, M., Goldberg, D. E. e Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 1999)*, Orlando, FL, pp. 525–532.

Pelikan, M., Kvasnicka, V. e Pospichal, J. (1997). Read's linear codes and genetic programming, *Proc. 2nd Annual Conf. Genetic Programming*, Stanford University, CA, USA, p. 268.

Pelikan, M. e Mühlenbein, H. (1999). The bivariate marginal distribution algorithm, *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 521–535.

Pelikan, M., Sastry, K. e Goldberg, D. (2008a). iBOA: The incremental Bayesian optimization algorithm, *Proc. 10th Annual Genetic and Evolutionary Computation Conf. (GECCO 2008)*, pp. 455–462.

Pelikan, M., Sastry, K. e Goldberg, D. (2008b). Sporadic model building for efficiency enhancement of the hierarchical BOA, *Genetic Programming and Evolvable Machines* **9**(1): 53–84.

Pereira, F. B. e Tavares, J. (2009). *Bio-inspired Algorithms for the Vehicle Routing Problem*, Studies in Computational Intelligence, Springer.

Pereira, T. L., Carrano, E. G., Takahashi, R. H. C., Wanner, E. F. e Neto, O. M. (2009). Continuous-space embedding genetic algorithm applied to the degree constrained minimum spanning tree problem, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondhein, Norway, pp. 1391–1398.

Perelson, A. S., Hightower, R. e Forrest, S. (1996). Evolution and somatic learning in V-region genes, *Research in Immunology* **147**: 202–208.

Perelson, A. S. e Oster, G. F. (1979). Theoretical studies of clonal selection: Minimal antibody repertoire size and reliability of self-nonself discrimination, *Journal of Theoretical Biology* **81**: 645–670.

Perelson, A. S. e Weisbuch, G. (1997). Immunology for physicists, *Reviews of Modern Physics* **69**(4): 1219–1267.

Phadke, M. S. (1995). *Quality Engineering Using Robust Design*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Pictet, O. V., Dacorogna, M. M., Dave, R. D., Chopard, B., Schirru, R. e Tomassini, M. (1996). Genetic algorithms with collective sharing for robust optimization in financial applications, *Working Papers 1995-02-06.*, Olsen and Associates.

Pietro, A. D., While, L. e Barone, L. (2002). Learning in robocup keepaway using evolutionary algorithms, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2002)*, pp. 1065–1072.

Pimenta, E. M. C. (2005). *Abordagens para decomposição de problemas multiclasse: os códigos de correção de erros de saída*, Master's thesis, Departamento de Ciências de Computadores, Faculdade de Ciências da Universidade do Porto, Portugal.

Pitsoulis, L. S. e Resende, M. G. C. (2002). Greedy randomized adaptive search procedures, *in* P. M. Pardalos e M. G. C. Resende (eds), *Handbook of Applied Optimization*, Oxford University Press, pp. 168–183.

Poli, R. (1999). Parallel distributed genetic programming, *in* D. Corne, M. Dorigo e F. Glover (eds), *New Ideas in Optimization*, Advanced Topics in Computer Science, McGraw-Hill, Maidenhead, Berkshire, England, chapter 27, pp. 403–431.

Poli, R., Langdon, W. B. e McPhee, N. F. (2008a). *A field guide to genetic programming*, Lulu Press, Morrisville, NC, USA.

Poli, R., Langdon, W. e McPhee, N. (2008b). *A Field Guide to Genetic Programming*, Published via *http://lulu.com* and freely available at *http://www.gp-field-guide.org.uk*.

Pourtakdoust, S. H. e Nobahari, H. (2004). An extension of ant colony system to continuous optimization problems, *Proc. 4th Int. Workshop Ant Colony, Optimization and Swarm Intelligence (ANTS 2004)*, pp. 294–301.

Powell, D. e Skolnick, M. M. (1989). Using genetic algorithms in engineering design optimization with nonlinear constraints, *Proc. 5th Int. Conf. Genetic Algorithms*, San Mateo, CA, pp. 424–431.

Prais, M. e Ribeiro, C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, *INFORMS Journal on Computing* **12**: 164–176.

Price, K. V. (1997). Differential evolution vs. the functions of the 2nd ICEO, *Proc. IEEE Int. Conf. Evolutionary Computation (ICEC 1997)*, Indianapolis, USA, pp. 153–157.

Price, K. V. (1999). An introduction to differential evolution, *in* D. Corne, M. Dorigo e F. Glover (eds), *New Ideas in Optimisation*, Advanced Topics in Computer Science, McGraw-Hill Inc., London, UK, pp. 79–108.

Price, K. V., Storn, R. M. e Lampinen, J. A. (2005). *Differential Evolution: A Practical Approach to Global Optimization*, Natural Computing Series, Springer.

Prudius, A. A. e Andradóttir, S. (2005). Two simulated annealing algorithms for noisy objective functions, *Proc. 37th Conf. Winter Simulation (WSC 2005)*, pp. 797–802.

Prüfer, H. (1918). Neuer Beweis eines Satzes über Permutationen, *Arch. Math. Phys.* **27**: 742–744.

Pu, G., Chong, Z., Qiu, Z., Lin, Z. e He, J. (2006). A hybrid heuristic algorithm for HW-SW partitioning within timed automata, *Proc. Knowledge-based Intelligent Information and Engineering Systems*, pp. 459–466.

Purshouse, R. C., Jalba, C. e Fleming, P. J. (2011). Preference-driven co-evolutionary algorithms show promise for many-objective optimisation, *Proc. 6th Int. Conf. Evolutionary Multicriterion Optimization (EMO 2011)*, Ouro Preto, Brazil, pp. 136–150.

Purshouse, R. e Fleming, P. J. (2007). On the evolutionary optimization of many conflicting objectives, *IEEE Transactions on Evolutionary Computation* **11**(6): 770–784.

Qian, B., Wang, L., Hu, R., Wang, W.-L., Huang, D.-X. e Wang, X. (2008). A hybrid differential evolution method for permutation flow-shop scheduling, *The International Journal of Advanced Manufacturing Technology* **38**: 757–777.

Qing, A. (2009). *Differential Evolution: Fundamentals and Applications in Electrical Engineering*, Wiley-IEEE Press.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Learning* **1**(1): 81–106.

Quinlan, J. R. (1996). Bagging, boosting, and c4.5, *In Proc. 13th National Conf. Artificial Intelligence*, pp. 725–730.

Rai, M. M. (2002). Towards a hybrid aerodynamic design procedure based on neural networks and evolutionary methods, *20th AIAA Applied Aerodynamics Conference*, Saint Louis, MO, USA.

Raidl, G. e Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem, *Evolutionary Computation* **13**(4): 441–475.

Raidl, G. R. e Julstrom, B. A. (2003). Edge-sets: An effective evolutionary encoding of spanning trees, *IEEE Transactions on Evolutionary Computation* **7**: 225–239.

Ramsey, C. L. e Grefenstette, J. J. (1993). Case-based initialization of genetic algorithms, *Proc. 5th Int. Conf. Genetic Algorithms*, pp. 84–91.

Ratsch, G., Smola, A. J. e Mika, S. (2003). Adapting codes and embeddings for polychotomies, *Proc. Advances in Neural Information Processing Systems (NIPS 2003)*, pp. 513–520.

Ray, T. (2002). Constrained robust optimal design using a multiobjective evolutionary algorithm, *Proc. IEEE Congress on Evolutionary Computation (CEC 2002)*, pp. 419–424.

Ray, T., Tai, K. e Seow, K. C. (2000). An evolutionary algorithm for constrained optimization, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2000)*, pp. 771–777.

Rechenberg, I. (1964). Cybernetic solution path of an experimental problem, *Royal Aircraft Establishment*, Library Translation No. 1122, Farnborough, England.

Rechenberg, I. (1973). *Evolutionsstrategie - Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart.

Rechenberg, I. (1994). *Evolutionsstrategie '94*, Frommann-Holzboog, Stuttgart.

Reddy, M. J. e Kumar, D. N. (2007). Multiobjective differential evolution with application to reservoir system optimization, *Journal of Computing in Civil Engineering* **21**(2): 136–146.

Rensberger, B. (1996). In self-defense, *in* B. Resenberger (ed.), *Life Itself*, Oxford University Press, pp. 212–228.

Resende, M. (2008). Metaheuristic hybridization with greedy randomized adaptive search procedures, *in* Z.-L. Chen e S. Raghavan (eds), *Tutorials in Operations Research*, INFORMS, pp. 295–319.

Resende, M. G. C., Martí, R., Gallego, M. e Duarte, A. (2010). GRASP and path relinking for the max-min diversity problem, *Computers and Operations Research* **37**: 498–508.

Resende, M. e Ribeiro, C. (2003). Greedy randomized adaptive search procedures, *in* F. Glover e G. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, pp. 219–249.

Resende, M. e Ribeiro, C. (2005a). GRASP with path-relinking: Recent advances and applications, *in* T. Ibaraki, K. Nonobe e M. Yagiura (eds), *Metaheuristics: Progress as Real Problem Solvers*, Springer, pp. 29–63.

Resende, M. e Ribeiro, C. (2005b). Parallel Greedy Randomized Adaptive Search Procedures, *in* E. Alba (ed.), *Parallel Metaheuristics: A new class of algorithms*, John Wiley and Sons, pp. 315–346.

Resende, M. e Werneck, R. (2004). A hybrid heuristic for the p-median problem, *Journal of Heuristics* **10**: 59–88.

Resende, M. e Werneck, R. (2006). A hybrid multistart heuristic for the uncapacitated facility location problem, *European Journal of Operational Research* **174**: 54–68.

Ribeiro, C. C. e Rosseti, I. (2002). A parallel GRASP for the 2-path network design problem, *Proc. 8th Int. Euro-Par Conference on Parallel Processing (Euro-Par 2002)*, pp. 922–926.

Ribeiro, C. e Souza, M. (2002). Variable neighborhood search for the degree constrained minimum spanning tree problem, *Discrete Applied Mathematics* **118**: 43–54.

Ribeiro, C., Uchoa, E. e Werneck, R. (2002). A hybrid GRASP with perturbations for the Steiner problem in graphs, *INFORMS Journal on Computing* **14**: 228–246.

Ribeiro, C. e Urrutia, S. (2007). Heuristics for the mirrored traveling tournament problem, *European Journal of Operational Research* **127**: 775–787.

Ribeiro, C. e Vianna, D. (2005). A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure, *International Transactions in Operational Research* **12**: 325–338.

Richardson, J. T., Palmer, M. R., Liepins, G. e Hilliard, M. (1993). Some guidelines for genetic algorithms with penalty functions, *Proc. 3rd Int. Conf. Genetic Algorithms*, pp. 191–197.

Rifkin, R. e Klautau, A. (2004). In defense of one-vs-all classification, *Journal of Machine Learning Research* **5**: 1533–7928.

Robles, V., Peña, J., Larrañaga, P., Perez, M. e Herves, V. (2006). GA-EDA: A new hybrid cooperative search evolutionary algorithm, *in* J. A. Lozano, P. Larrañaga, I. Inza e E. Bengoetxea (eds), *Towards a New Evolutionary Computation – Advances in the Estimation of Distribution Algorithms*, Studies in Fuzziness and Soft Computing, Springer, pp. 187–219.

Robles, V., Peña, J., Perez, M. S., Herrero, P. e Cubo, O. (2005). Extending the GA-EDA hybrid algorithm to study diversification and intensification in GAs and EDAs, *Proc. 6th International Symp. on Intelligent Data Analysis (IDA 2005)*, Madrid, Spain, pp. 339–350.

Ronconi, D. e Armentano, V. (1997). Busca tabu para a minimização do tempo total de atraso no problema de flowshop, *Pesquisa Operacional* **15**: 23–34.

Roseman, M. A. e Gero, J. S. (1985). Reducing the Pareto optimal set in multicriteria optimization, *Engineering Optimization* **8**: 189–198.

Rossi, A. L. D. e Carvalho, A. C. P. L. F. (2008). Bio-inspired optimization techniques for SVM parameter tuning, *Proc. of 10th Brazilian Symp. on Neural Networks*, pp. 435–440.

Rossi, A. L. D., Carvalho, A. C. P. L. F. e Soares, C. (2008). Bio-inspired parameter tunning of MLP networks for gene expression analysis, *Proc. 8th Int. Conf. Hybrid Intelligent Systems*, pp. 57–62.

Rossi, C., Barrientos, A. e del Cerro, J. (2007). Two adaptive mutation operators for optima tracking in dynamic optimization problems with evolution strategies, *Proc. 9th Annual Genetic and Evolutionary Computation Conf. (GECCO 2007)*, pp. 697–704.

Rothlauf, F. (2005). *Representations for Genetic and Evolutionary Algorithms*, 2nd edn, Springer Verlag, Berlim.

Routhlauf, F., Goldberg, D. e Heinzl, A. (2002). Network random keys - A tree network representation scheme for genetic and evolutionary algorithms, *Evolutionary Computation* **10**: 75–97.

Roy, B. (1987). Meaning and validity of interactive procedures as tools for decision making, *European Journal of Operational Research* **31**: 297–303.

Roy, B. (1996). *Multicriteria Methodology for Decision Aiding*, Kluwer Academic Publishers.

Roy, R., Azene, Y., Farrugia, D., Onisa, C. e Mehnen, J. (2009). Evolutionary multi-objective design optimisation with real life uncertainty and constraints, *CIRP Annals - Manufacturing Technology* **58**(1): 169 – 172.

Rutenbar, R. A. (1989). Simulated annealing algorithms: An overview, *IEEE Circuits and Devices Magazine* **5**(1): 19–26.

Ruzek, B. e Kvasnicka, M. (2001). Differential evolution algorithm in the earthquake hypocenter location, *Pure and Applied Geophysics* **158**(4): 667–693.

Salami, M. e Hendtlass, T. (2003). A fast evaluation strategy for evolutionary algorithms, *Applied Soft Computing* **2**(3): 156–173.

Sano, Y. e Kita, H. (2000). Optimization of noisy fitness functions by means of genetic algorithms using history of search, *Proc. 6th Int. Conf. Parallel Problem Solving from Nature (PPSN 2000)*, pp. 571–580.

Sano, Y. e Kita, H. (2002). Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation, *Proc. IEEE Congress on Evolutionary Computation (CEC 2002)*, pp. 360–365.

Sano, Y., Kita, H., Kamihira, I. e Yamaguchi, M. (2000). Online optimization of an engine controller by means of a genetic algorithm using history of search, *Proc. 26th Annual Conf. IEEE Industrial Electronics Society (IECON 2000)*, pp. 2929–2934.

Sanseverino, C. M. R. (2005). A hybrid approach based on evolutionary strategies and interval arithmetic to perform robust designs, *Proc. Applications of Evolutionary Computing (EvoWorkkshops 2005)*, pp. 623–628.

Santana, R. (2005). Estimation of distribution algorithms with Kikuchi approximations, *Evolutionary Computation* **13**(1): 67–97.

Santana, R., Larrañaga, P. e Lozano, J. (2004). Protein folding in 2-dimensional lattices with estimation of distribution algorithms, *Proc. 1st Int. Symp. Biological and Medical Data Analysis*, pp. 388–398.

Santana, R., Larrañaga, P. e Lozano, J. (2008a). Combining variable neighborhood search and estimation of distribution algorithms in the protein side chain placement problem, *Journal of Heuristics* **14**(5): 519–547.

Santana, R., Larrañaga, P. e Lozano, J. (2008b). Protein folding in simplified models with estimation of distribution algorithms, *IEEE Transactions on Evolutionary Computation* **12**(4): 418–438.

Santana, R., Larrañaga, P. e Lozano, J. (2009). Research topics in discrete estimation of distribution algorithms based on factorizations, *Memetic Computing* **1**(1): 35–54.

Santos, H. G., Ochi, L. S. e Souza, M. J. F. (2005). A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem, *ACM Journal of Experimental Algorithmics* **10**: art–2.09. 15 p.

Sareni, B. e Krahenbuhl, L. (1998). Fitness sharing and niching methods revisited, *IEEE Transactions on Evolutionary Computation* **2**(3): 97–106.

Sastry, K., Goldberg, D. E. e Pelikan, M. (2001). Don't evaluate, inherit, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2001)*, pp. 551–558.

Sastry, K., Pelikan, M. e Goldberg, D. (2006). Efficiency enhancement of estimation of distribution algorithms, *in* M. Pelikan, K. Sastry e E. CantúPaz (eds), *Scalable Optimization via Probabilistic Modeling*, Springer, pp. 161–185.

Sastry, K., Pelikan, M. e Goldberg, D. E. (2005). Limits of scalability of multiobjective estimation of distribution algorithms, *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, pp. 2217–2224.

Savic, D. A., Walters, G. A. e Schwab, M. (1997). Multiobjective genetic algorithms for pump scheduling in water supply, *AISB Workshop on Evolutionary Computation - Selected Papers*, pp. 227–236.

Schied, A. (2006). Risk measures and robust optimization problems, *Stochastic Models* **22**(4): 753–831.

Schmidt, M. e Lipson, H. (2008). Coevolution of fitness predictors, *IEEE Transactions on Evolutionary Computation* **12**(6): 736–749.

Schutze, O., Lara, A. e Coello-Coello, C. A. (2011). On the influence of the number of objectives on the hardness of a multiobjective optimization problem, *IEEE Transactions on Evolutionary Computation* . DOI: 10.1109/TEVC.2010.2064321.

Schwarz, G. (1978). Estimating the dimension of a model, *Annals of Statistics* **6**: 461–465.

Schwarz, J. e Jaros, J. (2008). Parallel bivariate marginal distribution algorithm with probability model migration, *in* Y.-P. Chen e M.-H. Lim (eds), *Linkage in Evolutionary Computation*, Springer, pp. 3–23.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*, Wiley & Sons, Chichester.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*, Wiley, New York.

Sebag, M. e Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces, *Proc. 5th Int. Conf. Parallel Problem Solving from Nature (PPSN 1998)*, London, UK, pp. 418–427.

Serra, D. e Colomé, R. (2001). Consumer choice and optimal location models: Formulations and heuristics, *Papers in Regional Science* **80**: 439–464.

Sevaux, M. e Sörensen, K. (2002a). A genetic algorithm for robust schedules, *Proc. 8th Int. Workshop on Project Management and Scheduling*, Valencia, Spain.

Sevaux, M. e Sörensen, K. (2002b). A genetic algorithm for robust schedules in a just-in-time environment, *Working Papers 2003008*, University of Antwerp, Faculty of Applied Economics.

Shachter, R. D. e Kenley, C. R. (1989). Gaussian influence diagrams, *Management Science* **35**(5): 527–550.

Shakya, S., McCall, J. e Brown, D. (2004). Updating the probability vector using MRF technique for a univariate EDA, *Proc. 4th European Starting AI Researcher Symp.*, pp. 15–25.

Shakya, S., McCall, J. e Brown, D. (2005). Using a Markov network model in a univariate EDA: An empirical cost-benefit analysis, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2005)*, pp. 727–734.

Shen, L. e Tan, E. C. (2005). Seeking better output-codes with genetic algorithm for multiclass cancer classification. Unpublished paper.

Shirasaka, M., Zhao, Q., Hammami, O., Kuroda, K. e Saito, K. (1998). Automatic design of binary decision trees based on genetic programming, *Proc. 2nd Asia-Pacific Conf. Simulated Evolution and Learning*, Canberra, Australia.

Simón, M. D. J., Pulido, J. A. G., Rodríguez, M. A. V., Pérez, J. M. S. e Criado, J. M. G. (2006). A genetic algorithm to design error correcting codes, *Proc. 13th IEEE Mediterranean Eletrotechnical Conf.*, pp. 807–810.

Sindhya, K., Deb, K. e Miettinen, K. (2011). Improving convergence of evolutionary multi-objective optimization with local search: a concurrent-hybrid algorithm, *Natural Computing* . DOI 10.1007/s11047-011-9250-4.

Smith, D. J., Forrest, S., Hightower, R. R. e Perelson, S. A. (1997). Deriving shape space parameters from immunological data, *Journal of Theoretical Biology* **189**: 141–150.

Smith, J. (2007). On replacement strategies in steady state evolutionary algorithms, *Evolutionary Computation* **15**(1): 29–59.

Smith, K. I., Everson, R. M. e Fieldsend, J. E. (2004). Dominance measures for multi-objective simulated annealing, *Proc. IEEE Congress on Evolutionary Computation (CEC 2004)*, pp. 23–30.

Smith, R. E., Dike, B. A. e Stegmann, S. A. (1995). Fitness inheritance in genetic algorithms, *Proc. ACM Symp. Applied computing (SAC 1995)*, pp. 345–350.

Soak, S., Corne, D. W. e Ahn, B. (2006). The edge-window-decoder representation for tree-based problems, *IEEE Transactions on Evolutionary Computation* **10**: 124–144.

Soares, G. L., Guimaraes, F. G., Maia, C. A., Vasconcelos, J. A. e Jaulin, L. (2009). Interval robust multi-objective evolutionary algorithm, *Proc. IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 1637–1643.

Socha, K. (2004). ACO for continuous and mixed-variable optimization., *Proc. 4th Int. Workshop Ant Colony, Optimization and Swarm Intelligence (ANTS 2004)*, pp. 25–36.

Souza, B. F. e Carvalho, A. C. P. L. F. (2005). Gene selection based on multi-class support vector machines and genetic algorithms, *Genetics and Molecular Research* **4**(3): 599–607.

Souza, B. F., Carvalho, A. C. P. L. F., Calvo, R. e Ishii, R. P. (2006). Multiclass SVM model selection using particle swarm optimization, *Proc. 6th Int. Conf. Hybrid Intelligent Systems*, Washington, DC, USA, p. 31.

Souza, M. J. F. (2000). *Programação de horários em escolas: uma aproximação por metaheurísticas*, Tese de doutorado, Programa de Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro.

Souza, M. J. F., Maculan, N. e Ochi, L. S. (2001). A GRASP-tabu search algorithm to solve a school timetabling problem, *Proc. 4th Metaheuristics Int. Conf. (MIC 2001)*, pp. 53–58.

Souza, M. J. F., Ochi, L. S. e Maculan, N. (2004). A GRASP-tabu search algorithm for solving school timetabling problems, *in* M. G. C. Resende e J. P. Souza (eds), *Metaheuristics: Computer Decision-Making*, Kluwer Academic Publishers, pp. 659–672.

Sprent, J. (1994). T and B memory cells, *Cell* **76**: 315–312.

Srinivas, N. e Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms, *Evolutionary Computation* **2**(3): 221–248.

Stagge, P. (1998). Averaging efficiently in the presence of noise, *Proc. 5th Int. Conf. Parallel Problem Solving from Nature (PPSN 1998)*, pp. 188–200.

Statnikov, A., Aliferis, C. F., Tsamardinos, I., Hardin, D. e Levy, S. (2005). A comprehensive evaluation of multicategory methods for microarray gene expression cancer diagnosis, *Bioinformatics* **21**(5): 631–643.

Statnikov, A., Tsamardinos, I., Dosbayev, Y. e Aliferis, C. F. (2005). GEMS: A system for automated cancer diagnosis and biomarker discovery from microarray gene expression data, *International Journal of Medical Informatics* **74**(7-8): 491–503.

Steuer, R. (1977). An interactive multiple objective linear programming procedure, *TIMS Studies in the Management Sciences* **6**: 225–239.

Steuer, R. (1986). *Multiple Criteria Optimization: Theory, Computation and Application*, Wiley.

Storb, U. (1998). Progress in understanding the mechanisms and consequences of somatic hypermutation, *Immunological Reviews* **162**: 5–11.

Storn, R. M. (1999). Designing digital filters with differential evolution, *in* D. Corne, M. Dorigo e F. Glover (eds), *New Ideas in Optimization*, Advanced Topics in Computer Science, McGraw-Hill Inc., London, UK, pp. 109–125.

Storn, R. M. e Price, K. V. (1995). Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces, *Technical Report TR-95-012*, International Computer Science Institute (ICSI), University of California, Berkeley.

Storn, R. M. e Price, K. V. (1996). Minimizing the real functions of the ICEC 96 contest by differential evolution, *Proc. IEEE Int. Conf. Evolutionary Computation (ICEC 1996)*, Nagoya, Japão, pp. 842–844.

Storn, R. M. e Price, K. V. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11**(4): 341–359.

Student (1908). The probable error of a mean, *Biometrika* **6**(1): 1–25.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem, *European Journal of Operational Research* **174**(3): 1519–1539.

Stützle, T. e Hoos, H. (1996). Improving the ant system: A detailed report on the max-min ant system, *Technical Report AIDA-96-12*, FG Intellektik, FB Informatik, TU Darmstadt/ Germany.

Stützle, T. e Hoos, H. (2002). Analyzing the run-time behaviour of iterated local search for the TSP, *in* P. Hansen e C. Ribeiro (eds), *Essays and Surveys on Metaheuristics*, Kluwer Academic Publishers, pp. 589–612.

Stützle, T. e Hoos, H. H. (2000). MAX–MIN ant system, *Future Generation Computer Systems* **16**: 889–914.

Suganthan, P. N. (1999). Particle swarm optimiser with neighbourhood operator, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, Washington, USA. DOI: 10.1109/CEC.1999.785514.

Suman, B. (2004). Study of simulated annealing based algorithms for multiobjective optimization of a constrained problem, *Computers and Chemical Engineering* **28**(9): 1849–1871.

Sun, J., Zhang, Q. e Tsang, E. (2005). DE/EDA: A new evolutionary algorithm for global optimization, *Information Sciences* **169**(3-4): 249–262.

Suzuki, J. (1996). Learning Bayesian belief networks based on the MDL principle: An efficient algorithm using the branch and bound technique, *Proc. Int. Conf. Machine Learning* **1**: 462–470.

Syswerda, G. (1989). Uniform crossover in genetic algorithms, *Proc. 3rd Int. Conf. Genetic Algorithms (ICGA 1989)*, San Francisco, CA, USA, pp. 2–9.

Takagi, H. et al. (2001). Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation, *Proceedings of the IEEE* **89**(9): 1275–1296.

Takahashi, H. e Matsuyama, A. (1980). An approximate solution for the Steiner problem in graphs, *Mathematica Japonica* **24**: 573–577.

Takahashi, R. H. C., Vasconcelos, J. A., Ramirez, J. A. e Krahenbuhl, L. (2003). A multiobjective methodology for evaluating genetic operators, *IEEE Transactions on Magnetics* **39**: 1321–1324.

Tan, P.-N., Steinbach, M. e Kumar, V. (2005). *Introduction to Data Mining*, Addison-Wesley, Boston, MA, USA.

Tang, M. e Lau, R. Y. K. (2005). A hybrid estimation of distribution algorithm for the minimal switching graph problem, *Proc. Int. Conf. Computational Intelligence for Modelling, Control and Automation (CIMCA'05) and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Washington, DC, USA, pp. 708–713.

Teller, A. (1996). Evolving programmers: The co-evolution of intelligent recombination operators, *in* P. J. Angeline e K. E. Kinnear, Jr. (eds), *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, USA, chapter 3, pp. 45–68.

Theodoridis, S. e Koutroumbas, K. (2006). *Pattern Recognition*, 3rd edn, Academic Press, Inc., Orlando, FL, USA.

Thierens, D. e Bosman, P. (2001). Multi-objective mixture-based iterated density estimation evolutionary algorithms, *Proc. Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 663–670.

Thompson, A. (1998). On the automatic design of robust electronics through artificial evolution, *Proc. 2nd Int. Conf. Evolvable Systems (ICES 1998)*, pp. 13–24.

Timmis, J., Bentley, P. J. e Hart, E. (eds) (2003). *Proc. 2nd Int. Conf. Artificial Immune Systems*, Springer.

Timmis, J., Neal, M. e Hunt, J. (2000). An artificial immune system for data analysis, *Biosystems* **55**.

Tizard, I. R. (1995). *Immunology: An Introduction*, 4th edn, Sanders College Publishing.

Trojanowski, K., Michalewicz, Z. e Xiao, J. (1997). Adding memory to the evolutionary planner/navigator, *Proc. IEEE Int. Conf. Evolutionary Computation (ICEC 1997)*, pp. 483–487.

Tsai, H., Yang, J., Tsai, Y. e Kao, C. (2004). An evolutionary algorithm for large traveling salesman problems, *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics* **34**(4): 1718–1729.

Tsai, J.-T., Chou, J.-H. e Liu, T.-K. (2006). Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm, *IEEE Transactions on Neural Networks* **17**(1): 69–80.

Tsutsui, S. (2004). Ant colony optimisation for continuous domains with aggregation pheromones metaphor, *Proc. 5th Int. Conf. Recent Advances in Soft Computing (RASC 2004)*, pp. 207–212.

Tsutsui, S., Fujimoto, Y. e Ghosh, A. (1997). Forking genetic algorithms: Gas with search space division schemes, *Evolutionary Computation* **5**(1): 61–80.

Tsutsui, S. e Ghosh, A. (1997). Genetic algorithms with a robust solution searching scheme, *IEEE Transactions on Evolutionary Computation* **1**(3): 201–208.

Tsutsui, S., Ghosh, A. e Fujimoto, Y. (1996). A robust solution searching scheme in genetic search, *Proc. 4th Int. Conf. Parallel Problem Solving from Nature (PPSN 1996)*, pp. 543–552.

Tür, G. e Güvenir, H. A. (1996). Decision tree induction using genetic, *Proc. 5th Turkish Symp. Artificial Intelligence and Neural Networks*, pp. 187–196.

Ursem, R. K. (2000). Multinational GAs: Multimodal optimization techniques in dynamic environments, *Proc. 2nd Genetic and Evolutionary Computation Conf. (GECCO 2000)*.

Vapnik, V. e Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of occurrence of events to their probabilities, *Theory of Probability and Its Applications* **2**(16): 264–280.

Vapnik, V. N. (1995). *The nature of statistical learning theory*, Springer-Verlag, New York, NY, USA.

Vapnik, V. N. (1998). *Statistical Learning Theory*, Wiley-Interscience.

Varela, F. J. e Coutinho, A. (1991). Second generation immune networks, *Immunology Today* **12(5)**: 159–166.

Varela, F. J., Coutinho, A., Dupire, E. e Vaz, N. (1988). Cognitive networks: Immune, neural and otherwise, *Theoretical Immunology* **2**: 359–375.

Vasconcelos, J. A., Krahenbuhl, L. e Nicolas, A. (1996). Simulated annealing coupled with the tabu search method for continuum optimization in electromagnetics, *IEEE Transactions on Magnetics* **32**(3): 1206–1209.

Vavak, F., Jukes, K. e Fogarty, T. C. (1997). Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variable range of local search., *Proc. Int. Conf. Genetic Algorithms (ICGA 1997)*, pp. 719–726.

Venkatraman, S. e Yen, G. G. (2005). A generic framework for constrained optimization using genetic algorithms, *IEEE Transactions on Evolutionary Computation* **9**(4): 424–435.

Wallis, J. L. e Houghten, S. K. (2002). A comparative study of search techniques applied to the minimum distance problem of BCH codes, *Technical Report CS-02-08*, Department of Computer Science, Brock University.

Wan, G. e Yen, B. P. C. (2002). Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties, *European Journal of Operational Research* **142**: 271–281.

Wang, J. (2004). A fuzzy robust scheduling approach for product development projects, *European Journal of Operational Research* **152**(1): 180–194.

Wang, J. (2007). Genetic particle swarm optimization based on estimation of distribution, *Proc. Int. Conf. Life System Modeling and Simulation (LSMS 2007)*, Shanghai, China, pp. 287–296.

Wang, J., Kuang, Z., Xu, X. e Zhou, Y. (2009). Discrete particle swarm optimization based on estimation of distribution for polygonal approximation problems, *Expert Systems and Applications* **36**(5): 9398–9408.

Wanner, E. F., Guimarães, F. G., Saldanha, R. R., Takahashi, R. H. C. e Fleming, P. F. (2005). Constraint quadratic approximation operator for treating equality constraints with genetic algorithms, *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK, pp. 2255–2262.

Wanner, E. F., Guimarães, F. G., Takahashi, R. H. C. e Fleming, P. J. (2008). Local search with quadratic approximations into memetic algorithms for optimization with multiple criteria, *Evolutionary Computation* **16**(2): 185–224.

Whigham, P. A. (1995). Grammatically-based genetic programming, *Proc. Workshop on Genetic Programming: From Theory to Real-World Applications*, Tahoe City, California, USA, pp. 33–41.

While, L., Bradstreet, L. e Barone, L. (2011). A fast way of calculating exact hypervolumes, *IEEE Transactions on Evolutionary Computation* . DOI: 10.1109/TEVC.2010.2077298.

Whitley, D. (1989). The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best, *Proc. 3rd Int. Conf. Genetic Algorithms (ICGA 1989)*, San Francisco, CA, USA, pp. 116–121.

Wiesmann, D., Hammel, U. e Bäck, T. (1998). Robust design of multilayer optical coatings by means of evolution strategies, *IEEE Transactions on Evolutionary Computation* **2**: 162–167.

Wiggins, G. A., Papadopoulos, G., Phon-Amnuaisuk, S. e Tuson, A. (1998). Evolutionary methods for musical composition, *International Journal of Computing Anticipatory Systems* **4**.

Wineberg, M. e Oppacher, F. (2000). Enhancing the GA's ability to cope with dynamic environments, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2000)*, pp. 3–10.

Witten, I. H. e Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann.

Xue, F., Sanderson, A. C. e Graves, R. J. (2005). Multi-objective differential evolution - algorithm, convergence analysis, and applications, *Proc. IEEE Congress on Evolutionary Computation (CEC 2005)*, pp. 743–750.

Yagiura, M., Glover, F. e Ibaraki, T. (2006). A path relinking approach with ejection chains for the generalized assignment problem, *European Journal of Operational Research* **169**: 548–569.

Yang, D. e Flockton, S. (1995). Evolutionary algorithms with a coarse-to-fine function smoothing, *Proc. Int. Conf. Evolutionary Computation (ICEC 1995)*, pp. 657–662.

Yang, S. e Yao, X. (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems, *Soft Computing Journal* **9**(11): 815–834.

Yang, Y., Kreipl, S. e Pinedo, M. (2000). Heuristics for minimizing total weighted tardiness in flexible flow shops, *Journal of Scheduling* **3**(2): 89–108.

Ye, N. (2003). *The Handbook of Data Mining*, Lea.

Yu, P. (1985). *Multiple Criteria Decision Making: Concepts, Techniques and Extensions*, Plenum Press, New York.

Yuan, B., Orlowska, M. E. e Sadiq, S. W. (2008). Extending a class of continuous estimation of distribution algorithms to dynamic problems, *Optimization Letters* **2**(3): 433–443.

Z. Bandar, A.-A. e McLean, H. (1999). Genetic algorithm based multiple decision tree induction, *Proc. 6th Int. Conf. Neural Information Processing*, pp. 429–434.

Zaharie, D. (2002). Critical values for the control parameters of differential evolution algorithms, *Proc. 8th Int. Conf. Soft Computing*, Brno, Czech Republic, pp. 62–67.

Zar, J. (2009). *Biostatistical Analysis, 5th edition*, Prentice Hall.

Zhang, Q., Sun, J. e Tsang, E. (2007). Combinations of estimation of distribution algorithms and other techniques, *International Journal of Automation and Computing* **4**(3): 273–280.

Zhang, Q., Sun, J., Tsang, E. e Ford, J. (2003). Hybrid estimation of distribution algorithm for global optimisation, *Engineering Computations* **21**(1): 91–107.

Zhang, Q., Zhou, A. e Jin, Y. (2008). RM-MEDA: A regularity model-based multiobjective estimation of distribution algorithm, *IEEE Transactions on Evolutionary Computation* **12**(1): 41–63.

Zhao, H. (2007). A multi-objective genetic programming approach to developing pareto optimal decision trees, *Decision Support System* **43**(3): 809–826.

Zhao, Q. e Shirasaka, M. (1999). A study on evolutionary design of binary decision trees, *Proc. IEEE Congress on Evolutionary Computation (CEC 1999)*, Washington D.C., USA, pp. 1988–1993.

Zhou, A., Kang, L. e Yan, Z. (2003). Solving dynamic tsp with evolutionary approach in real time, *Proc. IEEE Congress on Evolutionary Computation (CEC 2003)*, pp. 951–957.

Zhou, A., Zhang, Q., Jin, Y. e Sendhoff, B. (2008). Combination of EDA and DE for continuous bi-objective optimization, *Proc. IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 1447–1454.

Zhou, Y., Wang, J. e Yin, J. (2007). A discrete estimation of distribution particle swarm optimization for combinatorial optimization problems, *Proc. 3rd Int. Conf. Natural Computation (ICNC 2007)*, pp. 80–84.

Zionts, S. e Wallenius, J. (1976). An interactive programming method for solving the multiple criteria problem, *Management Science* **22**: 652–663.

Zionts, S. e Wallenius, J. (1983). An interactive multiple objective linear programming method for a class of underlying nonlinear utility functions, *Management Science* **29**: 519–529.

Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, ETH, Zurich, Switzerland.

Zitzler, E., Brockhoff, D. e Thiele, L. (2007). The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration, *Proc. 4th Int. Conf. Evolutionary Multi-Criterion Optimization (EMO 2007)*, Matsushima-Sendai, Japan, pp. 862–876.

Zitzler, E., Deb, K. e Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results, *Evolutionary Computation* **8**(2): 173–195.

Zitzler, E. e Kunzli, S. (2004). Indicator-based selection in multiobjective search, *Proc. 8th Int. Conf. Parallel Problem Solving from Nature (PPSN 2004)*, pp. 832–842.

Zitzler, E., Laumanns, M. e Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization, *in* X. Gandibleux, M. Sevaux, K. Sörensen e V. T'Kindt (eds), *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems, Springer.

Zitzler, E., Laumanns, M. e Thiele, L. (2001). SPEA2: Improving the stregth Pareto evolutionary algorithm, *Technical report*, TIK report no. 103, Swiss Federal Institute of Technology, Zurich, Switzerland.

Zitzler, E., Laumanns, M. e Thiele, L. (2002). SPEA2: Improving the strength Pareto evolutionary algorithm, *Proc. Evolutionary Methods for Design, Optimisation and Control (EUROGEN 2001)*, Barcelona, Spain, pp. 95–100.

Zitzler, E. e Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach, *IEEE Transations on Evolutionary Computation* **3**(4): 257–271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M. e Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review, *IEEE Transactions on Evolutionary Computation* **7**(2): 117–132.

Índice Remissivo

SÉRIE ENSINO
IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS
2012

IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS