Fundamentos de Algoritmos Evolutivos

Paulo Henrique Ribeiro Gabriel
Alexandre Cláudio Botazzo Delbem

Sumário

Capítulo 1

Introdução

A Computação Evolutiva (CE) é uma área de pesquisa que tem se expandido rapidamente. Entre os motivos para isso podem ser enumerados:

1. o desenvolvimento de algoritmos capazes de encontrar soluções adequadas para problemas complexos, ainda não resolvidos por outras técnicas computacionais;

2. simplicidade dos métodos, chamados de algoritmos evolutivos (AEs), utilizando princípios básicos de Teoria da Evolução e Genética que podem ser modelados por poucas e simples linhas de código;

3. adaptação relativamente fácil para problemas das mais diversas áreas.

Essa área de pesquisa originou-se de várias frentes de estudo, cuja interação produziu os AEs atuais. Dentre esses, sem dúvidas os algoritmos genéticos (AGs) são os mais conhecidos, principalmente devido à sua utilização em Inteligência Artificial (IA) (Rezende, 2003). Por outro lado, os AGs são uma técnica evolutiva universal. Este texto busca apresentar os AEs dando a ponderação merecida aos AGs sem, com isso, esconder aspectos dos outros AEs que, em geral, são importantes para solução de problemas que são difíceis para os próprios AGs e outros métodos de otimização e de busca.

Os primeiros trabalhos envolvendo AEs são datados da década de 1930, quando sistemas evolutivos naturais passaram a ser investigados como algoritmos de exploração de múltiplos picos de uma função objetivo. Porém, apenas com o maior acesso a computadores, a partir da década de 1960, é que se intensificaram os desenvolvimentos de AEs (De Jong, 2006), com a realização de diversos estudos teóricos e empíricos.

Nesse contexto, três abordagens de AEs foram desenvolvidas de forma independente (De Jong, 2006): a *programação evolutiva* (PE), as *estratégias evolutivas* (EEs) e os *algoritmos genéticos* (AGs). O princípio básico de todas essas técnicas é, no entanto, a mesma (Eiben e Smith, 2003): dada uma *população* de *indivíduos* (i.e., um conjunto de soluções), pressões do ambiente desencadeiam um processos de *seleção natural* (ou seja, um processo que privilegia

as melhores soluções até então encontradas), o que causa um incremento na adequação das soluções. Dada uma função a ser otimizada (seja maximizada ou minimizada), gera-se aleatoriamente um conjunto de soluções, i.e., elementos pertencentes ao domínio da função, e aplica-se a função para medir a qualidade das soluções candidatas, atribuindo-lhes um valor que mede sua adequação, chamado *fitness*.

Com base no *fitness*, algumas das melhores soluções são *selecionadas* para darem origem a uma nova população pela aplicação de operadores de *recombinação* e/ou *mutação*. A recombinação é um operador aplicado a duas ou mais soluções candidatas (chamadas *pais*) e resulta em duas ou mais novas soluções (chamadas *descendentes* ou *filhos*). A mutação é aplicada em uma candidata a fim de gerar outra. Ao final desse processo, as novas candidatas (descendentes) competem com as candidatas da geração anterior, com base no *fitness*, para assumir um lugar na nova população. Esse processo é iterado até que uma candidata apresente uma solução que seja suficientemente qualificada ou até que um número máximo de iterações, também chamadas *gerações* seja obtido.

Vários componentes de um processo evolutivo são estocásticos: a seleção favorece indivíduos mais bem adaptados (ou seja, com melhor *fitness*), mas existe também a possibilidade de serem selecionados outros indivíduos. A recombinação dos indivíduos é aleatória, assim como a mutação. Uma representação geral de um AE típico pode ser vista no pseudocódigo do Algoritmo 1 (Eiben e Smith, 2003).

Algoritmo 1: Pseudocódigo de um AE típico.

Entrada: Parâmetros típicos (De Jong, 2006).
Saída: População final de soluções

```
1 INICIALIZA população com soluções candidatas aleatórias
2 AVALIA cada candidata
3 repita
4     SELECIONA pais
5     RECOMBINA pares de pais
6     MUTA os descendentes resultantes
7     AVALIA novas candidatas
8     SELECIONA indivíduos para a nova geração
9 até CONDIÇÃO DE PARADA satisfeita ;
```

Conforme mencionado, os AEs foram desenvolvidos originalmente como ferramentas de modelagem e simulação computacional. Não demorou, no entanto, para que fossem explorados como técnica de otimização. Vale destacar também que, diferentemente de outras técnicas de busca e otimização, as quais em geral constroem uma única solução por iteração[1], os AEs trabalham com conjuntos de soluções, o que reduz, em muitos casos, o número de iterações necessárias para a obtenção das soluções (em outras palavras, reduz o *tempo de convergência*). Trabalhos mais recentes, datados da década de 1990, desenvolveram AEs específicos para as áreas de Aprendizado de Máquina (AM) (Rezende, 2003) e construção de modelos probabilísticos. Nos últimos anos, o desempenho dos AEs tem aumentado significativamente com o estudo

[1] Várias das técnicas que iteram um única solução são descritas em (Sait e Youssef, 1999).

de *algoritmos de estimação de distribuição* (AEDs) (Larraeñaga e Lozano, 2001; Pelikan et al., 1999).

Este texto foca os principais AEs, suas representações e operadores, e está organizado como segue. O Capítulo 2 apresenta aspectos básicos do processo evolutivo que são fundamentas para o desenvolvimento de AEs e a terminologia que será empregada no restante deste texto. O Capítulo 3 descreve os principais tipos de AEs e o Capítulo 4 apresenta mais detalhadamente os operadores de reprodução. O Capítulo 5 sintetiza os aspectos principais de AEs e considerações sobre a utilização deles.

Capítulo 2

Base Biológica

Os AEs podem ser vistos como técnicas de *Computação Bioinspirada* (Carvalho et al., 2004; Mange, 1998) ou Computação Natural (Ballard, 1999; Castro, 2006). Essas áreas de pesquisa abrangem uma série de técnicas computacionais fundamentadas em conceitos biológicos. As técnicas evolutivas apresentam conceitos cuja origem está em diversos campos da Biologia, especialmente em idéias evolucionistas e na Genética. Este Capítulo foca nesses conceitos (Seção 2.1) e sintetiza a terminologia empregada na definição de AEs (Seção 2.2).

2.1 O Processo Evolutivo

Os AEs são fortemente inspirados em processos evolutivos que ocorrem na natureza. Segundo De Jong (2006), os principais componentes dos sistemas evolutivos são:

- Populações de indivíduos: uma ou mais populações concorrem por recursos limitados;
- Aptidão, que reflete a habilidade do indivíduo para sobreviver e reproduzir-se;
- A noção de mudanças dinâmicas nas populações devido ao nascimento e morte dos indivíduos;
- Os conceitos de variabilidade e hereditariedade, ou seja, os novos indivíduos possuem muitas das características de seus pais, embora não sejam idênticos.

Tais conceitos foram inspirados na chamada *Teoria Sintética da Evolução*, também conhecida como *neodarwinismo* (Ridley, 1996). O neodarwinismo admite que os principais fatores evolutivos são a *mutação*, a *recombinação gênica* e a *seleção natural* (Amabis e Martho, 1985; Ridley, 1996), resumidos nas Subseções seguintes.

2.1.1 Mutação Gênica

A origem da variabilidade é a *mutação*, processo pelo qual o gene[1] sofre alterações em sua estrutura. Tais alterações são modificações na seqüência de bases do DNA. Essa molécula, quando duplicada, produz cópias idênticas de si, ou seja, diferentes da original (sem mutação), transmitindo hereditariamente a mudança. Isso pode acarretar a alteração da seqüência de aminoácidos da proteína, modificando o metabolismo celular, podendo favorecer o organismo ou mesmo ser letal.

2.1.2 Recombinação Gênica

O processo evolutivo seria relativamente lento se não fosse possível colocar juntas, em um mesmo indivíduo, mutações ocorridas em indivíduos da geração anterior. O fenômeno que possibilita esse evento é a *reprodução sexuada*. É importante considerar que a seleção natural não atua aceitando ou rejeitando mudanças individuais, mas sim escolhendo as melhores combinações gênicas entre todas variações presentes na população.

2.1.3 Seleção Natural

A seleção natural é conseqüência de dois fatores:

1. os membros de uma espécie diferem entre si;
2. a espécie produz descendência em maior número o de indivíduos que de fato podem sobreviver.

Os indivíduos mais aptos a sobreviver são aqueles que, graças à variabilidade genética, herdaram a combinação gênica mais adaptada para determinadas condições naturais.

2.2 Terminologia Básica

A seguir é apresentada a terminologia necessária, adaptada de (Sait e Youssef, 1999), para o estudo de AEs. Os principais termos discutidos encontram-se sintetizados na Figura 2.1.

2.2.1 Cromossomo, Genes e Alelos

A estrutura que codifica como os organismos são construídos é chamada *cromossomo*. Os cromossomos associam-se de modo a formar um organismo e seu número varia de uma espécie para outra (Amabis e Martho, 1985). O conjunto completo de cromossomos de um ser vivo é chamado *genótipo* e as características do organismo gerado com base no genótipo constituem o

[1] *Gene* é um segmento de DNA que contém uma informação codificada para determinada característica ou processo que a célula tem ou executa (Amabis e Martho, 1985).

fenótipo. De forma similar, a representação de soluções de um problema podem ser codificadas em uma estrutura de dados chamada cromossomo.

Os cromossomos são codificados em um conjunto de símbolos chamados *genes*. Os diferentes valores de um gene são chamados *alelos*. A posição do gene em um cromossomo é denominada *locus* (Coello Coello et al., 2002).

A representação das soluções candidatas (ou seja, dos indivíduos) é o primeiro estágio da elaboração de um AE e é crucial para o desempenho do algoritmo. Essa etapa consiste em definir o genótipo e a forma com que esse é mapeado no fenótipo. Dependendo da escolha, são necessários operadores de reprodução específicos (ver Capítulo 4).

A codificação mais simples é a *representação binária*: o genótipo é definido como um arranjo de 0s e 1s. É necessário definir o tamanho do arranjo, bem como o mapeamento genótipo–fenótipo. Entretanto, em muitas aplicações do mundo real, a representação binária pode apresentar fraco poder de expressão (Deb, 2001), não sendo eficiente na representação das possíveis soluções. Uma alternativa empregada é a *representação em ponto flutuante* ou *representação real*, segundo a qual as soluções são arranjos de números reais. Essa representação é usualmente empregada quando os genes são distribuídos em um intervalo contínuo, em vez de um conjunto de valores discretos (Eiben e Smith, 2003).

2.2.2 *Fitness*

O valor do *fitness* de um indivíduo (seja um genótipo ou um cromossomo) é um número *positivo* que mede o quão adequada é a solução. Em problemas de otimização, o *fitness* pode ser o custo da solução. Se o problema for de minimização, as soluções de maior *fitness* são as de menor custo.

2.2.3 Pais, Operadores de Reprodução e Descendentes

Os AEs trabalham sobre um ou mais cromossomos a fim de gerar novas soluções, chamadas *descendentes*. Os operadores que trabalham sobre cromossomos, chamados operadores de reprodução, são a *recombinação* (também chamada *crossover*) e a *mutação*. Esses operadores fazem analogia aos principais mecanismos de evolução natural, ou seja, a recombinação e a mutação gênica (ver Seção 2.1). A recombinação é aplicada, em geral, a um par de cromossomos. Os indivíduos selecionados para o processo de recombinação são chamados *pais*. A mutação é aplicada a um simples cromossomo, modificando-o aleatoriamente. Esse operadores são descritos mais detalhadamente no Capítulo 4.

2.2.4 Geração e Seleção

A *geração* é uma iteração do AE, na qual os indivíduos da população atual são selecionados e recombinados e/ou mutados, gerando descendentes. Devido à criação de novos descendentes, o tamanho da população cresce; então um mecanismo de seleção controla esse tamanho.

A idéia básica da seleção é a seguinte: seja uma população de tamanho M e seja N_d o número de descendentes, então, para a próxima geração, são selecionados M novos indivíduos entre as $M + N_d$ possíveis soluções, ou entre somente os N_d novos indivíduos (N_d pode ser maior que M, ver Capítulo 3). Cada AE desenvolve, com base nesse princípio, uma estratégia de seleção.

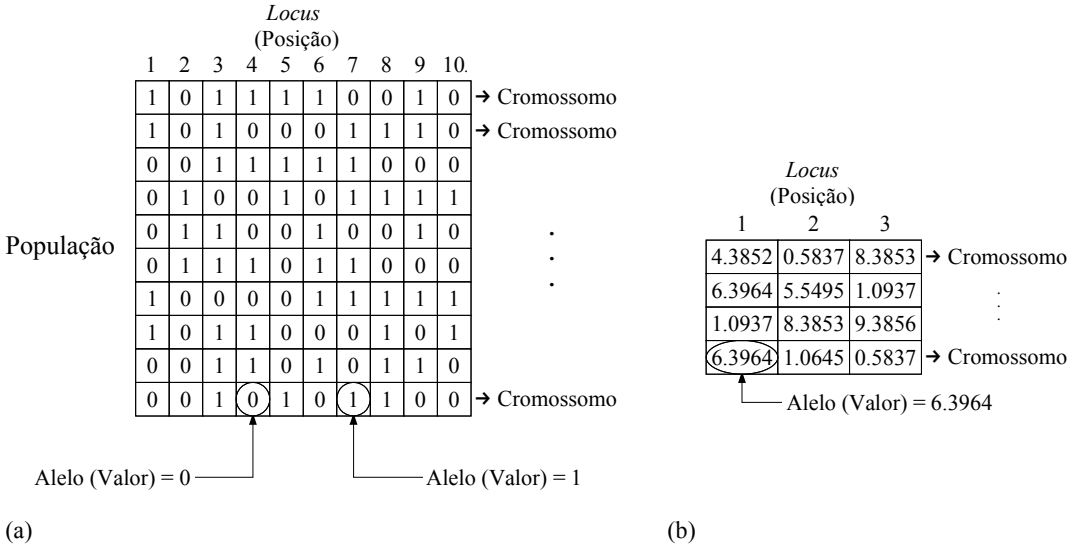

Figura 2.1: Dois exemplos de estrutura de dados e terminologia de um AE (Coello Coello et al., 2002): (a) cromossomos com 10 genes e alelos binários; (b) cromossomo com 3 genes e alelos correspondendo a valores reais.

Capítulo 3

Algoritmos Evolutivos

Ramos distintos de pesquisa desenvolveram técnicas computacionais que construíram os chamados AEs. A programação evolutiva, as estratégias evolutivas e os algoritmos genéticos foram fundamentais para o desenvolvimento dessa área de pesquisa e foram denominados AEs canônicos. A Seção 3.1 apresenta esses algoritmos. Outros AEs têm sido desenvolvidos buscando o aumento de desempenho com relação a vários aspectos. Buscando ilustrar a diversidade de AEs relvantes existentes na literatura, esse Capítulo também apresenta a programação genética (Seção 3.2.1), o micro-AG (Seção 3.2.2), os algoritmos de estimação de distribuição (Seção 3.2.3), além de uma introdução aos AEs para otimização multiobjetivo (Seção 3.2.4).

3.1 Algoritmos Canônicos

As principais características dos AEs canônicos são importantes para o entendimento de propostas de solução baseadas em conceitos evolutivos, bem como para o desenvolvimento de novos AEs. As subseções 3.1.1, 3.1.1 e 3.1.3 descrevem suscintamente os AEs canônicos.

3.1.1 Programação Evolutiva

A programação evolutiva (PE) foi proposta por Fogel (1962) com o objetivo de utilizar os conceitos de evolução no desenvolvimento Inteligência Artificial (IA).

Em PE, cada indivíduo da população é representado por uma máquina de estados finitos (MEF), que processa uma seqüência de símbolos. Durante a avaliação, os indivíduos são analisados por uma função de *payoff* de acordo com a saída da máquina e a saída esperada para solução do problema. A reprodução é feita apenas por operadores de mutação, sendo que todos os indivíduos da população atual geram novos descendentes. Esse processo caracteriza a chamada *reprodução assexuada*. Na seleção de indivíduos para a próxima geração, os descendentes competem com os μ pais e somente os indivíduos com maior *fitness* (no caso, os de maior *payoff* entre os $\mu + \lambda$ indivíduos) sobrevivem.

A PE garante que todos os indivíduos produzirão novos descendentes e, somente, os melhores indivíduos entre os atuais e os descendentes sobrevivem. O domínio total dos melhores indivíduos é chamado *elitismo total* (Kuri-Morales e Gutiérrez-García, 2001; Kuri-Morales, 2004). O elitismo mais utilizado garante a sobrevivência apenas dos k-melhores indivíduos, $k < N$, onde N é o tamanho da população. O elitismo total, no entanto, pode diminuir significativamente a diversidade de indivíduos, podendo estagnar em ótimos locais e/ou aumentar o tempo de convergência do algoritmo (De Jong, 2006).

3.1.2 Estratégias Evolutivas

As estratégias evolutivas (EEs) foram desenvolvidas com o objetivo de solucionar problemas de otimização de parâmetros. Foram propostas originalmente por Rechenberg (1965) e Schwefel (1975), na década de 1960, que desenvolveram a chamada $(1 + 1)$-EE em que um pai gera um único descendente (reprodução assexuada) e ambos competem pela sobrevivência.

Nas EEs, cada gene no cromossomo representa uma dimensão do problema, sendo que o alelo é representado em ponto flutuante. Os cromossomos são compostos por dois *arrays*, um com valores para cada dimensão e outro com o desvio padrão desses valores. A geração de um novo indivíduo é feita por meio da aplicação de um operador de mutação, com distribuição de probabilidade Gaussiana, com média zero e com desvio padrão do gene correspondente no pai. As EEs também utilizam elitismo completo.

O modelo original de EE possui convergência lenta (De Jong, 2006). Por isso, foram desenvolvidos outros modelos denominados respectivamente (μ, λ)-EE e $(\mu + \lambda)$-EE. No primeiro, μ pais morrem e sobrevivem λ descendente, sem competição entre os μ indivíduos já existentes com os λ novos indivíduos. No segundo modelo, sobrevivem apenas μ indivíduos selecionados entre os μ indivíduos atuais e os λ novos indivíduos gerados.

3.1.3 Algoritmos Genéticos

Os algoritmos genéticos (AGs) foram propostos por Holland e seus alunos na década de 1970. Holland estudou a evolução natural considerando esta um processo *robusto*, simples e poderoso, que poderia ser adaptado para obtenção de soluções computacionais eficientes para problemas de otimização. O conceito de robustez relaciona-se ao fato de os AGs, independentemente das escolha dos parâmetros iniciais, em geral, produzirem soluções de qualidade (Goldberg, 1989, 2002). O principal diferencial dos AGs é a criação de descendente pelo operador de recombinação (De Jong, 2006).

Além disso, a utilização de operadores de mutação e recombinação equilibra dois objetivos aparentemente conflitantes: o *aproveitamento das melhores soluções* e a *exploração do espaço de busca*. O processo de busca é, portanto, multidimensional, preservando soluções candidadas e provocando a troca de informação entre as soluções exploradas (Michalewicz, 1996; Von Zuben, 2000).

A seguir, mostram-se os principais passos de um AG, baseando-se em Michalewicz (1996):

- Durante a iteração gen, um AG mantém uma população de soluções potenciais $P(gen) = \{x_1^{gen}, \ldots, x_n^{gen}\}$;
- Cada indivíduo x_i^{gen} é avaliado produzindo uma medida de aptidão, ou *fitness*;
- Novos indivíduos são gerados a partir de μ indivíduos da população atual, os quais são selecionados para reprodução por um processo que tende a escolher indivíduos de maior *fitnes*;
- Alguns indivíduos sofrem alterações, por meio de *recombinação* e *mutação*, formando novas soluções potenciais;
- Dentre as soluções antigas e novas ($\mu + \lambda$), são selecionados indivíduos (*sobreviventes*) para a próxima geração ($gen + 1$);
- Este processo é repetido até que uma condição de parada ($cond?$) seja satisfeita. Essa condição pode ser um nível esperado de adequação das soluções ou um número máximo de iterações.

A Figura 3.1 apresenta um diagrama de fluxo do AG descrito nesta Seção.

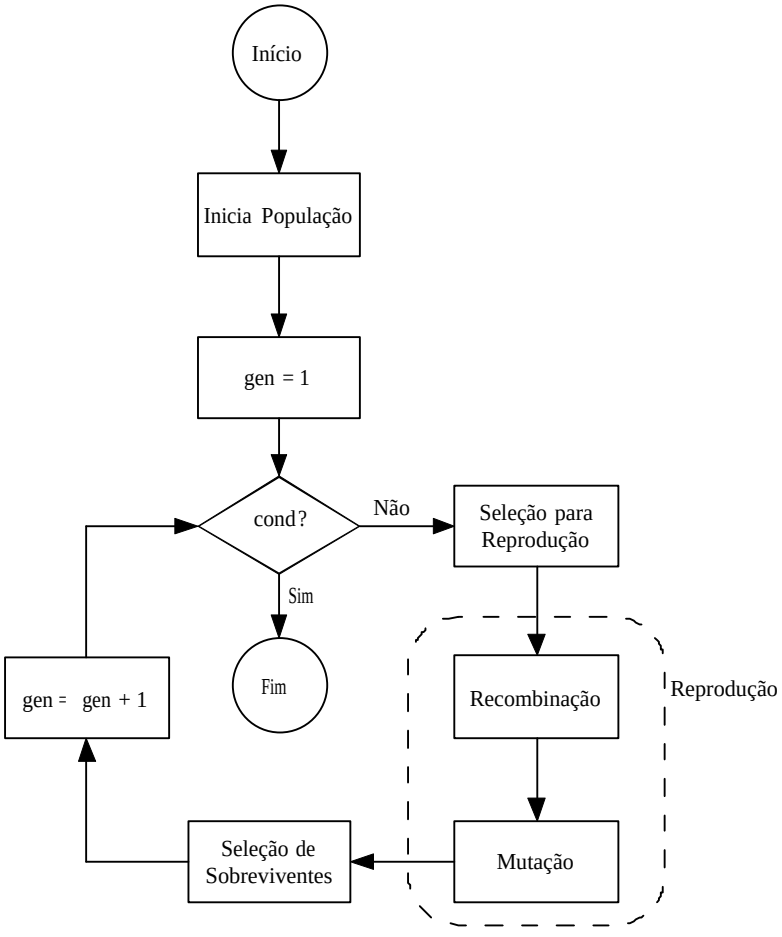

Figura 3.1: Diagrama de fluxo de um AG típico.

Nos AGs canônicos (Goldberg, 1989), as soluções candidatas são codificadas em *arrays* binários de tamanho fixo. A motivação para o uso de codificação binária é oriunda da *teoria dos esquemas* (Holland, 1992) utilizada com sucesso para explicar por que os algoritmos genéticos funcionam. Holland (1992) argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao AG e apresenta uma análise qualitativa baseada no *Teorema dos Esquemas* d que um alfabeto binário poderia maximizar esse paralelismo.

Entretanto, em diversas aplicações práticas a utilização de codificação binária leva a um desempenho insatisfatório. Em problemas de otimização numérica com parâmetros reais, algoritmos genéticos com representação inteira ou em ponto flutuante freqüentemente apresentam desempenho superior à codificação binária (Deb, 2001; Michalewicz, 1996).

Segundo Michalewicz (1996) e Deb (2001), a representação binária apresenta desempenho pobre se aplicada a problemas numéricos com alta dimensionalidade ou se alta precisão numérica é requerida. Além disso, descrevem simulações computacionais comparando o desempenho de AGs com codificação binária e com ponto flutuante. Os resultados apresentados mostram uma clara superioridade da codificação em ponto flutuante para os problemas estudados.

Em contrapartida, Fogel (1994) argumenta que o espaço de busca por si só (sem levar em conta a escolha da representação) não determina a eficiência do AG. Espaços de busca de dimensão elevada podem às vezes ser explorados eficientemente, enquanto que espaços de busca de dimensão reduzida podem apresentar dificuldades significativas. Concorda, entretanto, que a maximização do paralelismo implícito nem sempre produz um desempenho ótimo (Von Zuben, 2000).

O AG canônico utiliza um esquema de seleção de indivíduos para a próxima geração chamado *método da roleta* (Goldberg, 1989). Esse método é aplicado para selecionar λ indivíduos a partir de μ pais, mantidos em uma lista de reprodutores. Um implementação comum deste algoritmos calcula uma lista de valores $[a_1, a_2, \ldots, a_\mu]$, de modo que $a_i = \sum_1^i \mathcal{P}(i)$, onde $\mathcal{P}(i)$ é a probabilidade proporcional ao *fitness* de um indivíduo i passar para a próxima geração. O Algoritmo 2 mostra o pseudocódigo do método da roleta.

O desempenho de AGs pode, em muitos casos, ainda ser melhorado forçando a escolha do melhor indivíduo encontrado em todas as gerações do algoritmo. Outra opção é simplesmente manter sempre o melhor indivíduo da geração atual na geração seguinte, estratégia essa conhecida como *seleção elitista* (Fogel, 1994; Michalewicz, 1996).

Outro exemplo de mecanismo de seleção é a seleção baseada em *rank* (Michalewicz, 1996). Esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não lineares para determinar a probabilidade de seleção. Uma forma de implementação desse mecanismo é simplesmente passar os n melhores indivíduos para a próxima geração (Von Zuben, 2000).

Existe ainda a *seleção por torneio*, pela qual um subconjunto da população com k indivíduos é sorteado e os melhores indivíduos desse grupo são selecionados para decidir qual irá reproduzir. Em geral, utiliza-se *torneio de 2* ($k = 2$), isto é, dois indivíduos (obtidos aleato-

Algoritmo 2: Pseudocódigo do método da roleta.

Entrada: μ, `pais`.
Saída: `lista_reprodutores`.

```
1  cont ← 1;
2  a_i ← Σ_1^i P(i);
3  enquanto cont ≤ μ faça
4      Obter um valor aleatório r de probabilidade uniforme em [0, 1];
5      i ← 1;
6      enquanto a_i < r faça
7          i ← i + 1;
8          a_i ← a_i + P(i);
9      fim
10     lista_reprodutores[cont] ← pais[i];
11     cont ← cont + 1;
12 fim
```

riamente da população) competem entre si e o ganhador (o de melhor *fitness*) torna-se um dos pais. O pseudocódigo do Algoritmo 3 ilustra esse procedimento para selecionar μ pais.

Algoritmo 3: Pseudocódigo do algoritmo de seleção por torneio.

Entrada: População, μ.
Saída: `lista_reprodutores`.

```
1  cont ← 1;
2  enquanto cont ≤ μ faça
3      Obter k indivíduos aleatoriamente, com ou sem reposição;
4      Selecionar, com relação ao fitness o melhor indivíduo dentre k;
5      Denotar esse indivíduo por i;
6      lista_reprodutores[cont] ← i;
7      cont ← cont + 1;
8  fim
```

Uma importante propriedade da seleção por torneio que esta não depende de um conhecimento global da população. Além disso, essa seleção não leva em consideração o *rank* que o indivíduo ocupa na população, permitindo uma seleção com menos tendências (Eiben e Smith, 2003).

3.2 Outros Algoritmos Evolutivos

Mais recentemente, diversos aspectos complexos em problemas de otimização e aprendizado de máquina têm sido pesquisados em CE. Esses estudos têm buscado o desenvolvimento de novos AEs mais eficientes. As Subseções seguintes descrevem alguns AEs que apresentam desempenho relativamente melhor em seus diferentes propósitos.

3.2.1 Programação Genética

A programação genética (PG) pode ser vista como uma extensão dos AGs. Proposta por Koza (1989) a PG difere dos AEs canônicos devido a sua representação, seus operadores de reprodução e seus métodos de avaliação da função de avaliação (Koza, 1992, 1994; Koza et al., 1999, 2003). Introduzida para solucionar problemas de aprendizado de máquina (Rezende, 2003), a PG busca a construção automática de programas de computadores. Os indivíduos são codificados na forma de árvores, onde cada nó folha contém constantes, variáveis ou parâmetros para a execução de procedimentos e funções. Os nós internos contém operações primárias.

Os operadores de reprodução utilizados são operadores de recombinação e mutação específicos para representações por árvores. Na recombinação, partes das árvores são trocadas, o ponto de corte na árvore é escolhido de forma a evitar a criação de operações inválidas. Na mutação o valor de um nó ou subárvore é alterado. Se o nó escolhido para a mutação for um nó interno, este será alterado para ter uma nova operação ou função. No caso de mutação de subárvore, a subárvore selecionada é substituída por uma nova subárvore gerada aleatoriamente.

O processo de avaliação ocorre por meio da execução do programa representado pela árvore do indivíduo. Se este resolver o problema proposto ou se aproximar da resposta correta terá um valor de aptidão elevado; caso contrário, sua aptidão será baixa. Geralmente, os algoritmos de PG utilizam somente o operador de recombinação no processo de busca pelas melhores soluções.

3.2.2 Micro-AG

Os micro-AGs são variações de AGs desenvolvidos para de evoluir eficientemente soluções utilizando populações com poucos indivíduos. O primeiro trabalho envolvendo micro-AGs foi proposto por Krishnakumar (1989). Nesse modelo uma pequena população é gerada e evoluída até que todos os indivíduos tenham genótipos idênticos ou, pelo menos, muito similares. O melhor indivíduos é, então, transferido para uma nova população; os demais novos indivíduos são gerados aleatoriamente. Em geral utiliza-se alta taxa de recombinação (próxima de 1) e baixa de mutação (em muitos casos 0).

Micro-AGs apresentam grande importância em diversas aplicações, em especial projetos onde a disponibilidade de recursos computacionais é limitada, como é o caso da área de Robótica Evolutiva (Carvalho et al., 2004), onde cada robô é um indivíduo; como é inviável construir um grande números de robôs, a população deve ser pequena. Aplicações de tempo real também podem ser beneficiadas com a utilização de micro-AGs, uma vez que populações pequenas requerem menos tempo computacional para serem geradas (Delbem et al., 2005).

3.2.3 Algoritmos de Estimação de Distribuição

Os algoritmos de estimação de distribuição (AEDs) (Larraeñaga e Lozano, 2001; Paul e Iba, 2003) foram intensamente pesquisados nos últimos anos. OS AEDs foram introduzidos

por Mhlenbein e Paaβ (1996), incorporando técnicas de aprendizagem automatizada de correlações entre as variáveis codificadas na solução do problema. Os indivíduos, ou parte deles, são considerados amostras. Então, um modelo probabilístico de distribuição é ajustado à amostra. Em seguida, pode-se determinar os melhores novos indivíduos de acordo com o modelo ajustado. Este processo torna desnecessária a especificação de determinados parâmetros dos AEs canônicos, como a taxa de mutação e de recombinação.

A desvantagem dessas técnicas é que, em geral, os modelos probabilísticos utilizados para guiar a busca são complexos e de alto custo computacional. Novas abordagens têm sido propostas para solucionar esse problema, com destaque os algoritmos UMDA (do inglês "univariante marginal distribution algorithm") (Mhlenbein, 1998) e BOA (do inglês "Bayesian optimization algorithm") (Pelikan et al., 1999). Mais recentemente, Melo et al. (2007) desenvolveu um AED utilizando modelos simples que apresentam desempenho adequado para uma diversidade de problemas.

Um AED típico está representado no fluxograma do Figura 3.2. Paul e Iba (2003) argumentam que é razoável utilizar AEDs no lugar de outros AEs, como os AGs. Porém, a estimação da distribuição de probabilidade associada ao conjunto de indivíduos selecionados é um gargalo dessa técnica, não existindo métodos simples para realizar cálculos em modelos complexos.

3.2.4 Algoritmos Evolutivos para Otimização Multiobjetivo

Problemas de otimização multiobjetivo têm despertado grande interesse na área de Otimização. Nesses problemas, a qualidade da solução é definida com base na sua adequação em relação a diversos objetivos possivelmente conflitantes (Deb, 2001; Eiben e Smith, 2003). Na prática, os métodos de solução buscam reduzir esses problemas a outros com apenas um objetivo e depois buscam uma solução. Uma classe de métodos bastante utilizada nesse contexto é a dos *métodos baseados em pesos* ou simplesmente *método de pesos*, onde são atribuídos pesos às diferentes funções objetivo, podendo ser é realizado um processo de otimização da função de pesos (Arenales et al., 2007).

Esses métodos, no entanto, são dependentes da escolha adequada dos pesos. Isso, em muitos casos, implica um conhecimento prévio dos intervalos correspondentes aos pesos mais adequados. Por essa razão, métodos que tentam encontrar soluções que apresentam um compromisso com os vários objetivos sem a utilização de pesos passaram a ser explorados (Eiben e Smith, 2003). Nesse caso, não existe somente uma solução para o problema, mas sim um conjunto de soluções ótimas, denominado conjunto de *Pareto ótimo* ou *fronteira de Pareto* (Deb, 2001).

O primeiro AE para otimização multiobjetivos (AEOM) desenvolvido foi proposto por Schaffer (1985) e denominado VEGA (do inglês, "Vector Evaluated Genetic Algorithm"). Este algoritmo é um AG modificado que avalia cada objetivo separadamente. Um dos problemas do algoritmo proposto por Schaffer é que as soluções da fronteira obtidas, em geral, possuem baixa diversidade.

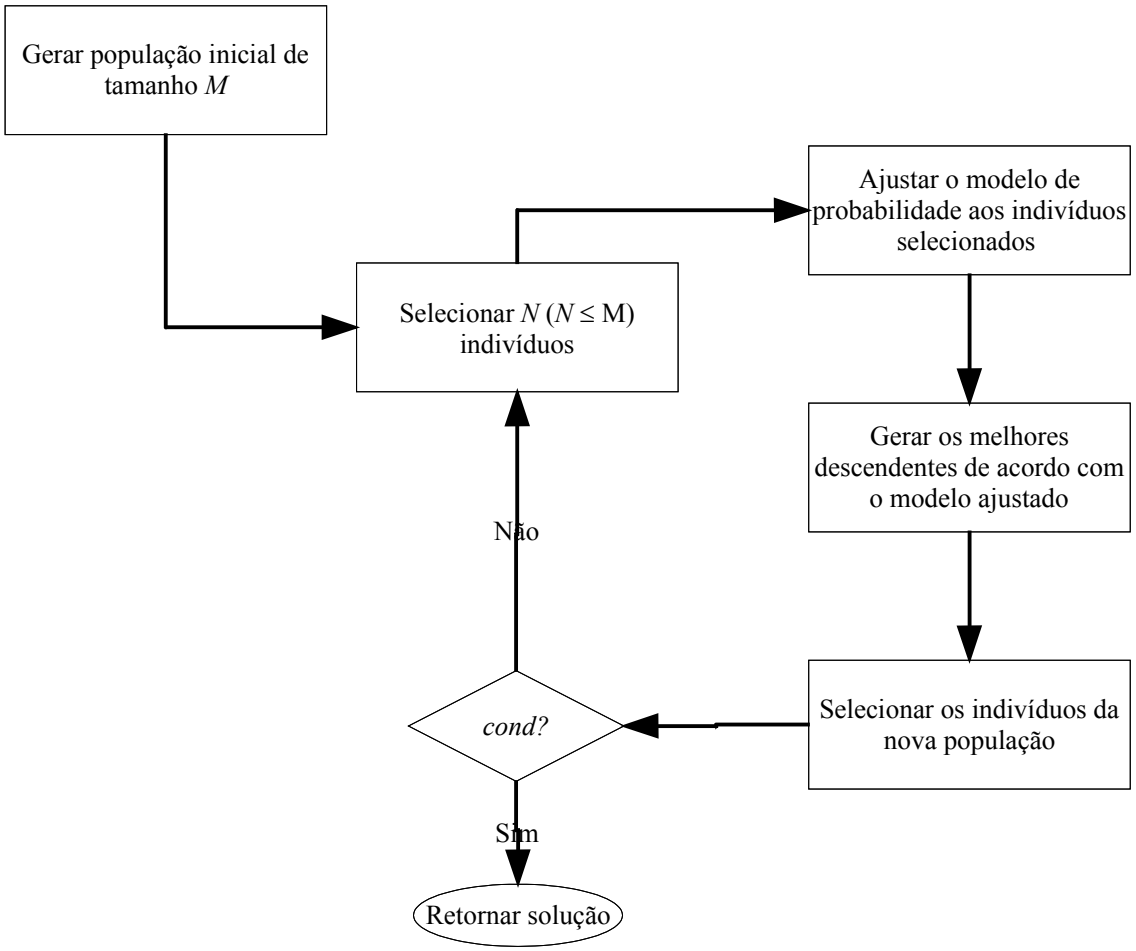

Figura 3.2: Diagrama de fluxo simplificado de um AED.

Goldberg (1989) criou em procedimento que ordena as soluções baseado no conceito de *dominância* que fornece um valor de aptidão para uma solução proporcional ao número de soluções que esta domina, i.e., não perde em nenhum objetivo e é melhor em pelo menos um. Com isso as soluções com maior número de dominados possuem maior aptidão e, assim, terão uma maior quantidade de cópias na lista de soluções. Com o objetivo de manter a diversidade das soluções, Goldberg sugeriu a utilização de um método de compartilhamento que calcula o nicho de cada solução dentro da fronteira que a solução pertence. Com base nas idéias iniciais de Goldberg, foram propostos vários modelos de AEOMs.

A principal diferença entre os AEs tradicionais e os AEOMs é o operador de seleção, dado que a comparação entre duas soluções deve realizar-se de acordo com o conceito de dominância. Em algumas proposta, como MOGA (Fonseca e Fleming, 1993) e SPEA (Zitzler e Thiele, 1998), o valor de aptidão é proporcional à dominância da solução. Os outros métodos, como NPGA (Horn et al., 1994), utilizam a dominância e não calculam um valor de aptidão.

Os modelos de AEMO são usualmente classificados em dois grupos (Deb, 2001; Eiben e Smith, 2003):

1. *Não elitistas*: compreendem os modelos que como o próprio nome indica, não utilizam nenhuma forma de elitismo nas suas interações;

2. *Elitistas*: compreendem os modelos que empregam alguma forma de elitismo. Por exemplo, como o SPEA e o PESA (Corne et al., 2000), utilizam uma população externa para armazenar as soluções não dominadas encontradas até o momento. O NSGA-II (Deb et al., 2000) combina a população atual com a população gerada e preserva as melhores soluções de ambas. Estudo realizado por Zitzler et al. (2000) concluiu que o elitismo melhora as soluções encontradas por um modelo de AEMO. A Tabela 3.2.4 apresenta os principais modelos de AEMO e seus autores.

Tabela 3.1: Diferentes modelos de AEMO.

Sigla	Nome do Modelo	Autores
VEGA	*Vector Evaluated Genetic Algorithm*	(Schaffer, 1985)
WBGA	*Weight Based Genetic Algorithm*	(Hajela e Lin, 1992)
MOGA	*Multiple Objective Genetic Algorithm*	(Fonseca e Fleming, 1993)
NSGA	*Non-Dominated Sorting Genetic Algorithm*	(Srinivas e Deb, 1994)
NPGA	*Niched-Pareto Genetic Algorithm*	(Horn et al., 1994)
PPES	*Predator-Prey Evolution Strategy*	(Laumanns et al., 1998)
REMOEA	*Rudoph's Elitist Multi-Objective Evolutionay Algorithm*	(Rudolph, 2001)
NSGA-II	*Elitist Non-Dominated Sorting Genetic Algorithm*	(Deb et al., 2000)
SPEA, SPEA-2	*Strenght Pareto Evolutionary Algorithm 1 e 2*	(Zitzler e Thiele, 1998), (Zitzler et al., 2001)
TGA	*Thermodynamical Genetic Algorithm*	(Kita et al., 1996)
PAES	*Pareto-Archived Evolutionary Strategy*	(Knowles e Corne, 1999)
MONGA-I, MONGA-II	*Multi-Objective Messy Genetic Algorithm*	(Veldhuizen, 1999)
PESA-I, PESA-II	*Pareto Envelope-Base Selection Algorithm*	(Corne et al., 2000), (Corne et al., 2001)

Capítulo 4

Operadores de Reprodução

A reprodução, seguindo uma nomenclatura utilizada pela Biologia, pode ocorrer de duas formas:

1. *Assexuada*, ou seja, um indivíduo sozinho gera um descendente, utilizando *mutação*;
2. *Sexuada*, isto é, há a participação de dois indivíduos na geração de um ou mais descendentes por meio do processo de *recombinação*.

As próximas Seções descrevem os operadores operadores correspondentes aos processos de mutação e recombinação, bem como suas principais variações.

4.1 Mutação

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. Com esse operador, um indivíduo gera uma cópia de si mesmo, a qual pode sofrer alterações. A probabilidade de ocorrência de mutação em um gene é denominada *taxa de mutação*. Usualmente, são atribuídos valores pequenos para a taxa de mutação, uma vez que esse operador pode gerar um indivíduo potencialmente pior que o original.

Considerando a codificação binária (ver Seção 2.2.1), o operador de mutação padrão simplesmente troca o valor de um gene em um cromossomo (Goldberg, 1989). Assim, se o alelo de um gene selecionado é 1, o seu valor passará a ser 0 após a aplicação da mutação (Figura 4.1)

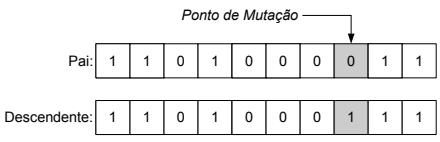

Figura 4.1: Representação gráfica do operador de mutação.

No caso de problemas com codificação em ponto flutuante, os operadores de mutação mais populares são a *mutação uniforme* e a *mutação Gaussiana* (Bäck et al., 2000; Deb, 2001). O operador para mutação uniforme seleciona aleatoriamente um componente $k \in \{1, 2, \ldots, n\}$ do cromossomo $x = [x_1, \ldots, x_k, \ldots, x_n]$ e gera um indivíduo $x' = [x_1, \ldots, x'_k, \ldots, x_n]$, onde x'_k é um número aleatório (com distribuição de probabilidade uniforme) amostrado no intervalo $[L_I, L_S]$, sendo L_I e L_S, respectivamente, os limites inferior e superior para o valor do alelo x_k. No caso da mutação Gaussiana, todos os componentes de um cromossomo x são modificados pela seguinte expressão:

$$x' = x + N(0, \sigma),$$

onde $N(0, \sigma)$ é um vetor de variáveis aleatórias Gaussianas independentes, com média zero e desvio padrão σ.

4.2 Recombinação

O operador de recombinação é o mecanismo de obtenção de novos indivíduos pela troca ou combinação dos alelos de dois ou mais indivíduos. É o principal operador de reprodução dos AGs (Goldberg, 1989). Fragmentos das características de um indivíduo são trocadas por um fragmento equivalente oriundo de outro indivíduo. O resultado desta operação é um indivíduo que combina características potencialmente melhores dos pais.

Um tipo bastante comum de recombinação é a de 1-ponto. Nessa operação, seleciona-se aleatoriamente um ponto de corte nos cromossomos, dividindo este em uma partição à direita e outra à esquerda do corte. Cada descendente é composto pela junção da partição à esquerda (direira) de um pai com a partição à direita (esquerda) do outro pai (Figura 4.2).

Outra recombinação, de n-pontos, divide os cromossomos em n partições, as quais são recombinadas. A Figura 4.3 ilustra um exemplo com 2 pontos de corte. Neste caso, o Filho 1 (Filho 2) recebe a partição central do Pai 2 (Pai 1) e as partições à esquerda e à direita dos corte do Pai 1 (Pai 2).

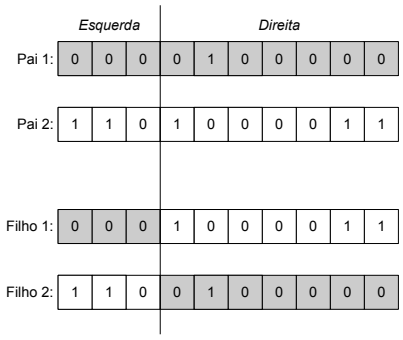

Figura 4.2: Recombinação de 1-ponto.

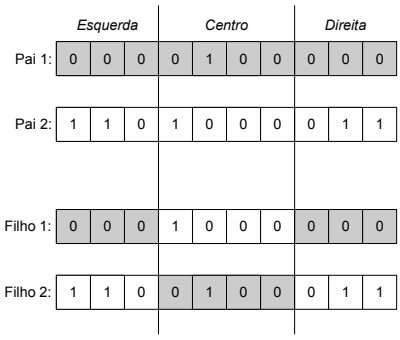

Figura 4.3: Recombinação de 2-pontos.

Outro tipo de recombinação muito comum é a *recombinação uniforme*, que considera cada gene independentemente, escolhendo de qual pai o gene do filho será herdado. Em geral, cria-se uma lista de variáveis aleatórias de distribuição uniforme em $[0, 1]$. Para cada posição, se o valor da variável aleatória for inferior a um dado $\mathcal{P}$ (usualmente $0,5$), o gene será oriundo do Pais 1; caso contrário, virá do Pai 2. O segundo filho é gerado pelo mapeamento inverso (ver Figura 4.4).

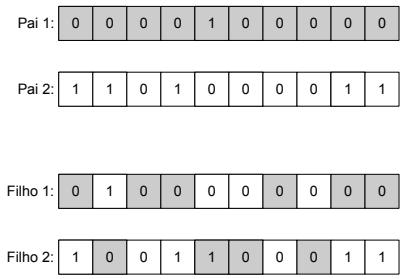

Figura 4.4: Recombinação uniforme. Nesse exemplo, adaptado de Eiben e Smith (2003), foram utilizadas as variáveis aleatórias $[0.31, 0.74, 0.10, 0.42, 0.89, 0.78, 0.21, 0.51, 0.36, 0.21]$ uniformemente distribuídas em $[0, 1)$.

Em representações de ponto flutuante (ver Seção 2.2.1), verificou-se que há duas categorias de operadores de recombinação: *a recombinação discreta* e a recombinação aritmética, as quais são explicadas nas Subseções 4.2.1 e 4.2.2.

4.2.1 Recombinação Discreta

Essa categoria de recombinação procura fazer analogia com o processo de recombinação em representação binária, de modo que cada cromossomo é um conjunto de números reais. A desvantagem da recombinação discreta está no fato de não inserir novos valores no cromossomo, uma vez que gera apenas combinações de vetores. A tarefa de inserir novos genes é atribuída apenas à mutação (Eiben e Smith, 2003).

4.2.2 Recombinação Aritmética

Proposta por Michalewicz e Janikow (1991), a recombinação aritmética (também chamada *recombinação intermediária*) cria novos alelos nos descendentes com valores intermediários aos encontrados nos pais. Define uma combinação linear entre dois cromossomos x e y, de modo a gerar um descendente z, da seguinte forma:

$$z = \alpha x + (1 - \alpha)y, \tag{4.1}$$

onde α é um número aleatório pertencente ao intervalo $[0, 1]$. Geralmente, no entanto, é utilizado um valor fixo para α (como, por exemplo, $\alpha = 0,5$, caracterizando a chamada *recombinação aritmética uniforme*). Esse operador é particularmente apropriado para problemas de otimização numérica com restrições, onde a região factível é convexa[1]. A combinação linear entre x e y pertencentes à região factível resulta em elementos também pertencentes à região factível.

Diversos operadores de recombinação foram propostos seguindo o princípio descrito acima. Na *recombinação simples*, um ponto de corte k é definido; o Filho 1 é construído com os primeiros k genes do Pai 1 e com os demais $n-k$ genes restantes obtidos pela média aritmética dos $n-k$ genes dos Pai 1 e Pai 2 (Figura 4.5). Na *recombinação individual*, apenas o alelo k, escolhido aleatoriamente, é recombinado pela média aritmética dos alelos k dos Pais 1 e 2 (Figura 4.6).

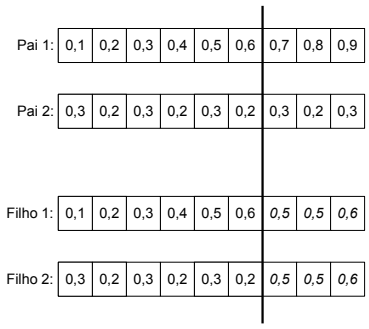

Figura 4.5: Recombinação simples, com $\alpha = 0,5$.

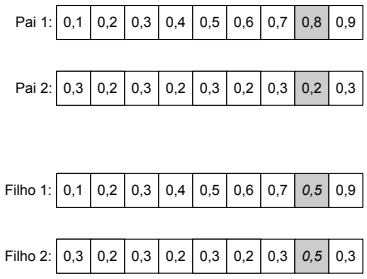

Figura 4.6: Recombinação individual, com $\alpha = 0,5$.

A *recombinação total*, uma das mais utilizadas, realiza uma ponderação de todos os alelos dos pais para cada gene:

$$z^1 = \alpha x + (1 - \alpha)y \tag{4.2}$$

$$z^2 = \alpha y + (1 - \alpha)x. \tag{4.3}$$

A Figura 4.7 ilustra esse processo para $\alpha = 0,5$. Este valor sempre produz descendentes iguais.

4.2.3 Blend Crossover

Uma abordagem alternativa de recombinação para valores reais é a *recombinação da mistura* ou *blend crossover* (BLX-α), desenvolvido por Eshelman e Schaffer (1993) com base em um estudo organizado por Goldberg (1991). Para dois cromossomos pais x e y, assumindo $x \leq y$, o BLX-α seleciona aleatoriamente uma solução pertencente ao intervalo $[x_i - \alpha(y_i - x_i), y_i - \alpha(x_i - y_i)]$, onde α é um número real entre 0 e 1. Assim, para $\alpha = 0.0$, por exemplo,

[1] A descrição e análise da convexidade de uma região factível pode ser encontrada em (Bazaraa et al., 2006).

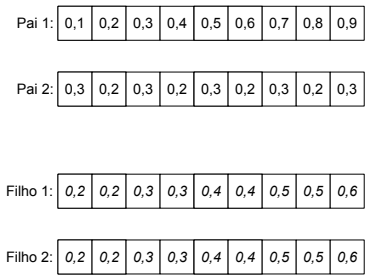

Figura 4.7: Exemplo de recombinação total para $\alpha = 0.5$.

esse operador cria uma solução no intervalo $[x_i, y_i]$. Esse procedimento é repetido para cada *locus* gerando uma nova solução.

É importante observar que a localização da solução gerada pelo BLX-α depende da diferença entre as soluções pais. Por exemplo, se a diferença entre os pais for pequena, a diferença entre os filhos também será. Essa propriedade permite afirmar que, se a diversidade na população for grande, é esperada uma grande diversidade na população da geração seguinte. Isso garante uma exploração de todo espaço de busca no início do processo evolutivo, focando a busca em uma determinada região à medida que as soluções convergem (Deb, 2001).

4.2.4 Recombinação com Múltiplos Pais

Além dos operadores de reprodução descritos neste Capítulo, assexuada e sexuada, é possível explorar um esquema de reprodução inexistente em sistemas biológico: a *recombinação com múltiplos pais*. Essa estratégia foi proposta logo no início dos estudos computacionais com AEs, na década de 1960 (Eiben et al., 1995). Tecnicamente, este tipo de recombinação amplifica os efeitos da recombinação. Esse operadores podem ser categorizados de acordo com o mecanismo de combinação empregado (Eiben et al., 1995; Eiben e Smith, 2003):

1. Recombinação baseada na seqüência de alelos (chamada *eleição p-sexual*), generalizando a recombinação uniforme;

2. Recombinação baseado em segmentação e recombinação dos pais (chamada *recombinação diagonal*), generalizando a recombinação de n-pontos;

3. Recombinação baseado em operações numéricas sobre alelos reais (chamada *recombinação massiva*), generalizando os operadores de recombinação aritmética.

Não há estudos teóricos conclusivos sobre as vantagens da utilização de vários pais no processo de recombinação. No entanto, resultados experimentas demonstram vantagem dessa técnica em uma grande variedade de problemas (Eiben e Schippers, 1996; Eiben e Smith, 2003).

4.3 Operadores de Reprodução para Permutação

Uma classe importante de problemas do mundo real consiste em decidir a ordem na qual uma seqüência de eventos deve ocorrer (Eiben e Smith, 2003). Um forma bastante natural de representar tais problemas é utilizando *permutações*. Uma conseqüência imediata desse tipo de problema é a necessidade de buscar novos operadores, de modo a trabalhar com soluções factíveis. Em permutações, o operador de mutação deve garantir que o novo indivíduo obtido também represente uma permutação válida. Uma primeira classe de operadores de mutação para permutações escolhe 2, 3 ou k pontos de corte no cromossomo e tenta trocar os segmentos entre esses pontos. No entanto, os operadores dessa classe são custoso (Bäck et al., 2000). Para problemas onde a permutação representa uma lista de prioridades, foram propostos operadores que fazem a troca de duas ou mais posições escolhidas de forma aleatória.

No caso da recombinação, também é necessário criar uma permutação válida. Com esse objetivo, foram desenvolvidos os operadores apresentados a seguir, nos quais são consideradas as seguintes permutações como cromossomos pais:

- Pai 1 = | C | E | D | B | F | A |;
- Pai 2 = | E | B | A | C | D | F |.

4.3.1 Crossover de Ordem

Os novos indivíduos do crossover de ordem, conhecido como OX, são obtidos da seguinte forma: selecionam-se dois pontos de corte que definem uma subseqüência entre esses pontos. Essa subseqüência é copiada do Pai 1 (Pai 2) para o Filho 1 (Filho 2) na mesma posição em que os genes estão no pai. Para preencher as posições restantes do filho, copia-se do Pai 2 (Pai 1) o primeiro gene posicionado a partir do segundo ponto de corte e que ainda não está presente no filho para a primeira posição não preenchida no filho depois do segundo ponto de corte. Esse processo de cópia continua com o próximo gene de Pai 2 (Pai 1) utilizando este pai como uma lista circular até completar os genes do Filho 1 (Filho 2).

No exemplo, sejam os pontos de corte entre as posições 3 e 4, e 7 e 8. Primeiramente os filhos são formados somente pelas subseqüências entre os pontos de corte, ou seja:

- Filho 1 = | X | X | D | B | X | X |
- Filho 2 = | X | X | A | C | X | X |

No passo seguinte, as demais posições dos filhos são definidas, a partir do segundo ponto de corte a partir do pai não utilizado para compor a subseqüência. Os seguintes filhos são obtidos:

- Filho 1 = | A | C | D | B | F | E |
- Filho 2 = | D | B | A | C | F | E |

4.3.2 Crossover Baseado em Posição

Para obter novos indivíduos também pode ser utilizado o crossover baseado em posição, chamado PBX, o qual começa pela seleção de um conjunto de posições aleatórias de um dos pais. Posteriormente, os genes das posições selecionadas do pai são copiados no filho exatamente na mesma posição em que estão no pai. Em seguida, os genes copiados são removidos do segundo pai, obtendo do segundo pai os genes que faltam no filho. Esses genes são copiados na ordem em que aparecem da esquerda para a direita nas posições vazias do filho. Para obter o outro filho, basta trocar a ordem dos pais (Michalewicz e Fogel, 2004).

Como exemplo, considere os pais definidos anteriormente e as posições 2, 4 e 5 escolhidas de forma aleatória e as posições começando em um. O primeiro filho é obtido inicialmente pela cópia das posições selecionadas do primeiro pai:

- Filho 1 = | X | E | X | B | F | X |

A seguir as posições vazias são preenchidas pelos genes do segundo pai ignorando os genes já inseridos no filho e assim obtém-se o seguinte indivíduo:

- Filho 1 = | A | E | C | B | F | D |

4.3.3 Crossover Baseado em Ordem

O operador de crossover baseado em ordem (OBX) é similar ao PBX. O processo inicia como em PBX, selecionando um conjunto de posições de forma aleatória. Os genes correspondentes às posições selecionadas do segundo pai são localizados no primeiro pai. Na etapa seguinte, esses genes são reordenados para aparecerem na mesma ordem presente no segundo pai, no entanto, mantendo as posições do primeiro pai. A seguir, os genes reordenados são copiados para o filho nas mesmas posições do primeiro pai. As demais posições são copiadas diretamente do primeiro pai. Novamente, para obter outro filho, o processo é igual somente trocando o primeiro com o segundo pai (Bäck et al., 2000).

Considere novamente, para exemplo, os pais definidos anteriormente e as posições 2, 4 e 5, que em Pai 1 correspondem aos genes 2, 3 e 4, respectivamente, os quais estão em Pai 1, respectivamente nas posições 4, 1 e 3. Reordenando para que os genes estejam na mesma ordem em que se apresentam em Pai 2 e respeitando as posições em Pai 1 temos a seguinte configuração inicial para o primeiro filho

- Filho 1 = | B | X | C | D | X | X |

Finalmente, as demais posições de Filho 1 são preenchidas com os genes de Pai 1, como segue:

- Filho 1 = | B | E | C | D | F | A |

4.3.4 Crossover cíclico

A permutação também pode ser realizada por um crossover circular, chamado CX. Nesse, dados dois pais, a primeira posição no filho recebe o primeiro gene do primeiro pai. O próximo gene do filho é obtido do segundo pai. Este gene é copiado para o filho na posição em que este mesmo gene aparece no primeiro pai. O algoritmo segue copiando genes do segundo pai na posição correspondente no primeiro pai até que se forme um ciclo, ou seja, até que na posição correspondente no segundo pai encontra-se um gene já inserido no filho. Neste caso, se restarem posições não preenchidas no filho, essas são preenchidas da esquerda para a direita no *array* do filho com os genes do segundo pai na ordem relativa que aparecem neste pai. Trocando-se o primeiro pai pelo segundo é possível obter outro descendente (Michalewicz e Fogel, 2004).

Por exemplo, começando com a primeira posição de Pai 1, definido anteriormente, tem-se o gene 3 que é copiado para a primeira posição do filho. O gene 3, corresponde ao gene 5, em Pai 2 que está na posição 2 de Pai 1, que é a posição utilizada para copiar o gene no filho. A seguir é considerado o gene correspondente a posição do gene 5 em Pai 1 no Pai 2 que retorna o gene 2. O gene 2 está na posição 4 de pai_1 e é copiado nesta posição para filho. A posição 4 em Pai 2 corresponde ao gene 3 e neste momento se forma o ciclo. Assim,

- Filho 1 = | C | E | X | B | X | X |

Agora, as posições não preenchidas são copiadas de Pai 2 e o filho resultante é

- Filho 1 = | C | E | A | B | D | F |

4.3.5 Crossover Parcialmente Mapeado

Uma variação do operador OX é o crossover parcialmente mapeado (PMX), pois ambos utilizam dois pontos aleatórios de corte para definir uma subseqüência que é copiada para os descendentes. As subseqüências de cada pai definem um mapeamento entre seus genes, de forma que um gene no primeiro pai na posição i é mapeado para o gene na posição i do segundo pai. O operador inicia copiando a subseqüência do primeiro pai para o filho. Os demais genes do filho são copiados das posições correspondentes dos genes do segundo pai, resolvendo cada conflito (gene do segundo pai que já foi inserido no filho) utilizando o gene correspondente dado pelo mapeamento. Para o outro filho, o processo é o mesmo só trocando o primeiro pai com o segundo pai (Michalewicz e Fogel, 2004; Bäck et al., 2000).

Por exemplo, com Pai 1 e Pai 2 definidos, escolhe-se de forma aleatória os pontos de corte como definidos para OX, ou seja, entre as posições 2 e 3, e 4 e 5. Estes pontos de corte definem o mapeamento de genes $4 \leftrightarrow 1$ e $2 \leftrightarrow 3$. Copia-se para Filho 1 a subseqüência definida entre os pontos de corte de Pai 1 e de Pai 2 os genes que não geram conflito com os genes copiados de Pai 1 obtendo:

- Filho 1 = | E | X | D | B | X | F |

Para os genes que geram conflito utiliza-se o mapeamento e obtém-se

- Filho 1 = | E | C | D | B | A | F |

Capítulo 5

Considerações Finais

Neste texto foram abordados os AEs, poderosas ferramentas aplicadas principalmente em problemas de otimização. Os AEs têm como base idéias evolucionistas, utilizando conceitos de Genética e dinâmica populacional.

Foram apresentados os três AEs canônicos, historicamente desenvolvidos de forma independente mas que mantêm idéias comuns. Tabela 5.1 apresenta um breve resumo dos pontos em que PE, EE e AG diferem entre si. Também foram descritas variações desses AEs, como a PG e o micro-AG, além dos AEDs e abordagens multiobjetivo.

Tabela 5.1: Diferenças fundamentais entre AEs canônicos.

AE	Representação	Operadores
PE	MEF	Mutação e seleção $(\mu + \lambda)$ ou (μ, λ)
EE	Ponto flutuante (dimensão do problema e desvio padrão)	Mutação e seleção $(\mu + \lambda)$
AG	Originalmente binária; ponto flutuante em algumas aplicações	Recombinação (principal), mutação e seleção aleatória

Deve-se observar que a principal característica dos AEs é a idéia de evolução de soluções em paralelo. Neste contexto, são incluídos algoritmos como: *swarm inteligence*, *ant colony optimization* (ACO)[1], *imune systems*, *learning classifier algorithms* (LCS), etc. Além disso, os AEs podem ser vistos como metaheurísticas[2], possibilitando uma interface (compartilhamento, idéias, hibridização) com algoritmos como GRASP, A-teams, VNS, entre outros. (Feo e Resende, 1995; Resende e Ribeiro, 2005; Ribeiro e Hansen, 2002) A interface amplia-se ao se considerar versões paralelizáveis dessas metaheurísticas, conforme apresentado em (Alba, 2005; Duni Ekisoglu et al., 2002; Resende e Ribeiro, 2005).

[1]ACO (Dorigo e Gambardella, 1997; Dorigo et al., 1999; Pilat e White, 2002) são AEs, no sentido em que evoluem soluções em paralelo, e também são AEDs, uma vez que utilizam um modelo probabilístico e matriz de ferormônios construída a partir da atividade de uma população de formigas.

[2]Uma metaheurística é um método heurístico para solução de uma classe geral de problemas computacionais utilizando a combinação de outras heurísticas (Blum e Roli, 2003).

Finalmente, é importante destacar os AEs são processos heurísticos, de modo podem não gerar, necessariamente, soluções ótimas. Além disso, muitos problemas de otimização encontrados na mundo real podem ser resolvidos por métodos exatos de forma eficiente. Por esse motivo, soluções baseadas em AEs devem ser utilizadas como um último recurso na resolução de problemas, ou seja, devem se aplicadas quando métodos de solução exatos sejam consideradas ineficientes.

Referências Bibliográficas

ALBA, E. *Parallel metaheuristics: A new class of algrothms*. Wiley Series on Parallel and Distributed Computing. Wiley, 2005.

AMABIS, J. M.; MARTHO, G. R. *Curso básico de biologia*, v. 3. São Paulo: Editora Moderna Ltda., 1985.

ARENALES, M. N.; ARMENTANO, V. A.; MORABITO, R.; YANASSE, H. H. *Pesquisa operacional*. Rio de Janeiro: Elsevier Editora Ltda., 2007.

BALLARD, D. H. *An introduction to natural computing*. MIT Press, 1999.

BAZARAA, M. S.; SHERALI, H. D.; SHETTY, C. M. *Nonlinear programming: Theory and algorithms*, cap. 2. 3 ed Hoboken, NJ: John Wiley & Sons, Inc., p. 39–95, 2006.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308, 2003.

BÄCK, T.; FOGEL, D. B.; MICHALEWICZ, T., eds. *Evolutionary computation 1: Basic algorithms and operators*. New York: Taylor & Francis Group, 2000.

CARVALHO, A. P. L. F.; DELBEM, A.; SIMÕES, E. V.; TELLES, G. P.; ROMERO, R. A. Computação bioinspirada. In: *Apostila de minicurso XXIII JAI - Jornada de Atualização em Informática*, Porto Alegre: Sociedade Brasileira de Computação, 2004, p. 50.

CASTRO, L. N. *Fundamentals of natural computing: Basic concepts,algorithms, and applications*. CRC Press LLC, 2006.

COELLO COELLO, C. A.; VAN VELDHUIZEN, D. A.; LAMONT, G. B. *Evolutionary algortihms for solving multi-objective problems*. Genetic Algorithms and Evolutionary Computation. New York, NY: Kluwer Academic, 2002.

CORNE, D.; JERRAM, N.; KNOWLES, J.; OATES, M. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In: SPECTOR, L.; GOODMAN, E.; WU, A.; LANGDON, W.; VOIGT, H.; GEN, M.; SEN, S.; DORIGO, M.; PEZESHK, S.; GARZON, M.; BURKE, E., eds. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufmann Publishers, 2001, p. 283–290.

CORNE, D.; KNOWLES, J.; OATES, M. The pareto envelope-based selection algorithm for multiobjective optimization. In: DEB, K.; G. RUDOLPH, X. Y.; LUTTON, E.; MERELO, J. J.; SCHWEFEL, H. P., eds. *Proceedings of the Parallel Problem Solving from Nature VI Conference,*, Springer. Lecture Notes in Computer Science No. 1917, 2000, p. 839–848.

DE JONG, K. A. *Evolutionary computation: A unified approach*. Cambridge, MA: MIT Press, 2006.
Disponível em: `http://www.cs.gmu.edu/~eclab/projects/ec_courseware` (Acessado em 10/08/2008)

DEB, K. *Multi-objective optimization using evolutionary algorithms*. Wiley-Interscience Series in Systems and Optimization. New York, NY: John Wiley & Sons, 2001.

DEB, K.; AGRAWAL, S.; PRATAB, A.; MEYARIVAN, T. *A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II*. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

DELBEM, A. C. B.; CARVALHO, A. C. P. L. F.; BRETAS, N. G. Main chain representation for evolutionary algorithm applied to distribution system reconfiguration. *IEEE Transactions on Power Systems*, v. 20, n. 1, p. 425–436, 2005.

DORIGO, M.; CARO, G. D.; GAMBARDELLA, L. M. Ant algorithms for discrete optimization. *Artificial Life*, v. 5, n. 2, p. 137–172, 1999.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 1–20, 1997.

DUNI EKISOGLU, S.; PARDALOS, P.; RESENDE, M. Parallel metaheuristics for combinatorial optimization. In: *Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications*, Kluwer Academic Publishers, p. 179–206, 2002.

EIBEN, A. E.; SCHIPPERS, C. A. Multi-parent's niche:n-ary crossovers on nk-landscapes. In: VOIGT, H.-M.; EBELING, W.; RECHENBERG, I.; SCHWEFEL, H.-P., eds. *Proceedings of th 4th Conference on Parallel Problem from Nature*, Berlin, Heidelberg, New York: Springer, 1996, p. 319–328.

EIBEN, A. E.; SMITH, J. E. *Introduction to evolutionary computing*. Natural Computing Series. Berlin: Springer, 2003.

EIBEN, A. E.; VAN KEMENADE, C. H.; KOK, J. N. Orgy in the computer: Multi-parent reproduction in genetic algorithms. In: MORÁN, F.; MORENO, A.; MERELO, J. J.; CHACÓN, P., eds. *Advances in Artificial Life*, Berlin, Heidelberg, New York: Springer, 1995, p. 389–402 (*Lecture Notes in Artificial Intelligence*, v.929).

ESHELMAN, L. J.; SCHAFFER, J. D. Real-coded genetic algorithms and interval-schemata. In: *Foundations of Genetic Algorithms 2 (FOGA-2)*, San Mateo, CA, 1993, p. 187–202.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *J. of Global Optimization*, v. 1, p. 109–133, 1995.

FOGEL, D. B. An introduction to simulated evolution. *IEEE Transaction on Neural Networks*, v. 5, n. 1, p. 3–14, 1994.

FOGEL, L. Autonomus automata. *Industrial Research*, v. 4, n. 1, p. 14–19, 1962.

FONSECA, C.; FLEMING, P. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In: FORREST, S., ed. *Proceedings of the Fifth International Conference on Genetic Algorithms*, University of Illinois at Urbana-Champaign, San Mateo, California: Morgan Kauffman Publishers, 1993, p. 416–423.

GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. New York: Addison-Wesley, 1989.

GOLDBERG, D. E. Real-coded gentic algorithms, virtual alphabets, and blocking. *Complex Systems*, v. 5, n. 2, p. 139–168, 1991.

GOLDBERG, D. E. *The design of innovation: Lessons from and for competent genetic algorithms*. Boston: Kluwer, 2002.

HAJELA, P.; LIN, C. Y. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, v. 4, p. 99–107, 1992.

HOLLAND, J. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial inteligence*. Complex Adaptive Systems, 2 ed. Cambridge, MA: MIT Press, 1992.

HORN, J.; NAFPLIOTIS, N.; GOLDBERG, D. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Piscataway, New Jersey: IEEE Service Center, 1994, p. 82–87.

KITA, H.; YABUMOTO, Y.; MORI, N.; NISHIKAWA, Y. Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In: VOIGT, H.-M.; EBELING, W.; RECHENBERG, I.; SCHWEFEL, H.-P., eds. *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 1996, p. 504–512 (*Lecture Notes in Computer Science*,).

KNOWLES, J.; CORNE, D. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation. In: *1999 Congress on Evolutionary Computation*, Washington, D.C.: IEEE Service Center, 1999, p. 98–105.

KOZA, J. R. Hierarchical genetic algorithms operating on population of computer programs. In: *Proceedings of the 11th International Join Conference on Artificial Intelligence (IJCAI-89)*, Detroid, MI: Morgan Kaufmann, 1989, p. 768–774.

KOZA, J. R. *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press, 1992.

KOZA, J. R. *Genetic programming II: Automatic discovery of reusable programs*. Cambridge, MA: MIT Press, 1994.

KOZA, J. R.; BENNETT III, F. H.; ANDRE, D.; KEANE, M. A. *Genetic programming III: Darwinian invention and problem solving*. San Mateo, CA: Morgan Kaufmann, 1999.

KOZA, J. R.; KEANE, M. A.; STREETER, M. J.; MYDLOWEC, W.; YU, J.; LANZA, G. *Genetic programming IV: Routine human-competitive machine intelligence*. Kluwer Academic Publishers, 2003.

KRISHNAKUMAR, K. Micro-genetice algorithms for stationary and non-stationary function optimization. *SPIE: Inteligent Control and Adaptative Systems*, v. 1196, p. 289–296, 1989.

KURI-MORALES, A. F. Pattern recognition via vasconcelos' genetic algorithm. In: *Progress in Pattern Recognition, Image Analysis and Applications*, v. 3287 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 328–335, 2004.

KURI-MORALES, A. F.; GUTIÉRREZ-GARCÍA, J. Penalty functions methods for constrained optimization with genetic algorithms: A statistical analysis. In: COELLO COELLO, C. A.; ALBORNOZ, A.; SUCAR, L. E.; BATTISTUTTI, O. C., eds. *Proceedings of the 2nd Mexican International Conference on Artificial Intelligence (MICAI 2002)*, Heidelberg, Germany: Springer-Verlag, 2001, p. 108–117 (*Lecture Notes in Artificial Intelligence*, v.2313).

LARRAEÑAGA, P.; LOZANO, J. *Estimation of distribuition algorithms: A new tool for evolutionary computation*. Kluwer Academic Publishers, 2001.

LAUMANNS, M.; RUDOLPH, G.; SCHWEFEL, H.-P. A Spatial Predator-Prey Approach to Multi-Objective Optimization: A Preliminary Study. In: EIBEN, A. E.; SCHOENAUER, M.; SCHWEFEL, H.-P., eds. *Parallel Problem Solving From Nature — PPSN V*, Amsterdam, Holland: Springer-Verlag, 1998, p. 241–249.

MANGE, D., ed. *Bio-inspired computing machine*. Informatique. EPFL Press, 1998.

MELO, V. V.; DELBEM, A. C. B.; PINTO JUNIOR, D. L.; FEDERSON, F. M. Improving global numerical optimization using a search-space reduction algorithm. In: *Genetic and Evolutionary Computation Conference*, ACM, 2007, p. 1195–1202.

MHLENBEIN, H. The equation for response to selection and its use for prediction. *Evolutionary Computation*, v. 5, n. 3, p. 303–346, 1998.

MHLENBEIN, H.; PAAβ, G. From recombination of genes to the estimation of distribuitions: binary parameters. In: *Parallel Problem Solving from Nature – PPSN IV*, 1996, p. 178–187 (*Lecture Notes in Computer Science*, v.1411).

MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs*. 3 ed. Berlin: Springer, 1996.

MICHALEWICZ, Z.; FOGEL, D. B. *How to solve it: Modern heuristics*. 2 ed. Berlin: Springer, 2004.

MICHALEWICZ, Z.; JANIKOW, C. Z. Handling constraints in genetic algorithms. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, p. 151–157.

PAUL, T. K.; IBA, H. Linear and combinatorial optimization by estimation of distribuition algorithms. In: *Proceedings of the 9th MPS Symposium on Evolutionary Computation*, Japão, 2003.

PELIKAN, M.; GOLDBERG, D. E.; LOBO, F. *A survey of optimization by building and using probabilistic models*. Relatório Técnico 99018, Illinois Genetic Algorithms Laboratory, University fo Illinois at Urbana-Champaign, Urbana, IL, 15 p., 1999.

PILAT, M. L.; WHITE, T. Using genetic algorithms to optimize ACSTSP. In: DORIGO, M.; CARO, G. D.; SAMPELS, M., eds. *Ant Algorithms : Third International Workshop*, Springer, 2002, p. 282–287.

RECHENBERG, I. *Cybernetc solution path of an experimental problem*. Relatório Técnico 1122, Royal Aircraft Establishment, Franborough, UK, 1965.

RESENDE, M. G. C.; RIBEIRO, C. C. Parallel greedy randomized adaptive search procedures. In: *Parallel Metaheuristics: A new class of algorithms*, John Wiley and Sons, 2005.

REZENDE, S. O. *Sistemas inteligentes: fundamentos e aplicações*. Barueri, SP: Malone, 2003.

RIBEIRO, C. C.; HANSEN, P. *Essays and surveys on metaheuristics*. Kluwer Academic Publishers, 2002.

RIDLEY, M. *Evolution*. 2 ed. Cambridge, MA: Blackwell Science, Inc., 1996.

RUDOLPH, G. Evolutionary Search under Partially Ordered Fitness Sets. In: *Proceedings of the International NAISO Congress on Information Science Innovations (ISI 2001)*, ICSC Academic Press: Millet/Sliedrecht, 2001, p. 818–822.

SAIT, S. M.; YOUSSEF, H. *Interative computer algorithms with applications in engineering: Solving combinatorial optimization problems*. Los Alamitos, CA: IEEE Computer Society, 1999.

SCHAFFER, J. Multiple objective optimization with vector evaluated genetic algorithms. In: *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum, 1985, p. 93–100.

SCHWEFEL, H.-P. *Evolutionsstrategie und numerishe optimierung*. Tese de Doutoramento, Technical University of Berlin, Berlin, Alemanha, 1975.

SRINIVAS, N.; DEB, K. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, v. 2, n. 3, p. 221–248, 1994.

VELDHUIZEN, D. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. Tese de Doutoramento, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1999.

VON ZUBEN, F. J. Computação evolutiva: Uma abordagem pragmática. In: *Anais da I Jornada de Estudos em Computação de Piracicaba e Região (1a. JECOMP)*, Piracicaba, SP, 2000, p. 25–45.

ZITZLER, E.; DEB, K.; THIELE, L. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, v. 8, n. 2, p. 173–195, 2000.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Relatório Técnico 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

ZITZLER, E.; THIELE, L. *An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach*. Relatório Técnico 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1998.