

CAPÍTULO 3

INTRODUÇÃO AOS ALGORITMOS

GENÉTICOS

Estéfane G. M. de Lacerda

André Carlos P. L. F. de Carvalho

3.1 UM ALGORITMO GENÉTICO SIMPLES

Algoritmos Genéticos, AGs, são métodos de otimização e busca inspirados nos mecanismos de evolução de populações de seres vivos. Foram introduzidos por John Holland (Holland, 1975) e popularizados por um dos seus alunos, David Goldberg (Goldberg, 1989). Estes algoritmos seguem o princípio da seleção natural e sobrevivência do mais apto, declarado em 1859 pelo naturalista e fisiologista inglês Charles Darwin em seu livro **A Origem das Espécies**. De acordo com Charles Darwin, “Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes”.

Otimização é a busca da melhor solução para um dado problema. Consiste em tentar várias soluções e utilizar a informação obtida neste processo de forma a encontrar soluções cada vez melhores. Um exemplo simples de otimização é a melhoria da imagem das televisões com antena acoplada no próprio aparelho. Através do ajuste manual da antena, várias soluções são testadas, guiadas pela qualidade de imagem obtida na TV, até a obtenção de uma resposta ótima, ou seja, uma boa imagem.

As técnicas de busca e otimização, geralmente, apresentam:

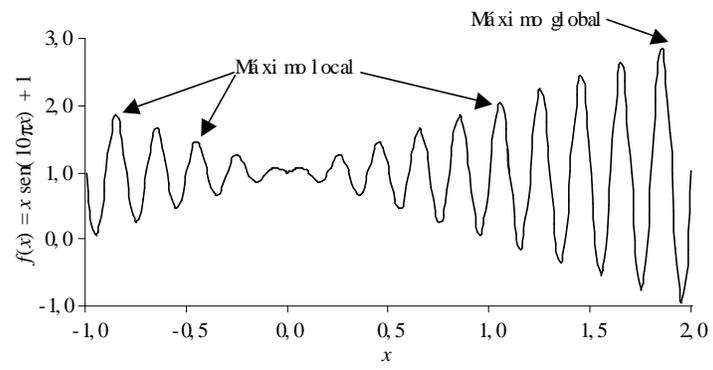
- Um espaço de busca, onde estão todas as possíveis soluções do problema;
- Uma função objetivo (algumas vezes chamada de função de aptidão na literatura de AGs), que é utilizada para avaliar as soluções produzidas, associando a cada uma delas uma nota.

Em termos matemáticos, a otimização consiste em achar a solução que corresponda ao ponto de máximo ou mínimo da função objetivo. Como exemplo, considere o seguinte problema de otimização que será utilizado no decorrer deste capítulo:

Problema 3.1 – Achar o ponto máximo da função $f(x) = x \operatorname{sen}(10\pi x) + 1$ dentro do intervalo $-1 \leq x \leq 2$.

Embora aparentemente simples, o Problema 3.1 não é de fácil solução. Existem vários pontos de máximos nesta função (pontos que maximizam o valor da função), mas muitos não representam o maior valor que a função pode atingir, conforme ilustrado na Figura 3.1. Tais pontos são denominados **máximos locais**, uma vez que a função nestes pontos atinge valores maiores do que na vizinhança destes pontos. A melhor solução para este problema está no ponto em que a função possui valor máximo, o **máximo global**. Neste problema, o máximo global encontra-se no ponto cujo valor de x é igual a 1,85055. Neste ponto, a função assume o valor 2,85027.

Conforme será mostrado na Seção 3.2, uma grande quantidade de técnicas de otimização (por exemplo, os métodos do gradiente) não é capaz de localizar o ponto de máximo global de uma função com múltiplos pontos de máximo. Esta seção mostrará como utilizar AGs para encontrar o máximo global do Problema 3.1.

Figura 3.1 - Gráfico da função da $f(x) = x \sin(10\pi x) + 1$

Algoritmo 3.1: Algoritmo Genético típico

Seja $S(t)$ a população de cromossomos na geração t .

$t \leftarrow 0$

inicializar $S(t)$

avaliar $S(t)$

enquanto o critério de parada não for satisfeito **faça**

$t \leftarrow t+1$

 selecionar $S(t)$ a partir de $S(t-1)$

 aplicar *crossover* sobre $S(t)$

 aplicar mutação sobre $S(t)$

 avaliar $S(t)$

fim enquanto

O primeiro passo de um Algoritmo Genético típico é a geração de uma **população inicial de cromossomos**, que é formada por um conjunto aleatório de cromossomos que representam possíveis soluções do problema a ser resolvido. Durante o **processo evolutivo**, esta população é avaliada e cada cromossomo recebe uma nota (denominada de **aptidão** no jargão da literatura de AGs), refletindo a qualidade da solução que ele representa. Em geral, os cromossomos mais aptos são selecionados e os menos aptos são descartados (Darwinismo). Os **membros** selecionados podem sofrer modificações em suas características fundamentais através dos **operadores** de **crossover** e **mutação**, gerando descendentes para a próxima geração. Este processo é repetido até que uma solução satisfatória seja encontrada. O Algoritmo 3.1 ilustra este procedimento. As seções seguintes descrevem com mais detalhes cada etapa deste algoritmo.

3.1.1 REPRESENTAÇÃO DOS PARÂMETROS

Um AG processa populações de **cromossomos**. Um cromossomo é uma estrutura de dados, geralmente vetor ou

cadeia de bits (cadeia de bits é a estrutura mais tradicional, porém nem sempre é a melhor), que representa uma possível solução do problema a ser otimizado. Em geral, um cromossomo representa um conjunto de parâmetros da função objetivo cuja resposta será maximizada ou minimizada. O conjunto de todas as configurações que o cromossomo pode assumir forma o seu **espaço de busca**. Se o cromossomo representa n parâmetros de uma função, então o espaço de busca é um espaço com n dimensões.

O primeiro passo para resolver o Problema 3.1 utilizando AGs é representar o único parâmetro deste problema (a variável x) na forma de um cromossomo. Será adotada uma cadeia de 22 bits para o cromossomo (maiores cadeias aumentam a precisão numérica da solução). Assim, um exemplo de cromossomo poderia ser:

$$s_1 = 1000101110110101000111$$

Para decodificar o cromossomo s_1 , converte-se s_1 da base 2 para a base 10:

$$b_{10} = (1000101110110101000111)_2 = 2288967$$

Como b_{10} é um número no intervalo $[0, 2^l - 1]$ (sendo l o tamanho da cadeia de bits), é necessário mapeá-lo para o intervalo do problema $[-1, 0; 2, 0]$. Isto pode ser feito pela fórmula:

$$x = \min + (\max - \min) \frac{b_{10}}{2^l - 1}$$

assim,

$$x_1 = -1 + (2 + 1) \frac{2.288.967}{(2^{22} - 1)} = 0,637197$$

Vale observar que funções objetivo com múltiplos parâmetros têm seus parâmetros representados na mesma cadeia de bit, com cada um ocupando uma parte da cadeia. A cada cromossomo s_i é atribuída uma aptidão f_i . Aptidão é uma nota

que mede quão boa é a solução codificada em \mathbf{s}_i . É baseada no valor da função objetivo, e será discutida na próxima seção.

3.1.2 SELEÇÃO

Um Algoritmo Genético começa com uma **população inicial** de N cromossomos. A população inicial do Problema 3.1, com $N = 30$, é mostrada em uma das colunas da Tabela 3.1 ordenados por ordem decrescente do valor da função objetivo, como também mostra-se o valor de x codificado no cromossomo, o valor da função objetivo e a aptidão para este valor. Os cromossomos foram gerados aleatoriamente, porque neste problema não existe nenhum conhecimento prévio sobre a região do espaço de busca onde se encontra a solução do problema. Os pontos da função representados pelos cromossomos da população inicial são mostrados na Figura 3.2.

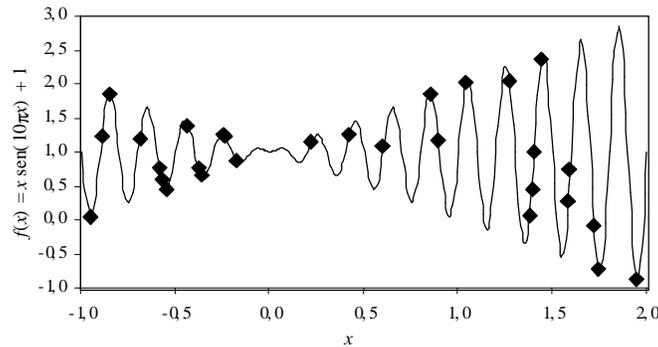


Figura 3.2 - Cromossomos da População Inicial

Inspirado no processo de seleção natural de seres vivos, o Algoritmo Genético seleciona os melhores cromossomos da população inicial (aqueles de alta aptidão) para gerar **cromossomos filhos** (que são variantes dos pais) através dos operadores de *crossover* e *mutação*. Uma **população intermediária** (também chamada de *mating pool*) é utilizada para alocar os cromossomos pais selecionados. Geralmente, os pais são selecionados com probabilidade **proporcional à sua**

aptidão. Portanto, a probabilidade de seleção p_i , de um cromossomo \mathbf{s}_i com aptidão f_i , é dada por:

$$p_i = \frac{f_i}{\sum_{i=1}^N f_i}$$

A seleção pode ser feita pelo seguinte procedimento prático: calcula-se uma coluna de aptidões acumuladas. Em seguida, gera-se um número aleatório r (tirado de uma distribuição uniforme) no intervalo $[0, \text{SOMATOTAL}]$, onde SOMATOTAL é a soma de todas as aptidões. Por fim, o cromossomo selecionado é o primeiro (segundo a tabela de cima para baixo) que possui aptidão acumulada maior que r . Por exemplo, se $r = 28,131$, então o cromossomo da linha 23 da Tabela 3.1 é selecionado, e sua cópia é alocada na população intermediária. Os mesmos passos são repetidos até preencher a população intermediária com N cromossomos. Este procedimento, conhecido como **Roda da Roleta**, é apresentado pelo Algoritmo 3.2.

Tabela 3.1 - População Inicial

Posição i	Cromossomo \mathbf{s}_i	x_i	Função objetivo $f(x_i)$	Aptidão f_i	Aptidão acumulada $\sum_{k=1}^i f_k$
1	110100000011110110 111	1,43891	2,35251	2,00000	2,00000
2	1100000110100100011 111	1,26925	2,04416	1,93103	3,93103
3	1010111001010110010 000	1,04301	2,01797	1,86207	5,79310
4	1001111000011001000 101	0,85271	1,84962	1,79310	7,58621
5	1001110110111000011 100	0,84829	1,84706	1,72414	9,31035
6	0000110011111010010 110	- 0,84792	1,84610	1,65517	10,96552

7	0011000000100111010 010	- 0,43570	1,39248	1,58621	12,55172
8	0111100101000001101 100	0,42098	1,25777	1,51724	14,06897
9	010000000110011101 000	- 0,24764	1,24695	1,44828	15,51724
10	0100000010001111011 110	- 0,24343	1,23827	1,37931	16,89655
11	0000100101000000111 010	- 0,89156	1,23364	1,31035	18,20690
12	0001101001100010101 111	- 0,69079	1,19704	1,24138	19,44828
13	1010000110011000011 011	0,89370	1,17582	1,17241	20,62069
14	0110100001011011000 100	0,22292	1,14699	1,10345	21,72414
15	1000100011110001000 011	0,60479	1,09057	1,03448	22,75862
16	1100110011001010001 110	1,39988	0,99483	0,96552	23,72414
17	0100011001000100011 101	- 0,17655	0,88140	0,89655	24,62069
18	0011010011110100101 000	- 0,37943	0,77149	0,82759	25,44828
19	0010001101001100101 100	- 0,58633	0,75592	0,75862	26,20690
20	1101110101101111111 111	1,59497	0,74904	0,68966	26,89655
21	0011011011001101110 110	- 0,35777	0,65283	0,62069	27,51724
22	0010010001001111100 111	- 0,57448	0,58721	0,55172	28,06897
23	1100101110110011111 000	1,38714	0,45474	0,48276	28,55172
24	0010011001100110100 111	- 0,54999	0,45001	0,41379	28,96552
25	1101110010010100100 001	1,58492	0,27710	0,34483	29,31035
26	1100101011000111010 011	1,37631	0,06770	0,27586	29,58621
27	0000010000100100110	-	0,04953	0,20690	29,79310

	001	0,95144			
28	1110100001000000010 001	1,72169	- 0,08458	0,13793	29,93103
29	1110101000111100000 000	1,74494	- 0,72289	0,06897	30,00000
30	1111101100000001010 111	1,94147	- 0,87216	0,00000	30,00000

Algoritmo 3.2: Roda da Roleta

```

total ←  $\sum_{i=1}^N f_i$  /* a soma das aptidões de todos os cromossomos da
população */
rand ← randômico(0, total)
total parcial ← 0
i ← 0
repetir
    i ← i + 1
    total parcial ← total parcial +  $f_i$ 
até total parcial ≥ rand
retornar o cromossomo  $s_i$ 

```

Várias alternativas têm sido propostas para definir a aptidão. A mais simples iguala a aptidão ao valor da função objetivo. Assim, a aptidão do cromossomo s_1 seria:

$$f_1 = 0,637197 \sin(10\pi \cdot 0,637197) + 1 = 1,586345$$

Vale observar que a aptidão definida desta forma pode assumir valores negativos e o algoritmo Roda da Roleta não funciona com aptidões negativas. Além disso, pode gerar também problemas como convergência prematura (conforme será mostrado na Seção 3.3). É possível, no entanto, dispensar o Algoritmo Roda da Roleta e utilizar **Seleção por Torneio**. Neste caso, são escolhidos, aleatoriamente, (com probabilidades iguais) n cromossomos da população, e o cromossomo com maior aptidão é selecionado para a

população intermediária. O processo repete-se até preencher a população intermediária. Utiliza-se, geralmente, o valor $n = 3$.

Outra forma de definir aptidão é pelo **ordenamento** do cromossomo na população, conforme se obteve na Tabela 3.1. O primeiro cromossomo do ordenamento recebeu uma aptidão arbitrária igual a 2,0, e ao último cromossomo do ordenamento foi atribuído o valor 0,0. As demais foram obtidas interpolando estes dois extremos por uma reta, ou seja, $f_i = 2(N-i)/(N-1)$, sendo N o tamanho da população.

3.1.3 CROSSOVER E MUTAÇÃO

Os operadores de *crossover* e a mutação são os principais mecanismos de busca dos AGs para explorar regiões desconhecidas do espaço de busca. O operador *crossover* é aplicado a um par de cromossomos retirados da população intermediária, gerando dois cromossomos filhos. Cada um dos cromossomos pais tem sua cadeia de bits cortada em uma posição aleatória, produzindo duas cabeças e duas caudas. As caudas são trocadas, gerando dois novos cromossomos. A Figura 3.3 ilustra o comportamento deste operador.

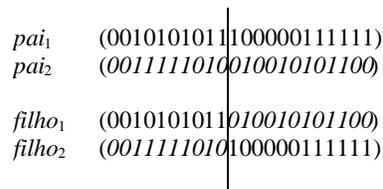


Figura 3.3 - *Crossover*

O *crossover* é aplicado com uma dada probabilidade a cada par de cromossomos selecionados. Na prática, esta probabilidade, denominada de **taxa de crossover**, varia entre 60% e 90%. Não ocorrendo o *crossover*, os filhos serão iguais aos pais (isto permite que algumas soluções sejam preservadas). Isto pode ser implementado, gerando números pseudo-aleatórios no intervalo $[0,1]$. Assim, o *crossover* só é aplicado se o número gerado for menor que a taxa de *crossover*.

Após a operação de *crossover*, o operador de mutação é aplicado, com dada probabilidade, em cada bit dos dois filhos. O operador de mutação inverte os valores de bits, ou seja, muda o valor de um dado bit de 1 para 0 ou de 0 para 1. A Figura 3.4 apresenta um exemplo em que dois bits do primeiro filho e um bit do segundo sofrem mutação (bits estes que passaram no teste de probabilidade). A mutação melhora a diversidade dos cromossomos na população, no entanto por outro lado, destrói informação contida no cromossomo, logo, deve ser utilizada uma **taxa de mutação** pequena (normalmente entre 0,1% a 5%), mas suficiente para assegurar a diversidade.

```

Antes  filho1 (0010101010010010101100)
        filho2 (0011111011100000111111)

Depois filho1 (0010001010010010111100)
        filho2 (0011111011000000111111)
    
```

Figura 3.4 - Mutação

A Figura 3.5 ilustra a aplicação do operador de *crossover* a cada par de cromossomos da população intermediária e os bits que sofreram mutação na nova população. Os pontos representados pela primeira geração de cromossomos são mostrados na Figura 3.6. Ainda não houve melhora significativa com relação à população inicial, ou seja, os cromossomos ainda estão distantes do máximo global da função.

Índice Original	População Intermediária (<i>Mating pool</i>)	<i>Crossover</i> e Mutação	Primeira Geração (o sublinhado indica mutação)	Ponto de Corte
1	1101000000011110110111		1101000000011000011100	12
5	1001110110111000011100		1001110110111110110111	12
12	0001101001100010101111		00011010011 <u>0</u> 010010110	12
6	0000110011111010010110		00 <u>1</u> 01100111100 <u>0</u> 101111	12
19	0010001101001100101100	•	<u>0</u> 110001101000100011101	9
17	0100011001000100011101		0100011001001100101100	9
15	1000100011110001000011		1000100011110001010010	17
7	0011000000100111010010	•	<u>0</u> 01000000100111000011	17

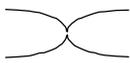
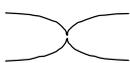
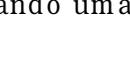
10	0100000010001111011110		0100000010001111011100	16
8	0111100101000001101100		0111100101000001011110	16
6	0000110011111010010110		0000110011111010010110	21
18	0011010011110100101000		001101001111010010 <u>0000</u>	21
9	0100000001100111010100		0100 <u>100000</u> 010010101111	13
12	0001101001100010101111		0001101001100011101000	13
17	0100011001000100011101		0100011001000100010010	17
7	0011000000100111010010		001100 <u>1000</u> 100111011101	17
11	0000100101000000111010	•	000011100101011001000 <u>1</u>	3
3	1010111001010110010000	•	10101001010 <u>100000</u> 1101 <u>1</u>	3
3	1010111001010110010000	•	1010111001010110010111	19
1	1101000000011110110111		1101000 <u>1000</u> 11110110000	19
15	1000100011110001000011		10001000100110010000101	9
4	1001111000011001000101		1001111001110001000011	9
13	1010000110011000011011		1010000110011000011010	20
6	0000110011111010010110		0 <u>100</u> 11 <u>10</u> 111110100101 <u>10</u>	20
15	1000100011110001000011		1000010000100100110001	3
27	0000010000100100110001		0000100011110001000011	3
8	0111100101000001101100		011 <u>000</u> 01010000 <u>1</u> 1101100	19
19	0010001101001100101100		<u>10</u> 10001101001100101 <u>000</u>	19
8	0111100101000001101100		0111100101000 <u>100</u> 111010	5
11	0000100101000000111010		00001001010000 <u>1</u> 1101100	5

Figura 3.5 - Gerando uma nova população

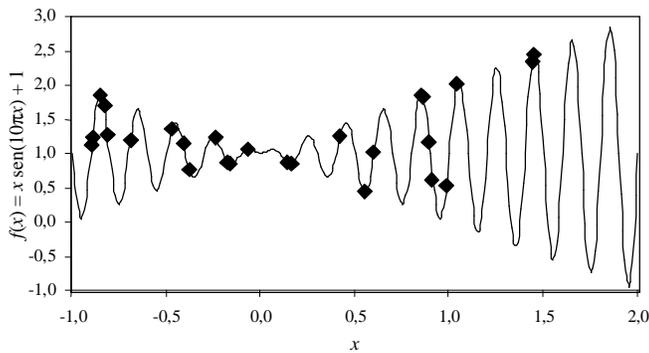


Figura 3.6 - Pontos da primeira geração de cromossomos

Após a definição da primeira população, o procedimento se repete por um dado número de gerações. Quando se conhece a resposta máxima da função objetivo, pode-se utilizar este valor como critério de parada do Algoritmo Genético. Na Figura 3.7, é ilustrada a última geração, a vigésima quinta, em

que boa parte dos cromossomos representa pontos no pico do máximo global. Não há um critério exato para terminar a execução do AG. Porém, com 95% dos cromossomos representando o mesmo valor, é possível dizer que o algoritmo convergiu.

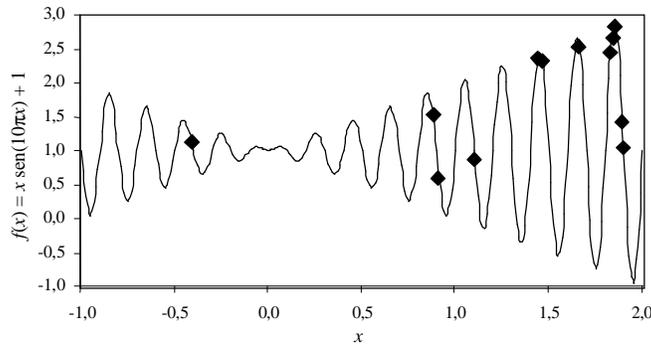


Figura 3.7 - Cromossomos da vigésima quinta geração

3.1.4 ELITISMO

A Figura 3.8 ilustra, para cada geração, o valor da função objetivo para o melhor cromossomo da população, além do valor médio da função objetivo de toda a população. Vale observar que o melhor cromossomo pode ser perdido de uma geração para outra devido ao corte do *crossover* ou à ocorrência de mutação. Portanto, é interessante transferir o melhor cromossomo de uma geração para outra sem alterações (porque perder a melhor solução encontrada até então?). Esta estratégia é denominada **Elitismo**, sendo muito comum nos AGs tradicionais. O Elitismo foi proposto por DeJong (1975), um dos trabalhos pioneiros sobre AGs.

A Figura 3.9 mostra o desempenho do melhor cromossomo em cada geração, usando o AG com e sem Elitismo, com os valores plotados no gráfico representando a média de 100 execuções do AG. Claramente é mostrado que, neste problema, o AG com elitismo encontra a solução mais rápido que o AG sem elitismo (vale ressaltar que, em algumas execuções, o AG ocasionalmente encontra máximos locais, tornando a média dos melhores cromossomos menor que o máximo da função).

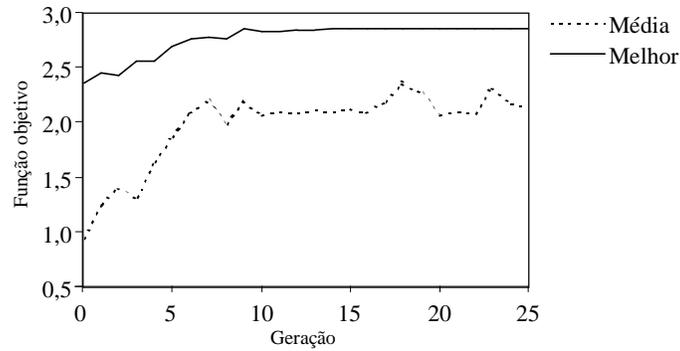


Figura 3.8 - O maior valor e o valor médio da função objetivo em cada geração

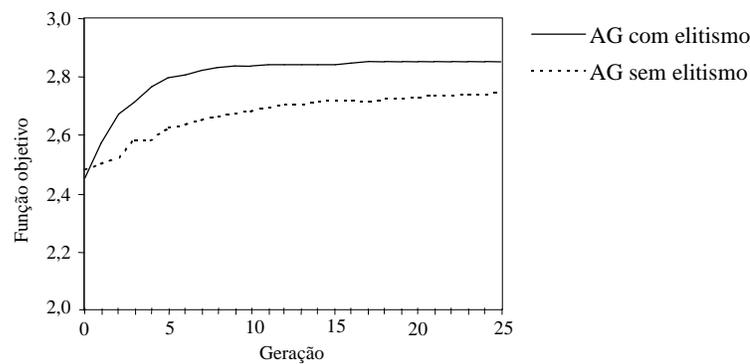


Figura 3.9 - Desempenho do AG com Elitismo

3.1.5 CROSSOVERS DE N PONTOS E UNIFORME

Os tipos de operadores *crossover* mais conhecidos para cadeias de bits são o de n pontos e o uniforme. O *crossover* de 1 ponto é o mesmo apresentado na Seção 3.3. O de 2 pontos é apresentado na Figura 3.10. Os dois pontos de corte são escolhidos aleatoriamente, e as seções entre os dois pontos são trocadas entre os pais. O *crossover* de 4 pontos é mostrado na Figura 3.11. O *crossover* de n pontos mais usado tem sido o de 2 pontos por algumas razões mostradas na Seção 3.7.

```

pai1 010011000101011
pai2 001001110001101
filho1 010001110101011
filho2 001011000001101

```

Figura 3.10 - *Crossover* de 2 pontos

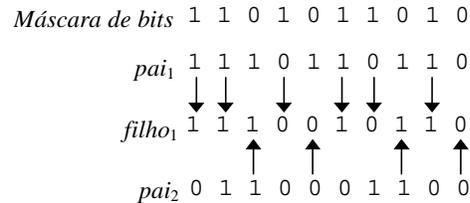
```

pai1 1010100100101001
pai2 00100111000110100
filho1 101001110010101001
filho2 0010100100011001100

```

Figura 3.11 - *Crossover* de 4 pontos

O *crossover* uniforme é apresentado na Figura 3.12. Para cada par de pais é gerada uma máscara de bits aleatórios. Se o primeiro bit da máscara possui o valor 1, então o primeiro bit do pai_1 é copiado para o primeiro bit do $filho_1$. Caso contrário, o primeiro bit do pai_2 é copiado para o primeiro bit do $filho_1$. O processo se repete para os bits restantes do $filho_1$. Na geração do $filho_2$ o procedimento é invertido, ou seja, se o bit da máscara é 1, então será copiado o bit do pai_2 . Se o bit for igual a 0, então será copiado o bit do pai_1 . Vale notar que o *crossover* uniforme não é a mesma coisa que o *crossover* de $(l-1)$ pontos (l é o número de bits do cromossomo), uma vez que este sempre leva a metade dos bits de cada pai.

Figura 3.12 - *Crossover* Uniforme

Em Eshelman et al. (1989), é investigada a diferença de desempenho entre vários *crossovers* de n pontos e uniforme. A conclusão, conforme relatado em Beasley (1993), é que não há grandes diferenças de desempenho entre eles. Aliás, segundo Grefenstette (1986), o AG é robusto de tal modo que, dentro de uma faixa relativamente larga de variação de parâmetros (e.g. taxas de *crossover* e mutação, tamanho da população, etc.), não ocorre alteração significativa no desempenho.

No item a seguir, será apresentada a relação entre AGs e a biologia, como também a explicação dos principais termos utilizados na literatura de AGs.

3.1.6 TERMINOLOGIA

Na biologia, a teoria da evolução diz que o meio ambiente seleciona, em cada geração, os seres vivos mais aptos de uma população para sobrevivência. Como resultado, somente os mais aptos conseguem se reproduzir, uma vez que os menos adaptados geralmente são eliminados antes de gerarem descendentes. Durante a reprodução, ocorrem fenômenos como mutação e *crossover* (recombinação), entre outros, que atuam sobre o material genético armazenado nos cromossomos. Estes fenômenos levam à variabilidade dos seres vivos na população. Sobre esta população diversificada age a seleção natural, permitindo a sobrevivência apenas dos seres mais adaptados.

Um Algoritmo Genético é a metáfora desses fenômenos, o que explica porque AGs possuem muitos termos originados da

biologia. A lista apresentada a seguir descreve os principais termos encontrados na literatura.

- **Cromossomo e Genoma:** Na biologia, genoma é o conjunto completo de genes de um organismo. Um genoma pode ter vários cromossomos. Nos AGs, os dois representam a estrutura de dados que codifica uma solução para um problema, ou seja, um cromossomo ou genoma representa um simples ponto no espaço de busca.
 - **Gene:** Na biologia, é a unidade de hereditariedade que é transmitida pelo cromossomo e que controla as características do organismo. Nos AGs, é um parâmetro codificado no cromossomo, ou seja, um elemento do vetor que representa o cromossomo.
 - **Indivíduo:** Um simples membro da população. Nos AGs, um indivíduo é formado pelo cromossomo e sua aptidão.
 - **Genótipo:** Na biologia, representa a composição genética contida no Genoma. Nos AGs, representa a informação contida no cromossomo ou genoma.
 - **Fenótipo:** Nos Algoritmos Genéticos, representa o objeto, estrutura ou organismo construído a partir das informações do genótipo. É o cromossomo decodificado. Por exemplo, considere que o cromossomo codifica parâmetros como as dimensões das vigas em um projeto de construção de um edifício, ou as conexões e pesos de uma Rede Neural. O fenótipo seria o edifício construído ou a Rede Neural.
 - **Alelo:** Na biologia, representa uma das formas alternativas de um gene. Nos AGs, representa os valores que o gene pode assumir. Por exemplo, um gene que representa o parâmetro cor de um objeto poderia ter o alelo azul, preto, verde, etc.
 - **Epistasia:** Interação entre genes do cromossomo, isto é, quando um valor de gene influencia o valor de
-

outro gene. Problemas com alta Epistasia são de difíceis solução por AGs.

A seção seguinte aborda alguns aspectos teóricos de AGs.

3.1.7 ESQUEMAS

O Teorema dos Esquemas de Holland (1975) procura fundamentar, teoricamente, o comportamento dos AGs. Sua compreensão pode auxiliar na construção de aplicações eficientes de AGs. Holland constatou que os AGs manipulam determinados segmentos da cadeia de bits. Tais segmentos foram por ele denominados de esquemas. Um é formado pelos símbolos 0, 1 e *. A ocorrência do símbolo * em uma posição no esquema significa que esta posição pode ser ocupada pelo símbolo 1 ou 0. A Figura 3.13 mostra alguns exemplos.

	H_1	H_2	H_3
	1****	**10*	*0*01
11001	X		
11011	X		
10101	X	X	X

Comprimento $\delta(H_1) = 0$ Ordem $O(H_1) = 1$
 Comprimento $\delta(H_2) = 1$ Ordem $O(H_2) = 2$
 Comprimento $\delta(H_3) = 3$ Ordem $O(H_3) = 3$

Figura 3.13 – Esquemas

Conforme mostrado na Figura 3.13, os esquemas $H_1 = 1****$, $H_2 = **10*$ e $H_3 = *0*01$ estão contidos no mesmo cromossomo 10101, que ao todo pode ter $2^5 = 32$ esquemas. Os cromossomos 11001, 11011 e 10101 possuem o esquema 1****. O comprimento de um esquema $\delta(H)$ é a diferença entre a última posição que é ocupada por 1 ou 0 e a primeira posição que é ocupada por 1 e 0. A ordem $O(H)$ de um esquema é o número de símbolos 1 e 0 que o esquema contém.

Para prever a variação do número de esquemas H entre duas gerações consecutivas, considere m o número de

cromossomos da população atual que contém o esquema H . Considere b a média das aptidões de toda população e a a média das aptidões dos cromossomos que contém o esquema H . Assim, o número esperado de cópias m' do esquema H na população intermediária (utilizando seleção proporcional à aptidão) é dado por:

$$m' = \frac{a}{b}m$$

Pela equação, conclui-se que o número de esquemas H aumentará na população intermediária se $a > b$, ou seja, se o esquema H estiver contido em cromossomos de aptidão acima da aptidão média da população (bons cromossomos). No entanto, ao passar para a próxima população, o esquema H pode ser destruído pelos operadores de *crossover* e de mutação. Por exemplo:

Esquema contido em pai_1 (01*|**10)

Esquema contido em pai_2 (***|*101)

Esquema contido em $filho_1$ (01*|*101)

O esquema do pai_2 (que tem pequeno comprimento) foi transmitido ao $filho_1$, mas o esquema do pai_1 (que tem maior comprimento) foi destruído pelo *crossover*. Porém, mesmo considerando estes fatores, o Teorema dos Esquemas afirma que:

Pequenos esquemas contidos em bons cromossomos (i.e. aqueles com aptidão acima da média) aumentam exponencialmente nas gerações seguinte, ao passo que esquemas contidos em cromossomos ruins (i.e. aqueles com a aptidão abaixo da média) tendem a desaparecer nas gerações seguintes (Goldberg, 1989).

Os bons esquemas de pequeno tamanho recebem o nome especial de **blocos de construção**. A informação contida em um bloco de construção é combinada com as informações de outros blocos de construção. No decorrer das gerações, esta combinação produz cromossomos de alta aptidão. Esta

afirmação é conhecida como a **Hipótese dos Blocos de Construção**.

Os problemas que não obedecem à Hipótese dos Blocos de Construção são conhecidos como **AG-Deceptivos**. Por exemplo, pode ocorrer que, combinando dois ótimos blocos de construção, resulte em um cromossomo ruim, ocorrendo isto com genes que têm alta Epistasia (ver Seção 3.1.6). Funções com esta propriedade tendem a ter um ponto ótimo isolado cercado por pontos extremamente ruins. Felizmente, tais funções são raras na prática. E, além disso, são funções difíceis para qualquer técnica de otimização.

Holland também notou que apesar do AG manipular N cromossomos, a quantidade de esquemas manipulados é muito maior (na ordem de $O(N^3)$ esquemas). Tal propriedade foi denominada de **Paralelismo Implícito**. Ou seja, o AG manipula uma grande quantidade de informações em paralelo com apenas N cromossomos.

Os bons blocos de construção, quando incorporados em um cromossomo, melhoram sua aptidão. Portanto, é importante estruturar a codificação dos cromossomos de modo a estimular a formação de blocos de construção.

Sob a luz dos esquemas, é possível analisar os diversos tipos de crossovers. Por exemplo, o esquema 1*****0 será fatalmente destruído pelo *crossover* de 1 ponto, seja onde for o ponto de corte. O mesmo problema não ocorre no *crossover* de 2 pontos, porém, o aumento excessivo de pontos de corte normalmente não leva a bons resultados, uma vez que destrói com facilidade os blocos de construção.

É interessante notar que o cromossomo pode ser interpretado como um anel formado pela união de suas extremidades, como ilustra a Figura 3.14. Observando o anel, nota-se que o *crossover* de 1 ponto é um caso particular do de 2 pontos quando um dos pontos de corte é fixo sobre a junção do anel. Portanto, o *crossover* de 2 pontos desempenha também a função do de 1 ponto. Este é um dos motivos pelo qual o *crossover* de 2 pontos é considerado melhor do que o *crossover* de 1 ponto (Beasley, 1993b; Davis, 1991).

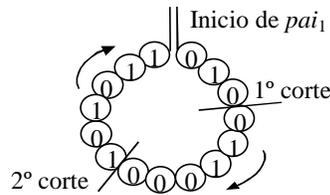


Figura 3.14 - Crossover de 2 pontos visto como um anel

O *crossover* de n pontos tende a manter juntos os genes que são codificados próximos um do outro. Existe um operador denominado **Inversão** (Holland, 1975) que busca o ordenamento ideal dos genes no cromossomo. Dois pontos aleatórios são escolhidos no cromossomo e os genes entre eles são invertidos. Vale frisar que a inversão não é um tipo de mutação brutal. Na verdade não ocorre mutação. A cada gene é associado um número para identificar quem é quem depois da troca de posição. Este operador, no entanto, tem sido raramente utilizado na prática.

Considerando agora o efeito destrutivo dos blocos de construção causada pelo *crossover* uniforme, Syswerda (1989) argumenta que tal destruição é compensada pelo fato de ele poder combinar qualquer material dos cromossomos pais, independentemente da ordem dos genes.

3.2 REPRESENTAÇÃO REAL

3.2.1 REPRESENTAÇÃO BINÁRIA \times REAL

Para ilustrar esta seção, considere o Problema 3.2 que é maximizar uma função (conhecida na literatura de AG como função F6 (Davis, 1991)) com dois parâmetros x e y . Na Figura 3.15 é mostrado a representação binária do cromossomo e sua decomposição nos dois parâmetros x e y .

**Problema
3.2**

$$\begin{aligned} &\text{Maximiz} \\ &\text{ar} \quad f(x, y) = 0,5 - \frac{\left(\sin \sqrt{x^2 + y^2}\right)^2 - 0,5}{\left(1,0 + 0,001(x^2 + y^2)\right)^2} \\ &\text{Sujeito a} \quad -100 \leq x \leq 100 \\ &\quad \quad \quad -100 \leq y \leq 100 \end{aligned}$$

Decodificação de Cromossomo em dois parâmetros

$s_1 = 01101001001001101000001000111000100001110010$

Passo 1: Divisão da cadeia de bits s_1 em duas cadeias de 22 bits:

0110100100100110100000 e 1000111000100001110010

Passo 2: Conversão da base 2 para a base 10:

1722784 e 2328690.

Passo 3: Mapear estes resultados para o intervalo $[-100, 100]$:

$$x_1 = \frac{1722784}{2^{22} - 1} [100 - (-100)] + 100 = -17,851$$

$$y_1 = \frac{2328690}{2^{22} - 1} [100 - (-100)] + 100 = 11,041$$

Figura 3.15 - Decodificação do cromossomo em dois parâmetros (x,y)

A representação binária é historicamente importante, uma vez que foi utilizado nos trabalhos pioneiros de Holland (1975). É a representação tradicional, sendo fácil de utilizar e manipular, como também é simples de analisar teoricamente. Contudo, se um problema tem parâmetros contínuos e o usuário quer trabalhar com boa precisão numérica, ele precisará armazenar cromossomos longos na memória.

Pois, para cada ponto decimal acrescentado na precisão, é necessário adicionar 3,3 bits na cadeia. Caso 8 casas decimais

sejam necessárias, $8 \times 3,3 \approx 27$ bits serão utilizados para cada parâmetro. Quando há muito parâmetros, obtém-se longas cadeias de bits que podem fazer o algoritmo convergir vagarosamente. Além disso, não existe uniformidade nos operadores. Por exemplo, mutação nos primeiros bits do gene afeta mais a aptidão que mutação nos últimos bits do gene.

A representação real (i.e. com ponto flutuante) gera cromossomos menores e é compreendida mais naturalmente pelo ser humano do que a cadeia de bits. No Problema 3.2, o cromossomo seria representado por um vetor de dois números com ponto flutuante. Por exemplo $(-17,851; 11,041)$. Outra vantagem da representação real é a facilidade de criar novos operadores. Existe grande número deles conforme mostrado na próxima seção.

Vários pesquisadores têm discutido qual a melhor representação, a binária ou a real, e muitos deles têm mostrado experimentos favoráveis à representação real (Michalewicz, 1994; Haupt e Haupt, 1998).

3.2.2 CROSSOVERS PARA REPRESENTAÇÃO REAL

A seguir será apresentada uma lista de operadores para representação real. Considere a seguinte notação utilizada nesta seção. Os cromossomos pais serão representados por:

$$\mathbf{p}_1 = (p_{11}, p_{12}, \dots, p_{1l})$$

$$\mathbf{p}_2 = (p_{21}, p_{22}, \dots, p_{2l})$$

e o cromossomo filho por

$$\mathbf{c} = (c_1, c_2, \dots, c_l).$$

onde $p_{ij} \in \mathcal{R}$ e $c_i \in \mathcal{R}$. Quando há mais de um filho, cada filho i será representado por $\mathbf{c}_i = (c_{i1}, c_{i2}, \dots, c_{il})$.

Os genes podem ter variados tipos de restrições, mas por simplicidade, considere que o gene c_i do filho \mathbf{c} está restrito ao intervalo $[a_i, b_i]$. Se o gene c_i não satisfaz a esta restrição então o filho \mathbf{c} é denominado de **infectível**. Será utilizado a notação $U(x,y)$ para representar uma distribuição uniforme no intervalo

$[x,y]$ e a notação $N(\mu,\sigma)$ para representar uma distribuição normal com média μ e desvio padrão σ .

Operadores Convencionais

Os operadores convencionais são resultados das adaptações dos operadores utilizados para representação binária. Os operadores convencionais (*crossover* de n pontos e uniforme) funcionam bem na representação binária, mas na representação real eles basicamente trocam valores dos genes e, portanto, não criam informações novas (i.e., novos números reais). Melhor então é usar operadores aritméticos.

Operadores Aritméticos

Os operadores aritméticos realizam algum tipo de combinação linear entre os cromossomos pais.

Crossover média (Davis, 1991): dado dois cromossomos \mathbf{p}_1 e \mathbf{p}_2 , é produzido um cromossomo \mathbf{c} da seguinte forma:

$$\mathbf{c} = (\mathbf{p}_1 + \mathbf{p}_2) / 2$$

Um variação deste crossover é o:

Crossover média geométrica, onde cada gene c_i do filho \mathbf{c} é dado por:

$$c_i = \sqrt{p_{1i} p_{2i}} \quad \text{para } i=1\dots l$$

O crossover média tende a levar os genes para o meio do intervalo permitido $[a_i, b_i]$ causando perda de diversidade. Isto pode ser melhorado com o crossover BLX- α .

Crossover BLX- α ou **crossover mistura** (do inglês *blend crossover*) (Eshelman e Shaffer, 1993): dado dois cromossomos \mathbf{p}_1 e \mathbf{p}_2 , é produzido um cromossomo \mathbf{c} da seguinte forma:

$$\mathbf{c} = \mathbf{p}_1 + \beta(\mathbf{p}_2 - \mathbf{p}_1)$$

onde $\beta \in U(-\alpha, 1 + \alpha)$.

O BLX- α é ilustrado na Figura 3.16 na qual foi escolhido um único valor de β para todos os pares de genes. Quando $\alpha = 0$ o filho situa-se sobre o intervalo \mathbf{I} entre os dois pontos que representam os pais. O parâmetro α estende o intervalo \mathbf{I} . Por exemplo, se $\alpha = 0,5$, o intervalo \mathbf{I} é estendido $\alpha \mathbf{I}$ em ambos os lados. O BLX-0,5 balanceia a tendência de gerar filhos próximos ao centro do intervalo \mathbf{I} evitando a perda de diversidade. Esta tendência ainda pode ser reduzida com a mutação limite mostrada mais tarde. O BLX- α tem sido usado com sucesso em muitos trabalhos e é talvez o operador o mais utilizado para representação real.

Exemplo: Considere dois possíveis cromossomos para o Problema 2.1:

$$p_1 = (30,173; 85,342)$$

$$p_2 = (75,989; 10,162)$$

Aplicando o BLX-0,5 com $\beta = 1,262$ (onde $\beta \in U(-0,5; 1,5)$), tem-se:

$$c_1 = 30,173 + 1,262(75,989 - 30,173) = 87,993$$

$$c_2 = 85,342 + 1,262(10,162 - 85,342) = -9,535$$

assim o filho é dado por:

$$c = (87,993; -9,535)$$

Se o filho c for infactível, então gerar-se outro filho com novo β . O processo é repetido até obter um filho factível.

Note que neste exemplo, foi utilizado apenas um β para todos os genes. Alternativamente, pode-se usar um β diferente para cada par de gene. Neste caso, um

possível filho situa-se em algum lugar de uma área limitada por um retângulo (ver Figura 3.17). Nestes métodos, os genes podem ser combinados de várias maneiras. Por exemplo, combinar todos os genes ou escolher (como no *crossover* de 1 ponto) um gene aleatório k e combinar somente o genes $i \geq k$, ou seja:

$$p_1 = (p_{11}, p_{12}, \dots, p_{1k}, p_{1,k+1}, \dots, p_{1l})$$

$$p_2 = (p_{21}, p_{22}, \dots, p_{2k}, p_{2,k+1}, \dots, p_{2l})$$

$$c = (p_{11}, p_{12}, \dots, c_k, c_{k+1}, \dots, c_l)$$

Uma alternativa para o *crossover* BLX- α é o:

Crossover linear (Wright, 1991): dados dois pais p_1 e p_2 obtém-se três filhos c_1 , c_2 e c_3 da seguinte forma:

$$c_1 = 0,5p_1 + 0,5p_2$$

$$c_2 = 1,5p_1 - 0,5p_2$$

$$c_3 = -0,5p_1 + 1,5p_2$$

Desses três filhos, apenas o melhor é escolhido, os outros dois são descartados.

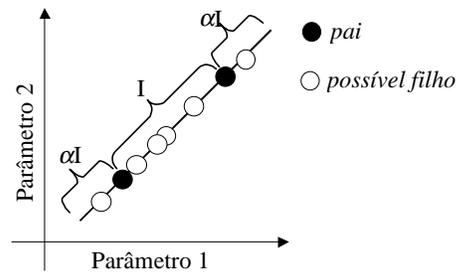


Figura 3.16 - *Crossover* BLX- α

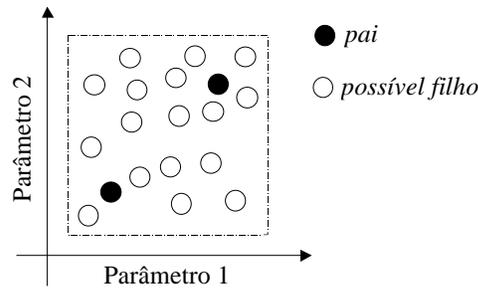


Figura 3.17 - *Crossover* BLX- α com β variável

Outro estudo que vale mencionar é o de Michalewicz (1994), que inventou vários operadores para representação real. A combinação destes operadores no mesmo AG apresentou melhor desempenho que o AG binário tradicional. Ele usou: *crossover* aritmético, *crossover* heurístico, *crossover* simples, mutação uniforme, mutação limite, mutação não uniforme e a mutação não-uniforme múltipla. A seguir serão apresentados os *crossovers*, e na próxima seção, as mutações.

Crossover aritmético (Michalewicz, 1994): dado dois cromossomos \mathbf{p}_1 e \mathbf{p}_2 , é produzido dois cromossomos \mathbf{c}_1 e \mathbf{c}_2 da seguinte forma

$$\mathbf{c}_1 = \beta \mathbf{p}_1 + (1 - \beta) \mathbf{p}_2$$

$$\mathbf{c}_2 = (1 - \beta) \mathbf{p}_1 + \beta \mathbf{p}_2$$

onde $\beta \in U(0,1)$. Este operador difere do *crossover* BLX- α por não extrapolar o intervalo entre \mathbf{p}_1 e \mathbf{p}_2

Crossover heurístico (Michalewicz, 1994): realiza uma extrapolação linear entre os pais usando a informação da aptidão.

Dado dois cromossomos \mathbf{p}_1 e \mathbf{p}_2 em que \mathbf{p}_1 é melhor do que \mathbf{p}_2 em termos de aptidão. Então é produzido um cromossomo \mathbf{c} da seguinte forma

$$\mathbf{c} = \mathbf{p}_1 + r(\mathbf{p}_1 - \mathbf{p}_2), \text{ onde } f(\mathbf{p}_1) > f(\mathbf{p}_2)$$

onde $r \in U(0,1)$. Caso o *crossover* produza um filho infactível, gera-se outro número aleatório r , e obtém-se novo filho. Se em t tentativas o filho continuar infactível, então o *crossover* pára sem produzir filhos.

Crossover simples (Michalewicz, 1994): é uma variação do *crossover* convencional de 1 ponto adaptado para representação real.

Outros Operadores

É possível encontrar na literatura operadores genéticos combinados com técnicas mais complicadas, como por exemplo operadores que levam em consideração a direção aproximada do gradiente (Gen e Cheng, 1997) ou operadores como o *crossover* quadrático (Adewuya, 1996) que usa três pais para fazer um ajuste de curvas quadrático com base no valor da aptidão.

3.2.3 MUTAÇÃO PARA REPRESENTAÇÃO REAL

Mutação uniforme: é a simples substituição de um gene por um número aleatório.

Ou seja, dado um cromossomo \mathbf{p} com o j -ésimo gene selecionado para mutação, é produzido um cromossomo \mathbf{c} da seguinte forma

$$c_i = \begin{cases} U(a_i, b_i), & \text{se } i = j \\ p_i & \text{caso contrário} \end{cases}$$

onde a_i e b_i representam os limites do intervalo permitido para o gene c_i

Mutação gaussiana: é a substituição de um gene por um número aleatório de uma distribuição normal.

Ou seja, dado um cromossomo \mathbf{p} com o j -ésimo gene selecionado para mutação, é produzido um cromossomo \mathbf{c} da seguinte forma:

$$c_i = \begin{cases} N(p_i, \sigma), & \text{se } i = j \\ p_i & \text{caso contrário} \end{cases}$$

onde $N(p_i, \sigma)$ é uma distribuição normal com média p_i e desvio padrão σ . Alternativamente, pode-se diminuir o valor de σ , à medida que aumenta a número de gerações.

Mutação *creep*: adiciona ao gene um pequeno número aleatório obtido de uma distribuição normal (com média zero e desvio padrão pequeno) ou de uma distribuição uniforme. Alternativamente, a mutação *creep* pode ser realizada, multiplicando o gene por um número aleatório próximo de um. O número aleatório deve ser pequeno o suficiente para que cause apenas pequena perturbação no cromossomo, porque estando perto do ponto máximo, tal perturbação pode movê-lo rapidamente ao topo. A taxa de mutação *creep* pode ser relativamente alta, visto que ele é usada apenas para explorar localmente o espaço de busca (a mutação *creep* não é muito destrutiva).

Mutação limite (Michalewicz, 1994): é a substituição do gene por um dos limites do intervalo permitido $[a_i, b_i]$.

Ou seja, dado um cromossomo \mathbf{p} com o j -ésimo gene selecionado para mutação, é produzido um cromossomo \mathbf{c} da seguinte forma:

$$c_i = \begin{cases} a_i & \text{se } r < 0,5 \text{ e } i = j \\ b_i & \text{se } r \geq 0,5 \text{ e } i = j \\ p_i & \text{caso contrário} \end{cases}$$

onde $r \in U(0,1)$. Note que este operador leva os genes para os limites dos intervalos permitidos $[a_i, b_i]$. Isto é feito para evitar a perda de diversidade dos filhos gerados pelo crossover aritmético que tende a trazer os genes para o centro dos intervalos permitidos.

Mutação não-uniforme (Michalewicz, 1994): é a simples substituição de um gene por um número extraído de uma distribuição não-uniforme.

Ou seja, dado um cromossomo \mathbf{p} com o j -ésimo gene selecionado para mutação, é produzido um cromossomo \mathbf{c} da seguinte forma:

$$c_i = \begin{cases} p_i + (b_i - p_i)f(G) & \text{se } r_1 < 0,5 \text{ e } i = j \\ p_i - (p_i - a_i)f(G) & \text{se } r_1 \geq 0,5 \text{ e } i = j \\ p_i & \text{caso contrário} \end{cases}$$

$$f(G) = \left(r_2 \left(1 - \frac{G}{G_{\max}} \right) \right)^b$$

Onde r_1 e $r_2 \in U(0,1)$, G é o número da geração corrente, G_{\max} é o número máximo de gerações e b é um parâmetro do sistema que determina a forma da função (o autor usou $b=6$).

Mutação não-uniforme múltipla (Michalewicz, 1994): é a simples aplicação do operador mutação não-uniforme em todos os genes do cromossomo \mathbf{p} .

3.2.4 USANDO VÁRIOS OPERADORES

Uma solução para trabalhar com vários operadores é ponderá-los para indicar quantos filhos cada operador produzirá.. Por exemplo, considere que há dois operadores crossovers e um operador de mutação e a necessidade de gerar 10 filhos para a próxima geração. Considere também que foram atribuídos para os três operadores os pesos $w_1 = 40\%$, $w_2 = 40\%$ e $w_3 = 20\%$, respectivamente. Então, o primeiro crossover produziria 4 filhos, o segundo crossover 4 filhos, e a mutação 2 filhos (note que aqui a mutação é um operador independente que produz seus próprios filhos).

Em geral, os pesos permanecem constante por todas as gerações. Alternativamente, dependendo do operador e do problema, pode-se variar os pesos em cada geração para melhorar o desempenho do algoritmo (Davis, 1991).

3.3 ALGORITMOS GENÉTICOS E OTIMIZAÇÃO CONVENCIONAL

Em geral, um problema otimização, conforme mencionado anteriormente, consiste em achar a solução que corresponda ao ponto de máximo ou mínimo de uma função $f(x,y,z,u,\dots)$ de n parâmetros, x,y,z,u,\dots . Um problema de maximização de uma função numérica pode ser transformado em um problema de minimização, e vice-versa. Por exemplo, achar o valor de x que maximiza a função $f(x) = 1 - x^2$ no intervalo $-1 \leq x \leq 1$ é o mesmo que minimizar a função $f(x)$ multiplicada por -1 , ou seja, é o mesmo que minimizar $g(x) = -f(x) = x^2 - 1$. Assim, para simplificar a exposição deste tópico, serão considerados a seguir apenas problemas de minimização (salvo quando mencionado o contrário).

O gráfico mostrado na Figura 3.18 apresenta dois pontos de mínimo em x_0 e x_1 . O ponto x_0 é denominado de **mínimo local**, pois $f(x_0) \leq f(x)$ para todo x suficientemente próximo de x_0 . O ponto x_1 é conhecido como **mínimo global**, pois $f(x_1) \leq f(x)$ para todos os valores que x possa assumir. Uma função que apresenta apenas um ponto de mínimo/máximo é denominada de **função unimodal**. Quando uma função possui mais de um ponto de mínimo/máximo, ela é denominada de **função multimodal**. Mais adiante será mostrado que, em funções multimodais complicadas, muitas técnicas de otimização possuem dificuldades para decidir se um dado ponto ótimo é local ou global.

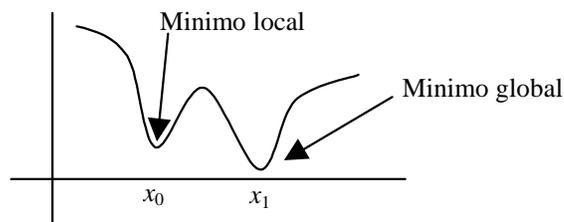


Figura 3.18 - Mínimo Local e Global

Os parâmetros da função objetivo podem ser **restritos** ou **irrestritos**. As restrições nos parâmetros restritos podem aparecer na forma de equações, como por exemplo:

$$\begin{array}{ll} \text{Minimiza} & f(x, y) \\ \text{r} & \\ \text{Sujeito a} & x^2 + y^2 = 5 \end{array}$$

ou na forma de inequações:

$$\begin{array}{ll} \text{Minimiza} & f(x, y) \\ \text{r} & \\ \text{Sujeito a} & x^2 + 2y^2 \leq 5 \end{array}$$

Quando os parâmetros são irrestritos, eles podem assumir qualquer valor. Por exemplo:

$$\begin{array}{ll} \text{Minimiza} & f(\mathbf{x}) \\ \text{r} & \\ \text{Sujeito a} & \mathbf{x} \in \mathfrak{R}^n \end{array}$$

Vários métodos de otimização trabalham melhor com parâmetros irrestritos. Em muitos casos, é possível converter um parâmetro restrito em um parâmetro irrestrito. Por exemplo, considere minimizar $f(x)$ sujeita à restrição $Min \leq x \leq Max$. O parâmetro restrito x pode ser convertido no parâmetro irrestrito u fazendo $x = Min + (Max - Min)\text{sen}^2(u)$ e minimizando f para qualquer valor de u . Quando a função objetivo e as restrições são funções lineares dos parâmetros, o problema de otimização é conhecido como problema de **Programação Linear**. Quando a função objetivo ou as restrições são funções não lineares dos parâmetros, o problema é conhecido como problema de **Programação Não-Linear**.

Os parâmetros da função objetivo podem ser **contínuos** ou **discretos**. Otimização com parâmetros contínuos tem um número infinito de soluções. Otimização de parâmetros discretos, em geral, tem somente um número finito de possíveis soluções, resultante de uma certa combinação dos parâmetros. Um exemplo é a decisão sobre a melhor ordem com que um conjunto de ações ou tarefas devem ser

realizadas. Este tipo de otimização é conhecido como **Otimização Combinatória**.

Visando situar Algoritmos Genéticos no contexto de otimização em geral, considerem-se alguns das principais classes de métodos de otimização:

Gerar-e-Testar: O algoritmo Gerar-e-testar (também conhecido como método da busca exaustiva ou aleatória) é uma abordagem da força bruta. Emprega dois módulos: o módulo de geração, que enumera possíveis soluções sistematicamente ou aleatoriamente, e o módulo de teste, que avalia cada possível solução, podendo aceitar ou rejeitá-la. O módulo gerador pode produzir todas as possíveis soluções antes do módulo de teste começar a agir. O mais comum é o uso intercalado destes dois módulos. O método pode encerrar sua execução quando uma solução satisfatória for encontrada, depois de encontrar um número de soluções satisfatórias ou continuar até que todas as possíveis soluções sejam achadas.

Métodos Analíticos: Os métodos analíticos utilizam técnicas do Cálculo Diferencial para determinar os pontos extremos de uma função e apresentam várias desvantagens: não informam se o ponto encontrado é um ponto de mínimo local ou global; requerem funções com derivadas e, além disso, quando existe grande número de parâmetros torna-se difícil encontrar, analiticamente, todos os pontos de mínimo e máximo. Isto torna estes métodos impraticáveis para otimizar diversos problemas do mundo real.

Subida de Encosta: Os métodos de Subida de Encosta (*hill climbing*) investigam os pontos adjacentes do espaço de busca e movem-se na direção que melhora o valor da função objetivo. Pode ser observado, portanto, que para funções com muitos ótimos locais, um método de Subida de Encosta terá dificuldades em localizar qual “encosta” leva ao ótimo global. Métodos de Subida de Encosta também têm dificuldades quando existem planícies ou platôs na superfície de busca. Contudo, métodos de Subida de Encosta são geralmente rápidos. Um grande número de técnicas importantes de otimização segue os princípios da Subida de Encosta. Um exemplo é o método do gradiente.

Os Algoritmos Genéticos têm sido empregados em problemas complicados de otimização, em que, muitas vezes, os métodos acima falham. Algumas vantagens dos AGs são:

- Funcionam tanto com parâmetros contínuos como discretos ou uma combinação deles.
 - Realizam buscas simultâneas em várias regiões do espaço de busca, pois trabalham com uma população e não com um único ponto.
 - Utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar.
 - Não é necessário conhecimento matemático aprofundado do problema considerado.
 - Otimizam um número grande de variáveis.
 - Otimizam parâmetros de funções objetivos com superfícies complexas e complicadas, reduzindo a incidência de mínimos locais.
 - Adaptam-se bem a computadores paralelos.
 - Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros.
 - Fornecem uma lista de parâmetros ótimos e não uma simples solução.
 - Trabalham com dados gerados experimentalmente e são tolerantes a ruídos e dados incompletos.
 - São fáceis de serem implementados em computadores.
 - São modulares e portáteis, no sentido que o mecanismo de evolução é separado da representação particular do problema considerado. Assim, eles podem ser transferidos de um problema para outro.
 - São flexíveis para trabalhar com restrições arbitrárias e otimizar múltiplas funções com objetivos conflitantes.
-

- São também facilmente hibridizados com outras técnicas e heurísticas.

Apesar dessas vantagens, os AGs não são eficientes para muitos problemas. São bastante lentos e não raro ainda estão avaliando a população inicial enquanto muitos métodos de Subida de Encosta já têm encontrado a solução. O principal campo de aplicação dos AG é em problemas complexos, com múltiplos mínimos/máximos e para os quais não existe um algoritmo de otimização eficiente conhecido para resolvê-los. A seguir, serão comparados aspectos do mecanismo de busca dos AG's com os métodos convencionais.

Os métodos Gerar-e-Testar possuem a característica de explorar pontos inteiramente novos do espaço de busca. Os métodos de Subida de Encosta, por outro lado, caracterizam-se por utilizar a informação oriunda de pontos anteriormente visitados para encontrar os melhores. Um algoritmo de otimização eficiente deve usar estas duas técnicas (denominadas de *Exploration* e *Exploitation*, respectivamente) para encontrar o ótimo global da função objetivo (Beasley et al, 1993a). Ambas as técnicas têm a mesma tradução para o português, exploração. Entretanto, *Exploitation* significa exploração no sentido de tirar de informações presentes nas soluções encontradas e *Exploration* diz respeito à exploração no sentido de visitar pontos desconhecidos no espaço de busca à procura por novas soluções.

O *crossover* e a mutação são dois mecanismos de busca dos AGs que levam à exploração de pontos inteiramente novos do espaço de busca (*exploration*). Enquanto a seleção dirige a busca em direção aos melhores pontos do espaço de busca (*exploitation*), a **pressão da seleção**, dada pela razão entre aptidão máxima da população e a aptidão média, influencia a quantidade de *Exploitation* e *Exploration*. Quando a pressão de seleção é muito baixa (i.e., a aptidão é praticamente a mesma para toda a população), o AG assume um comportamento aleatório, pois não há seleção. Isto torna o AG semelhante à busca aleatória dos métodos Gerar-e-Testar (muito *exploration*). Quando a pressão de seleção é muito alta, o AG assume o comportamento dos métodos de Subida de Encosta

(muito *exploitation*). Na prática, é difícil conhecer qual a pressão de seleção que fornece o equilíbrio ideal.

O *crossover* é o principal mecanismo de busca do AG. Ele é capaz de combinar as boas porções dos cromossomos pais, isto é, os bons blocos de construção. Como resultado, cromossomos filhos com aptidões mais elevadas que as dos pais podem ser produzidos. A mutação também desempenha importante papel no Algoritmo Genético, uma vez que permite que qualquer ponto do espaço de busca seja alcançado. A fase de seleção descarta os cromossomos com baixa aptidão e com eles muitos genes também são eliminados. Na ausência de mutação, genes presentes apenas nos cromossomos descartados não serão mais recuperados, pois o *crossover* não gera novos genes, apenas combina os que já existem.

O desaparecimento de determinados genes da população no decorrer das gerações (fenômeno denominado na biologia de *Genetic drift*) impossibilita um Algoritmo Genético de explorar completamente o espaço de busca. Como resultado, o AG pode convergir para um mínimo ou máximo local. Este problema é conhecido como **Convergência Prematura**, que tem como uma de suas principais causas o *Genetic drift*. Com taxas adequadas de mutação, é possível manter uma boa diversidade de genes da população a fim de contrabalançar estas perdas de genes.

Além dos Algoritmos Genéticos, existem outros métodos de otimização global (i.e. métodos que levam ao ótimo global), sendo a maioria deles recentes. Os métodos antigos de otimização são quase todos métodos de Subida de Encosta. Alguns exemplos são Reconhecimento Simulado, Busca Tabu e *Branch and Bound* combinado com outras técnicas (e.g. Análise Intervalar) (Stolfi e Figueiredo, 1997).

3.4 ASPECTOS AVANÇADOS

Neste item, serão comentados alguns aspectos práticos de Algoritmos Genéticos considerados importantes pelos autores.

3.4.1 POPULAÇÃO INICIAL

A população inicial pode ser gerada de várias maneiras. Se uma população inicial pequena for gerada aleatoriamente, provavelmente, algumas regiões do espaço de busca não serão representadas.

Este problema pode ser minimizado gerando a população inicial de maneira uniforme (i.e. com pontos igualmente espaçados, como se preenchessem uma grade no espaço de busca). Outra alternativa é gerar a primeira metade da população aleatoriamente e a segunda metade a partir da primeira, invertendo os bits. Isto garante que cada posição da cadeia de bits tenha um representante na população com os valores 0 e 1, como pode ser visto na Tabela 3.2.

Tabela 3.2 - Geração da população inicial

1ª metade gerada aleatoriamente	2ª metade inverte os bits da 1ª metade
1011010	0100101
0111011	1000100
0001101	1110010
1100110	0011001

Pode ser interessante usar uma população inicial maior que a utilizada nas gerações subsequentes, visando a melhorar a representação do espaço de busca.

Uma técnica denominada “*seeding*” pode ser útil em muitos problemas práticos. Consiste em colocar, na população inicial, soluções encontradas por outros métodos de otimização. Isto garante que a solução gerada pelo AG não seja pior que as soluções geradas por estes métodos.

3.4.2 FUNÇÃO OBJETIVO

A função objetivo em alguns problemas pode ser bastante complicada, demandando um alto custo computacional. Por exemplo, existem problemas em que, para avaliar um

cromossomo, é necessária uma simulação completa do processo, o que pode chegar a consumir horas. Haupt e Haupt (1998) sugerem, para lidar com tais funções objetivo, algumas dicas que propõem cuidados a serem tomados para não avaliar cromossomos idênticos mais de uma vez, reutilizando deste modo a avaliação feita anteriormente.

Isto pode ser feito de várias maneiras: 1. evitando gerar cromossomos idênticos na população inicial; 2. verificando se foi aplicado *crossover* ou mutação nos pais, pois, caso não tenham sido aplicados, os filhos serão iguais ao pais; 3. observando se o filho é igual a um dos pais; 4. mantendo a população com todos os cromossomos distintos entre si, o que também ajuda na manutenção da diversidade (no AG do tipo *Steady State* isto é feito evitando inserção de cromossomos duplicatas na população); 5. antes de avaliar um filho, verificando se já existe um cromossomo igual a este filho na população. Em situações mais extremas, devem-se armazenar todos os cromossomos das gerações atual e passada, verificando se algum deles é igual ao novo filho gerado. É claro que tais abordagens também incorporam um custo computacional extra ao AG. Deve-se analisar se este custo extra compensa o tempo economizado na avaliação da função objetivo.

Uma outra abordagem é procurar uma maneira de simplificar a função objetivo. A versão simplificada da função objetivo seria utilizada nas gerações iniciais para acelerar a busca por regiões promissoras do espaço de busca. Nas gerações finais, a versão original da função objetivo seria utilizada para melhorar a precisão da solução.

Outro método é usar o AG para localizar a encosta do máximo global, e posteriormente, substituir o AG por um Método de Subida de Encosta que rapidamente encontre a solução (os AG são bons para localizar, velozmente, regiões promissoras do espaço de busca, porém depois são lentos para refinar as soluções).

3.4.3 CRITÉRIOS DE PARADA

Alguns dos vários critérios de parada para os AGs, são: 1. quando o AG atingir um dado número de gerações (ou avaliações); 2. chegada ao valor ótimo da função objetivo, se este é conhecido; 3. convergência, isto é, quando não ocorrer melhoramento significativo no cromossomo de maior aptidão por um dado número de gerações.

Outras alternativas são também usadas, por exemplo: considere que um gene converge se 90% da população tem o mesmo valor para este gene. Se entre 90% e 95% dos genes convergiram, o AG convergiu.

3.4.4 SUBSTITUIÇÕES GERACIONAL E *STEADY- STATE*

Existem dois tipos básicos de substituição de cromossomos: substituição geracional e substituição *steady-state*.

Na substituição geracional (utilizada na Seção 3.1), toda a população é substituída em cada geração, ou seja, são criados N filhos para substituir N pais. Alternativamente, podem ser criados N filhos, e os N melhores indivíduos da união de pais e filhos substituem a população atual. Na substituição geracional com elitismo, os k melhores pais nunca são substituídos por filhos piores. Geralmente é utilizado um valor de $k = 1$. Aumentando o valor de k , aumenta-se o risco da convergência prematura.

Na substituição *steady-state*, são criados apenas dois (ou um) filhos em cada “geração”, que substituem os dois piores cromossomos da população. Alternativamente, os dois filhos podem substituir os pais ou os dois cromossomos mais velhos da população, baseado na suposição de que o cromossomo existente na população há muitas gerações, já transmitiu seus genes à população. Na forma geral da substituição *steady-state*, são criados n filhos que substituem os n piores pais. Em geral, a taxa de *crossover* é maior (≈ 1) no AG *steady-state* do que no AG geracional.

Esta abordagem agressiva utiliza os filhos para *crossover* tão logo eles sejam gerados (não há população intermediária), e

tendo apresentado bons resultados (Davis, 1991). Porém, gera um custo computacional extra, por exemplo, para cada filho criado é necessário recalcular estatísticas, a aptidão média, reordenar a população, etc. Contudo, em muitos problemas práticos, este custo extra não é significativo, pois o custo computacional está quase todo concentrado no cálculo da aptidão. Nesta técnica, é interessante evitar inserir filhos duplicados na população, pois a mesma gera, naturalmente, um grande número de filhos duplicados (Davis, 1991).

3.4.5 PROBLEMAS DE CONVERGÊNCIA

A convergência prematura é um conhecido problema dos AGs. Ocorre quando surgem cromossomos de alta aptidão (mas não com aptidão ótima), e os cromossomos realmente ótimos ainda não estão presentes na população. Tais cromossomos (chamados de **superindivíduos**) geram um número excessivo de filhos que dominam a população, uma vez que a mesma é finita. Estes cromossomos espalham seus genes por toda a população, enquanto outros genes desaparecem (tal desaparecimento de genes é denominado de *genetic drift*). Como consequência, o algoritmo converge para um máximo ou mínimo local, conforme a Figura 3.19 ilustra este processo.

Combate-se a convergência prematura, limitando o número de filhos por cromossomos. Esta limitação pode ser realizada através do escalonamento da aptidão, ordenamento e outras técnicas a serem descritas mais adiante.

Manter a diversidade dos cromossomos na população também combate a convergência prematura, visto que é também causada pela perda de diversidade. O aumento da taxa de mutação também melhora a diversidade (mais genes são criados). Evitar a inserção de filhos duplicados na população é uma outra alternativa para melhorar a diversidade.

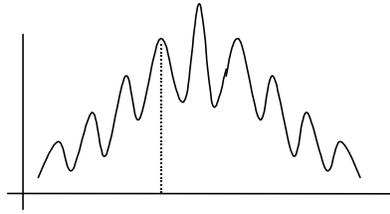


Figura 3.19 - Convergência prematura

Um outro problema que pode ocorrer quando AGs são utilizados é explicado pelo seguinte exemplo. Considere substituir a função objetivo utilizada no Problema 3.2 pela nova função objetivo (3.8). Tal substituição torna o valor da função objetivo praticamente o mesmo para toda a população, como na Tabela 3.3.

$$f(x, y) = 2000 - \frac{(\sin \sqrt{x^2 + y^2})^2 - 0,5}{(1 + 0,001(x^2 + y^2))^2}$$

Tabela 3.3 - Intervalo estreito de aptidão

Cromossomo	Função objetivo	Probabilidade de seleção
A	2.000,999588	20,004%
B	2.000,826877	20,002%
C	2.000,655533	20,001%
D	2.000,400148	19,998%
E	2.000,102002	19,995%

Neste caso, deixar a aptidão igual à função objetivo torna a seleção um processo aleatório (qualquer cromossomo tem a mesma probabilidade de seleção), ocorrendo problema semelhante ocorre nas gerações finais do algoritmo, mas por outro motivo. Grande parte da população convergiu e os cromossomos têm aptidões praticamente iguais. A seleção

torna-se também aleatória. O algoritmo tem dificuldade de melhorar a solução e converge lentamente. Este problema é resolvido expandindo o intervalo da função objetivo através de ordenamento ou outra forma de mapeamento.

3.4.6 MAPEAMENTO DA FUNÇÃO OBJETIVO

O valor da função objetivo nem sempre é adequado para ser utilizado como valor de aptidão. Por exemplo, a função objetivo pode assumir valores negativos (a roda da roleta não funciona), valores muito próximos (torna a seleção aleatória), alguns valores podem ser muito elevados em relação ao resto da população (causa a convergência prematura), etc. O mapeamento da função objetivo para o valor da aptidão pode ser feito de vários modos, alguns dos quais serão discutidos a seguir.

Ordenamento

No método **ordenamento linear**, a aptidão é dada por (Baker, 1987; Whitle, 1989):

$$f_i = \text{Min} + (\text{Max} - \text{Min}) \frac{N - i}{N - 1}$$

em que i é o índice do cromossomo na população em ordem decrescente de valor da função objetivo. Normalmente é utilizado $1 \leq \text{Max} \leq 2$ e $\text{Max} + \text{Min} = 2$. Vale notar que deste modo a aptidão representa o número de filhos esperados do cromossomo e $\text{Max} - \text{Min}$ representa a pressão de seleção (razão entre a maior aptidão e a aptidão média, f_{\max} / \bar{f}). Um exemplo é mostrado na Tabela 3.4, que expande o intervalo dos valores da função objetivo que estão muito próximos (resolvendo assim o problema mostrado na Tabela 3.3).

Tabela 3.4 - Ordenamento linear

Cromossom	Função	Posiçã	Aptidã	Probabilidade
o	objetivo	o	o	de seleção

A	2.000,99958 8	1	2,0	40%
B	2.000,82687 7	2	1,5	30%
C	2.000,65553 3	3	1,0	20%
D	2.000,40014 8	4	0,5	10%
E	2.000,10200 2	5	0,0	0%

A Figura 3.20 mostra o controle da pressão de seleção utilizando ordenamento linear. Na Figura 3.20(a), a alta pressão de seleção favorece fortemente os melhores cromossomos, direcionando a busca às melhores soluções encontradas até então (muito *exploitation*). Na Figura 3.20(b), a baixa pressão de seleção favorece um pouco mais os cromossomos de baixa aptidão, direcionando a busca para regiões desconhecidas do espaço de busca (muito *exploration*).

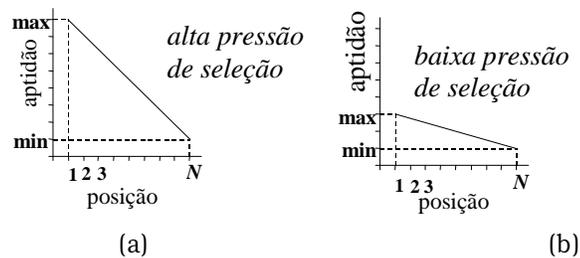


Figura 3.20 - Posição do cromossomo \times pressão da seleção

No método **ordenamento exponencial**, a aptidão é dada por (Michalewicz, 1994):

$$f_i = q(1-q)^{i-1}$$

em que $q \in [0, 1]$ e i é o índice do cromossomo na população em ordem decrescente de valor da função objetivo. Alternativamente, a aptidão pode ser normalizada dividindo a Eq.(4.3) pelo fator $1 - (1 - q)^N$. O ordenamento exponencial permite maior pressão de seleção do que o ordenamento linear.

Escalonamento Linear

No método **escalonamento linear**, a aptidão é obtida pela equação:

$$f = ag + b$$

em que g é o valor da função objetivo (ver Figura 3.21). Os coeficientes a e b são determinados, limitando o número esperado de filhos dos cromossomos (filhos em excesso causam perda de diversidade). O escalonamento linear de Goldberg (1989) transforma as aptidões de tal modo que a aptidão média torna-se igual ao valor médio da função objetivo (i.e. $\bar{f} = \bar{g}$), e a aptidão máxima igual a C vezes a aptidão média (i.e. $f_{\max} = C\bar{g}$). Tipicamente, o valor de C está entre 1,2 e 2,0. Quando o escalonamento gera aptidões negativas, os coeficientes a e b são calculados de outro modo (impondo $f_{\min} = 0$). Os resultados são resumidos na Algoritmo 3.3 (o teste $g_{\min} > (C\bar{g} - g_{\max}) / (C - 1)$, verificando se ocorre aptidão negativa).

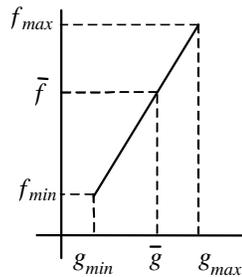


Figura 3.21 - Gráfico de escalonamento da aptidão

Algoritmo 3.3: Cálculo dos coeficientes a e b do escalonamento linear.

$\bar{g} = \frac{1}{N} \sum_{i=1}^N g_i$ Elaborar equação

se $g_{\min} > (C\bar{g} - g_{\max}) / (C - 1)$ **então** /* testa se ocorre aptidão negativa */

$\Delta \leftarrow g_{\max} - \bar{g}$
 $a \leftarrow (C - 1)\bar{g} / \Delta$
 $b \leftarrow \bar{g}(g_{\max} - C\bar{g}) / \Delta$

senão

$\Delta = \bar{g} - g_{\min}$
 $a \leftarrow \bar{g} / \Delta$
 $b = -\bar{g} g_{\min} / \Delta$

fim se

retorne a e b

3.4.7 SELEÇÃO POR TORNEIO

Na seleção por torneio, cada cromossomo é selecionado para a população intermediária do seguinte modo: são escolhidos aleatoriamente (com probabilidades iguais) n cromossomos da população e o melhor dentre estes cromossomos é selecionado. O valor $n = 2$ é usual. A seleção por torneio não precisa de escalonamento da aptidão e nem de ordenamento.

Em uma outra versão, a seleção por torneio utiliza probabilidades diferenciadas. Se o torneio envolve dois cromossomos, o primeiro ganha o torneio com probabilidade q (onde $0,5 < q < 1$); e o segundo, com probabilidade $1 - q$. Para um torneio entre n cromossomos, o primeiro cromossomo ganha o torneio com probabilidade q , o segundo com probabilidade $q(1 - q)$; o terceiro, com $q(1 - q)^2$, e assim por diante... (vale notar que se $n = N$, em que N é o tamanho da população, tal seleção é equivalente à seleção com ordenamento exponencial).

Aumentando o número n de cromossomos do torneio ou a probabilidade q do primeiro cromossomo vencer, aumenta-se

a pressão de seleção, isto é, cromossomos com aptidão acima da média terão mais chances de serem selecionados.

3.4.8 AMOSTRAGEM ESTOCÁSTICA UNIVERSAL

O algoritmo Roda da Roleta possui o problema de apresentar uma grande variância em relação ao número esperado de filhos dos cromossomos pais. A **Amostragem Universal Estocástica** ou **SUS** (do inglês, *Stochastic Universal Sampling*) (Baker, 1987) soluciona este problema de maneira simples e tão perfeita quanto possível.

Neste método, a população é embaralhada e um gráfico do tipo “torta” é construído com uma fatia associada a cada cromossomo da população. A área das fatias é proporcional à aptidão do cromossomo que ela representa. Em volta da parte externa da “torta” são colocados N ponteiros igualmente espaçados. Por fim, o cromossomo apontado por cada ponteiro é selecionado para *crossover* e mutação (ver Figura 3.22). Na prática, os cromossomo selecionados podem ser alocados em uma população intermediária e então a cada dois cromossomos é aplicado o *crossover*

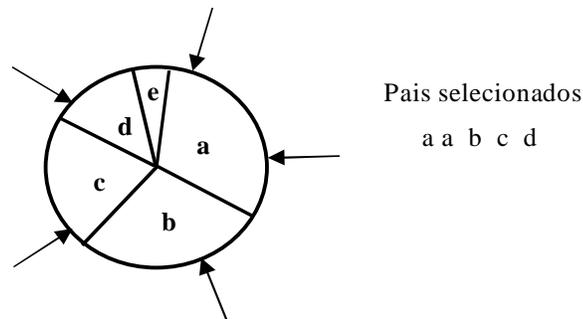


Figura 3.22 - Amostragem Universal Estocástica

3.4.9 OTIMIZAÇÃO MULTI OBJETIVO

Muitos problemas do mundo real envolvem a otimização de mais de uma função objetivo. Uma abordagem simples para tratar de um conjunto de funções objetivo f_1, f_2, \dots, f_n , é formar

uma nova função objetivo F , obtida da soma ponderada das funções objetivo, ou seja:

$$F = \sum_{i=1}^n w_i f_i$$

em que w_i é um peso (que pode ser usado pelo usuário para dar mais relevância a uma função do que às outras). A minimização de F minimiza todas as funções f_i . No entanto, as funções objetivo podem ser conflitantes, no sentido de que otimizar uma função degrada a otimização da outra função (e.g. minimizar custos de um produto e maximizar sua qualidade tem, em geral, objetivos conflitantes).

Uma melhor abordagem seria usar o conceito de **Pareto-ótimo** (Fonseca e Fleming, 1995, 1997) (devido a Vilfredo Pareto, um dos primeiros a usar tal conceito). Este conceito introduz a idéia de **dominação** e **não dominação**. Uma solução \mathbf{x}_1 é dita não dominada por uma solução \mathbf{x}_2 , quando ela, em todas as funções, é no mínimo tão boa quanto a solução \mathbf{x}_2 , e em pelo menos em uma das funções, \mathbf{x}_1 é melhor que \mathbf{x}_2 , ou seja:

$$(\forall i: f_i(\mathbf{x}_1) \geq f_i(\mathbf{x}_2)) \text{ E } (\exists j: f_j(\mathbf{x}_1) > f_j(\mathbf{x}_2))$$

A meta do AG é encontrar o conjunto Pareto-ótimo (também denominado de **fronteira Pareto-ótimo**), que é o conjunto de todas as soluções não-dominadas do espaço de busca. Seja S o espaço de busca, o conjunto Pareto-ótimo P é dado por:

$$P = \{\mathbf{x}_i \in S \mid \nexists \mathbf{x}_j \in S : \mathbf{x}_i \text{ é dominado por } \mathbf{x}_j\}$$

Exemplo: Considere um problema com duas funções objetivo, $f_1(x)$ e $f_2(x)$, em que todas as soluções possíveis são mostradas na Tabela 3.5. O conjunto Pareto-ótimo é dado pela soluções $\{2,4,8\}$, uma vez que a solução $x=10$ é dominada por $x=8$, e $x=12$ é dominado por $x=4$.

Tabela 3.5 - Duas funções objetivo

x	$f_1(x)$	$f_2(x)$
2	80	30
4	20	100
8	30	50
10	30	40
12	20	80

Vale notar que o AG retornará ao usuário um conjunto de soluções ótimas ao invés de uma única solução. A aptidão pode ser calculada utilizando ordenamento com base na não-dominância de indivíduos, conforme sugerido pelo seguinte procedimento: atribui-se a posição 1 a todos os indivíduos não dominados da população. Em seguida, tais indivíduos de posição 1 são removidos, temporariamente, da população. Na que restou, atribui-se a posição 2 aos indivíduos não dominados. Em seguida, retira-se temporariamente os indivíduos de posição 2 da população. Na restante, repete-se o procedimento, atribuindo a posição 3 aos indivíduos não dominados, e assim por diante até tornar-se vazia.

A otimização com o Pareto-ótimo tem sido usado com nichos, visando a estabilizar o grande número de subpopulações que surgem no conjunto Pareto-ótimo (Horn e Nafpliotis, 1992).

3.4.10 RESTRIÇÕES

Vários problemas de otimização do mundo real contêm restrições. Em geral, um problema de otimização restrita pode ser declarado como segue:

$$\begin{aligned}
 &\text{Minimize } f(\mathbf{x}) \\
 &\text{Sujeito a } \begin{aligned} &g_1(\mathbf{x}) \leq 0 \\ &g_2(\mathbf{x}) \leq 0 \\ &\dots \\ &g_m(\mathbf{x}) \leq 0 \end{aligned} \\
 &\mathbf{x} \in \mathcal{R}^n
 \end{aligned}$$

Uma abordagem para lidar com restrições em AG é simplesmente atribuir aptidão zero aos cromossomos inactivos (i.e. aqueles que não satisfazem as restrições). Porém, os cromossomos inactivos próximos das regiões factíveis podem conter informações importantes para gerar filhos factíveis. Portanto, ao invés de zerar a aptidão de tais cromossomos, pode-se apenas penalizar a aptidão. Para isto, usa-se uma função de pênalti $\Phi_j(\mathbf{x})$, que define o quão a solução \mathbf{x} viola a restrição j . (i.e. o quão distante o cromossomo inactivo está da região factível).

As funções de pênalti podem ser definidas como segue: quando a restrição j é uma inequação, a função de pênalti é dada por:

$$\Phi_j(\mathbf{x}) = \max(0, g_j(\mathbf{x}))$$

Quando a restrição j é uma equação, a função de pênalti é dada por:

$$\Phi_j(\mathbf{x}) = |g_j(\mathbf{x})|$$

Agora o problema de otimização mencionado acima torna-se irrestrito com a soma da antiga função objetivo com o termo de pênalti, isto é:

$$\begin{aligned} \text{Minimize } & h(\mathbf{x}) = f(\mathbf{x}) + r \sum_{j=1}^m \Phi_j^2(\mathbf{x}) \\ \text{Sujeito a } & \mathbf{x} \in \mathcal{X}^n \end{aligned}$$

onde r é a constante de pênalti (serve para controlar o tamanho da penalidade).

Uma outra abordagem é fazer de cada restrição uma função objetivo diferente e resolver o problema como um problema de otimização multiobjetivo (Camponogara e Talukdar, 1997). Mais detalhes sobre restrições ver (Michalewicz, 1997a, 1997b).

3.4.11 AS FUNÇÕES DE TESTE DE DEJONG

Existem muitas variações de AGs, cada uma delas é mais indicada para um determinado tipo de problema. Para permitir

comparações entre os vários AGs, é importante testá-los sob o mesmo conjunto de funções. A literatura de AGs é repleta destas funções.

DeJong (1975), na sua tese de doutorado, propôs e utilizou um conjunto de funções para testar vários AGs. Tal conjunto foi empregado por muitos pesquisadores e ainda hoje, freqüentemente, utilizado na literatura para testes e comparações. O conjunto é formado por 5 funções simples com vários tipos de superfícies: contínua/descontínua, convexa/não-convexa, unimodal/multimodal, determinística/estocástica e de baixa/alta dimensionalidade. A Tabela 3.6 apresenta as 5 funções propostas.

Tabela 3.6 - Funções de teste de DeJong

Nome	Função	Limites	Características
F1	$\sum_{i=1}^3 x_i^2$	$-5,12 \leq x_i \leq 5,12$	Unimodal e quadrática. Com mínimo em $f(0,0,0) = 0$
F2	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2,048 \leq x_i \leq 2,048$	Unimodal. É a função clássica de Rosenbrock em duas dimensões. Com mínimo em $f(1,1) = 0$
F3	$\sum_{i=1}^5 \text{inteiro}(x_i)$	$-5,12 \leq x_i \leq 5,12$	Descontínua. Mínimo em $f(0,0,0,0,0) = 0$
F4	$\sum_{i=1}^{30} ix_i^4 + \text{gaussiana}(0,1)$	$-1,28 \leq x_i \leq 1,28$	Alta dimensionalidade, estocástica. Com mínimo em $f(0, \dots, 0) = 0$

F5	$0,002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^2 (x_i - a_{ij})^6}$	$-65,536 \leq x_i \leq 65,536$	Extremamente multimodal, com picos agudos.
----	---	--------------------------------	--

Todas estas funções representam problemas de minimização. Na função F3, a função inteiro(x) arredonda o valor de x para o inteiro mais próximo. Na função F4, a função gaussiana (0,1) gera um número aleatório obtido de uma distribuição normal com média zero e desvio padrão 1. Na função F5, os coeficientes a_{ij} são constantes determinadas pelo usuário (sugestão: usar todos $a_{ij} = 10$).

3.5 PROBLEMA DO CAIXEIRO VIAJANTE

O objetivo deste item é mostrar como aplicar AGs em problemas de natureza diferente daqueles anteriormente descritos (otimização de funções numéricas). Por exemplo, problemas que dependem da ordem com que ações ou tarefas são executadas. Tais problemas têm sido exaustivamente estudados na literatura de AGs e têm levado a proposta de vários operadores genéticos específicos. Para ilustrar seu uso, estes operadores são aplicados a um problema conhecido de otimização combinatória: o problema do caixeiro viajante (PCV).

Este problema é NP-Difícil, o que significa que os algoritmos conhecidos para encontrar sua solução exata são intratáveis pelo computador (i.e. requerem uma quantidade de tempo computacional que aumenta exponencialmente com o tamanho do problema). Problemas NP-Difícil têm sido resolvidos com algoritmos heurísticos (por exemplo, Algoritmos Genéticos) que não garantem achar a solução exata, mas que reduzem o tempo de processamento.

O PCV é descrito como segue. Existe uma lista de cidades que um vendedor precisa visitar, devendo fazer a visita em cada uma única vez. Cada par de cidades é ligado por uma estrada.

O objetivo deste problema é encontrar o caminho mais curto que o vendedor deve seguir, começando em uma cidade e terminando na mesma (tal cidade pode ser qualquer uma da lista).

A alternativa mais simples para encontrar o caminho mais curto seria a realização de uma busca exaustiva, isto é, a verificação de todos os caminhos possíveis. Esta alternativa funciona quando existem poucas cidades, no entanto, com o aumento do número de cidades, esta abordagem torna-se proibitiva.

Para N cidades, o número total de caminhos possíveis é igual a $(N-1)!$. O tempo para checar cada caminho é proporcional a N . Como resultado, o tempo total de busca é proporcional a $N \times (N-1)! = N!$. Para uma lista de apenas 30 cidades, a busca exaustiva implica na verificação de $30!$ caminhos (que é um número de magnitude astronômica), tornando a busca impraticável.

A função objetivo natural para este problema é o comprimento total do caminho (que começa em uma cidade e retorna a mesma), que é dado por:

$$f(x, y) = \sum_{i=1}^N \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

Uma maneira de resolver o PCV via AGs é representar cada cromossomo por uma lista de cidades.

Por exemplo, considere as cidades representadas pelas letras A, B, ..., G na Figura 3.18, com um possível caminho a ser seguido pelo vendedor:

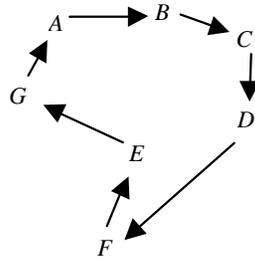


Figura 3.18 - Uma possível rota de viagem para o vendedor do PCV

Tal caminho seria representado por um cromossomo como:

Cromossomo
= A B C D F E G

Vale notar que fazendo permutações na lista de cidades, podem-se obter diferentes caminhos que são soluções potenciais para o PCV. Têm sido propostos vários operadores genéticos para realizar estas permutações. Na seção, a seguir, são relacionados alguns destes operadores.

3.5.1 OPERADORES GENÉTICOS PARA PERMUTAÇÕES

Uma permutação de m elementos é uma seqüência de m elementos em que nenhum elemento é repetido. Por exemplo, (A,B,C) e (C,A,B) são exemplos de duas permutações dos elementos A,B e C . Um grande número de problemas de otimização combinatória pode ter suas soluções representados por permutações, entre os quais o PCV e problemas de agendamento. Existem vários operadores genéticos que realizam permutações (Goldberg, 1989) (Syswerda, 1991), sendo alguns deles descritos a seguir.

Considere um cromossomo como uma lista de elementos, por exemplo:

Cromossomo : A B C D F E G

Os operadores de mutação para permutações são relativamente simples. Na mutação baseada na posição, dois elementos do cromossomo são escolhidos, aleatoriamente, e o segundo é colocado antes do primeiro. Na mutação baseada na ordem, dois elementos do cromossomo são escolhidos aleatoriamente, e suas posições são trocadas. A mutação por embaralhamento começa escolhendo aleatoriamente dois cortes no cromossomo. Depois os elementos na sublista entre os cortes são embaralhados, por exemplo:

Cromossomo : A B | C D F | E G

Após a mutação : A B | F C D | E G

Existem vários operadores de *crossover* para permutação. Alguns deles (Goldberg, 1989) (Syswerda, 1991) são apresentados a seguir:

- OBX (*Order- Based Crossover*);
- PBX (*Position- Based Crossover*);
- PMX (*Partially Matched Crossover*);
- CX (*Cycle Crossover*);
- OX (*Order Crossover*).

Crossover OBX

O *crossover* OBX começa selecionando um conjunto de posições aleatoriamente (cada posição tem uma probabilidade igual a 0,5 de ser selecionada). Depois, é imposto aos elementos de pa_i , nas posições selecionadas, o mesmo ordenamento que estes mesmos elementos apresentam em pa_i . Em seguida, o novo ordenamento nas posições selecionadas de pa_i é

copiado para $filho_1$. Os elementos nas posições não selecionadas de pai_1 são copiados sem alterações para $filho_1$. O cromossomo $filho_2$ é obtido através de um processo similar, por exemplo:

Pai_1 :	A	B	C	D	F	E	G
Pai_2 :	C	E	G	A	D	F	B
		*		*	*		
$Filho_1$:	A	D	C	F	B	E	G
$Filho_2$:	C	A	G	D	E	F	B

Crossover PBX

O *crossover* PBX também começa selecionando um conjunto de posições aleatórias. Porém, ao invés de impor a ordem, impõe a posição. É imposto, nas posições selecionadas, que $filho_1$ tenha os mesmos elementos de pai_2 . Os demais elementos de $filho_1$ provêm de pai_1 mantendo o mesmo ordenamento presente em pai_1 (já que manter um elemento em mais de uma posição não é possível). $Filho_2$ é obtido através de processo similar. Por exemplo:

Pai_1 :	A	B	C	D	F	E	G
Pai_2 :	C	E	G	A	D	F	B
		*		*	*		
$Filho_1$:	B	E	C	A	D	F	G
$Filho_2$:	C	B	E	D	F	G	A

Crossover PMX

O *crossover* PMX inicia com dois pontos de corte escolhidos aleatoriamente, que definem uma sublista. Em seguida, este operador realiza trocas no sentido de pai_1 para pai_2 e depois no sentido inverso, isto é, de pai_2 para pai_1 , para evitar cromossomos inválidos. O

exemplo seguinte ilustra este procedimento. Considere os seguintes pais e os dois cortes:

$$\begin{array}{l} \text{Pai}_1 : \quad A \ B \ | \ C \ D \ F \ | \ E \ G \\ \text{Pai}_2 : \quad C \ E \ | \ G \ A \ D \ | \ F \ B \end{array}$$

A primeira troca ocorre no início da sublista, no sentido de pai_1 para pai_2 : C de pai_1 é trocado com G de pai_2 . Como esta troca gera elementos duplicados, então ao mesmo tempo C de pai_2 é trocado com G de pai_1 .

$$\begin{array}{l} \text{Pai}_1 : \quad A \ B \ | \ G \ D \ F \ | \ E \ C \\ \text{Pai}_2 : \quad G \ E \ | \ C \ A \ D \ | \ F \ B \end{array}$$

Um procedimento similar ocorre na segunda posição da sublista: D de pai_1 é trocado com A de pai_2 . Simultaneamente D de pai_2 é trocado com A de pai_1 .

$$\begin{array}{l} \text{Pai}_1 : \quad D \ B \ | \ G \ A \ F \ | \ E \ C \\ \text{Pai}_2 : \quad G \ E \ | \ C \ D \ A \ | \ F \ B \end{array}$$

Seguindo o mesmo processo, F de pai_1 é trocado com A de pai_2 . Simultaneamente A de pai_2 é trocado com F de pai_1 . O resultado final é dado por:

$$\begin{array}{l} \text{Filho}_1 : \quad D \ B \ | \ G \ F \ A \ | \ E \ C \\ \text{Filho}_2 : \quad G \ E \ | \ C \ D \ F \ | \ A \ B \end{array}$$

Crossover CX

O *crossover* CX começa copiando o primeiro elemento de pai_1 para filho_1 (alternativamente, pode-se começar copiando um elemento qualquer da lista):

Pai_1 : A B C D F E G
 Pai_2 : C E G B D F A

$Filho_1$: A _ _ _ _ _ _

Para evitar a duplicação de C no $filho_2$ é requerido que C de pai_1 seja copiado para $filho_1$.

$Filho_1$: A _ C _ _ _ _

Do mesmo modo, G de pai_1 é copiado para $filho_1$:

$Filho_1$: A _ C _ _ _ G

Seguindo o mesmo processo, A de pai_1 deve ser copiado para $filho_1$. Porém, como A já foi copiado para $filho_1$, o ciclo termina.

Na etapa final, as posições que ficaram em branco, são obtidas por simples troca de elementos entre pai_1 e pai_2 , resultando em:

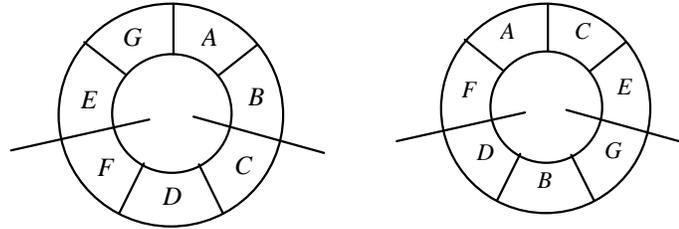
$Filho_1$: A E C B D F G
 $Filho_2$: C B G D F E A

Crossover OX

O *crossover* OX inicia com dois cortes escolhidos aleatoriamente.

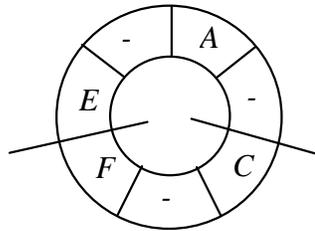
Pai_1 : A B | C D F | E G
 Pai_2 : C E | G B D | F A

Seu funcionamento é melhor ilustrado interpretando o cromossomo como um círculo, onde o primeiro e o último elementos se juntam.



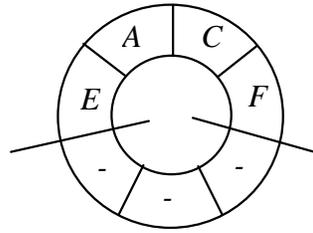
$$Pai_1 : A B \parallel C D F \parallel E G \quad Pai_2 : C E \parallel G B D \parallel F A$$

Inicialmente, os elementos de pai_1 que correspondem aos elementos da sublista entre os dois cortes de pai_2 são deletados.



$$Pai_1 : \quad A \quad - \quad | \quad C \quad - \quad F \quad | \quad E \quad -$$

Em seguida, os situados na sublista de pai_1 que não foram deletados são movidos no sentido anti-horário de modo que todos os elementos fiquem juntos a partir do segundo corte, ou seja:



Pai_1 : C F | - - - | E A

Finalmente, os espaços vazios são substituídos pela sublista entre os dois cortes de pai_2 , resultando no $filho_1$:

$Filho_1$: C F | G B D | E A

Por processo similar é obtido $filho_2$:

$Filho_2$: G B | C D F | A E

Vale notar que no *crossover* OX o que conta é a ordem dos elementos e não sua posição, sendo válido no caso do PCV, pois o que importa é somente a ordem das cidades visitadas.

3.5.2 PROBLEMAS DE PERMUTAÇÃO

Além do PCV, vários problemas de otimização combinatória podem ser resolvidos usando os operadores de permutação, pois dependem de alguma forma de permutação de tarefas ou ações. Exemplos de problemas resolvidos usando tais operadores são o Problema de Agendamento (Syswerda, 1991) e o Problema Coloração de Grafos (Davis, 1991).

3.6 FONTES ADICIONAIS DE CONSULTA

Há vários livros de AG na literatura, desde de clássicos como Goldberg (1989), passando por bons livros texto básicos tais como Michalewicz (1994), Mitchell (1998), Haupt e Haupt (1998), até livros avançados, como Bäck et al. (1997). Há também muitos tutoriais, como por exemplo Beasley et al. (1993a, 1993b), Whitley (1992) e Janikow e Clair (1995). Na *web* um dos sites mais conhecidos é o ENCORE (1998).

REFERÊNCIAS BIBLIOGRÁFICAS

- Bäck, T.; Fogel, D.B. e Michalewicz, Z. (1997). (eds), *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, Bristol, New York.
- Baker, J. (1987). Reducing bias and inefficiency in the selection algorithm. In: GREFENSTETTE, J. ed., *Proc. of the Second International Conference on Genetic Algorithms and Their Applications*, p.14-21. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Beasley, D.; Bull, D. R.; Martin, R. R. (1993a) *An Overview of Genetic Algorithms: Part 1, Fundamentals*. *University Computing*, v.15, n.2, p.58-69. (Disponível por ftp no ENCORE no arquivo: GA/papers/over92.ps.gz)
- Beasley, D.; Bull, D. R.; Martin, R. R. (1993b). *An Overview of Genetic Algorithms: Part 2, Research Topics*, *University Computing*, 15(4) 170-181. (Disponível por ftp no ENCORE no arquivo: GA/papers/over93-3.ps.gz)
- Darwin, C. *A origem das espécies e a seleção natural*. Hemus editora.
- Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.
- Dejong, K. (1975). The analysis and behaviour of a class of genetic adaptive systems. PhD thesis, University of Michigan.
-

-
- ENCORE. The Evolutionary Computation REpository network. (1998). (<http://alife.santafe.edu/~joke/encore/>) (<http://www.cs.purdue.edu/coast/archive/clife/Welcome.html>).
- Eshelman, L. J.; Shaffer, D. J. (1992). Real-coded genetic algorithms and interval-schemata. In: WHITLEY, D. L. (ed). *Foundations of Genetic Algorithms 3*. San Mateo, CA: Morgan Kaufman, p.187-203.
- Fonseca. C. M.; Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*. 3(1):1-16. (<http://www.lania.mx/~ccoello/EMOO/EMOObib.html>).
- Fonseca. C. M.; Fleming, P. J. (1997). Multiobjective optimization. In: BÄCK, T.; FOGEL, D. B.; MICHALEWICZ, Z., eds. *Handbook of Evolutionary Computation*, p.C4.5:1-9. Institute of Physics Publishing and Oxford University Press, Bristol, New York.
- Gen, M.; Cheng, R. (1997). *Genetic Algorithms and Engineering Design*. Wiley Series in Engineering Design and Automation, John Wiley & Sons.
- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.
- Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. *IEEE trans SMC*. v16, p.122-128.
- Haupt, R. L.; Haupt, S. E. (1998). *Practical Genetic Algorithms*. Wiley-Interscience.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press.
- Horn, J.; Nafpliotis, N. (1992). Multiobjective optimization using the niched pareto genetic algorithm. Technical Report IllIGAL 93005, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL. (<http://gal4.ge.uiuc.edu/illigal.home.html>)
-

- Janikow, C. Z.; Clair, D. S. (1995). Genetic algorithms simulating nature's methods of evolving the best design solution. *IEEE Potentials*, v14, p.31-35.
- Lacerda, E.; Carvalho, A. (1997). *Combinando os Algoritmos Genético e K média para Configurar Redes de Funções Base Radial*. Anais do IV Simpósio Brasileiro de Redes Neurais, páginas 95-97, Goiânia, Brasil.
- Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. 3.ed. Springer-Verlag.
- Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. The MIT Press.
- Stolfi, J.; Figueiredo, L. H. *Métodos numéricos auto-validados e aplicações*. 21º Cólóquio Brasileiro de Matemática, IMPA, 1997.
- Syswerda, G. (1989). Uniform crossover in genetic algorithms. In: J. D. Schaffer, ed. *Proceedings of the Third International Conference on Genetic Algorithms*, p.2-9. Morgan Kaufmann.
- Syswerda, G. (1991). Schedule optimization using genetic algorithms. In: DAVIS, L. *Handbook of Genetic Algorithms*. p.332-349, Van Nostrand Reinhold.
- Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In: SCHAFFER J., ed. *Proceedings of the Third International Conference on Genetic Algorithms*, p.116-121. San Mateo, Calif.: Morgan Kaufmann.
- Whitley, D. L. (1992). A genetic algorithm tutorial. Technical Report CS-93-103, Colorado State University. (Disponível por ftp no ENCORE no arquivo: GA/papers/tutor92.ps).
-