Um Tutorial sobre Algoritmos Genéticos.

Article · January 2002

Source: DBLP

CITATIONS
11

READS
1,019

3 authors, including:

Andre de Carvalho
University of São Paulo
414 PUBLICATIONS **7,820** CITATIONS
SEE PROFILE

Teresa Bernarda Ludermir
Federal University of Pernambuco
441 PUBLICATIONS **4,998** CITATIONS
SEE PROFILE

Um Tutorial sobre Algoritmos Genéticos

E. G. M. de Lacerda *

André C. P. L. F. de Carvalho †

Teresa B. Ludermir **

Resumo

Este documento apresenta os principais conceitos de Algoritmos Genéticos e sua aplicação para a solução de problemas teóricos e práticos. Algoritmos Genéticos constituem uma técnica de busca e otimização global baseada nos principios de evolução natural e genética.

Keywords: Algoritmos Genéticos, Otimização, Problema do Caixeiro Viajante

*Centro de Informática
Universidade Federal de Pernambuco
e-mail: egml@cin.ufpe.br

†ICMC - Universidade de São Paulo
São Carlos, SP, Brazil
e-mail: andre@icmc.usp.br

**Centro de Informática
Universidade Federal de Pernambuco
e-mail: tbl@cin.ufpe.br

1. Introdução aos Algoritmos Genéticos

1.1 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 [21] e popularizados por um dos seus alunos, David Goldberg [16]. 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:

$$\begin{aligned} &\text{Maximizar} && f(x) = x\sin(10\pi x) + 1 \\ &\text{Sujeita a} && -1 \leq x \leq 2 \end{aligned} \tag{1.1}$$

Embora aparentemente simples, o Problema (1.1) 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 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., uma grande quantidade de técnicas de otimização (por exemplo, os métodos do gradiente) não é capaz de localizar o ponto de

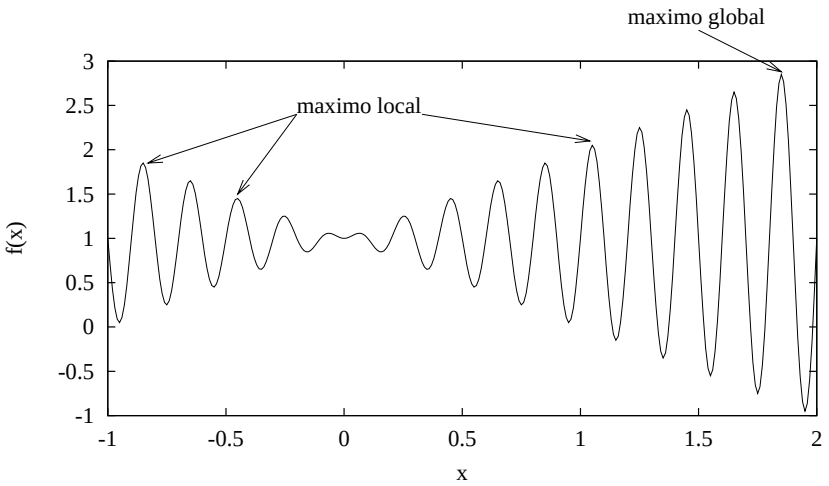


Figura 1: Gráfico da função $f(x) = x\sin(10\pi x) + 1$

máximo global de uma função com múltiplos pontos de máximo. Esta seção mostrará como utilizar AGs para encontrar o máximo global deste problema.

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. Os cromossomos mais aptos são selecionados e os menos aptos são descartados. Os membros selecionados podem sofrer modificações em suas estruturas 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 da Figura 2 ilustra este procedimento. As seções seguintes descrevem com mais detalhes cada etapa deste algoritmo.

1.2 Cromossomo

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

```
/* Seja S(t) a população de cromossomos na geração t." */
t ← 0
inicializar S(t)
avaliar S(t)
ENQUANTO o critério de parada não for satisfeito FAÇA
    t ← 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
```

Figura 2: Algoritmo Genético típico

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 (1.1) 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:

$$b_{10} = (1000101110110101000111)_2 = 2.288.967$$

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:

$$x = \min + (\max - \min)\frac{b_{10}}{2^l - 1} \tag{1.2}$$

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

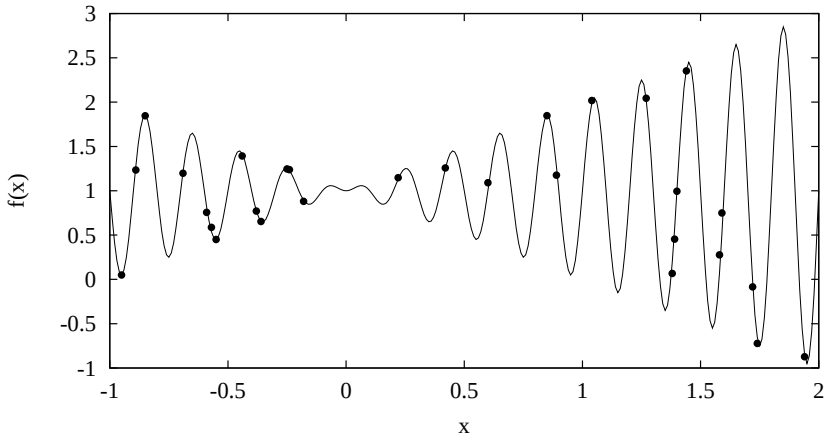


Figura 3: Cromossomos da População Inicial

quão boa é a solução codificada em s_i. É baseada no valor da função objetivo, e será discutida na próxima seção.

1.3 Seleção

Um Algoritmo Genético começa com uma população inicial de N cromossomos. A população inicial do Problema (1.1), com $N = 30$, é mostrada em uma das colunas da Tabela 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.

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 s_i com aptidão f_i, é dada por:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{1.3}$$

A seleção pode ser feita pelo seguinte procedimento prático: calcula-se uma coluna

Total $\leftarrow \sum_{i=1}^{N} f_i$
Rand $\leftarrow$ randômico$(0, \text{Total})$
TotalParcial $\leftarrow 0$
i $\leftarrow 0$
REPETIR
    $i \leftarrow i + 1$
    TotalParcial $\leftarrow$ TotalParcial $+ f_i$
ATÉ TotalParcial $\geq$ Rand
retornar o cromossomo s_i

Figura 4: Algoritmo da Roleta

de aptidões acumuladas em uma tabela, como ocorre, por exemplo, na Tabela 1. Em seguida, gera-se um número aleatório r (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 (seguindo 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 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 *Algoritmo da Roleta*, é apresentado na Figura 4.

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:

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

Vale observar que a aptidão definida desta forma pode assumir valores negativos e o algoritmo da Roleta não funciona com aptidões negativas. Além disso, pode gerar também problemas como convergência prematura (conforme será mostrado na Seção4.4). É possível, no entanto, dispensar o algoritmo 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 $n = 2$. Os outros métodos serão mostrados na seção 4..

Outra forma de definir aptidão é pelo ordenamento dos cromossomos na população, conforme se obteve na Tabela 1. O primeiro cromossomo do ordenamento recebeu uma

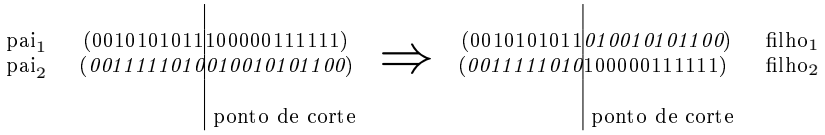
Tabela 1: População Inicial

Posição i	**Cromossomo** s_i	x_i	**Função objetivo** $f(x_i)$	**Aptidão** f_i	**Aptidão acumulada** $\sum_{k=1}^{i} f_k$
1	11010000000111110110111	1,43891	2,35251	2,00000	2,00000
2	11000001101001000111111	1,26925	2,04416	1,93103	3,93103
3	10101110010101100100000	1,04301	2,01797	1,86207	5,79310
4	10011110000110010001010 1	0,85271	1,84962	1,79310	7,58621
5	10011101101110000111100	0,84829	1,84706	1,72414	9,31035
6	00001100111110100101100	-0,84792	1,84610	1,65517	10,96552
7	00110000001001110100100	-0,43570	1,39248	1,58621	12,55172
8	01111001010000011011000	0,42098	1,25777	1,51724	14,06897
9	01000000001100111010000	-0,24764	1,24695	1,44828	15,51724
10	01000000100011110111100	-0,24343	1,23827	1,37931	16,89655
11	00001001010000000111010	-0,89156	1,23364	1,31035	18,20690
12	00011010011000101011111	-0,69079	1,19704	1,24138	19,44828
13	10100001100110000110 11	0,89370	1,17582	1,17241	20,62069
14	01101000010110110001000	0,22292	1,14699	1,10345	21,72414
15	10001000111100010000011	0,60479	1,09057	1,03448	22,75862
16	11001100110010100011100	1,39988	0,99483	0,96552	23,72414
17	01000110010001000111101	-0,17655	0,88140	0,89655	24,62069
18	00110100111101001010000	-0,37943	0,77149	0,82759	25,44828
19	00100011010011001011000	-0,58633	0,75592	0,75862	26,20690
20	11011101011011111111111	1,59497	0,74904	0,68966	26,89655
21	00110110110011011101100	-0,35777	0,65283	0,62069	27,51724
22	00100100010011111001111	-0,57448	0,58721	0,55172	28,06897
23	11001011101100111111000	1,38714	0,45474	0,48276	28,55172
24	00100110011001101001111	-0,54999	0,45001	0,41379	28,96552
25	11011100100101001000 01	1,58492	0,27710	0,34483	29,31035
26	11001010110001110100 11	1,37631	0,06770	0,27586	29,58621
27	00000100001001001100 01	-0,95144	0,04953	0,20690	29,79310
28	11101000010000000100 01	1,72169	-0,08458	0,13793	29,93103
29	11101010001111000000000	1,74494	-0,72289	0,06897	30,00000
30	11111011000000010101 11	1,94147	-0,87216	0,00000	30,00000

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, $f_i = 2\frac{(N-i)}{(N-1)}$, sendo N o tamanho da população.

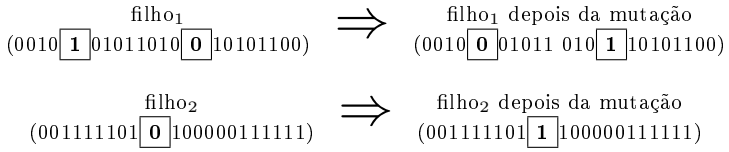
1.4 Crossover e Mutação

Os operadores de crossover e de 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 seguir é ilustrado a aplicação do 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 (permitindo que algumas soluções sejam preservadas). Isto pode ser implementado, gerando números pseudo-aleatórios no intervalo $[0, 1]$. Assim, o crossover só ocorre 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, a cada um dos bits 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. Abaixo é apresentado 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 de cromossomos.

Na Figura 5 é ilustrado 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 6. Ainda não houve melhora significativa com relação à população inicial, ou seja, os cromossomos ainda estão distantes do máximo global da função.

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 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 bem definido para terminar a execução do AG. Porém, com 95% dos cromossomos representando o mesmo valor, é possível dizer que o algoritmo convergiu.

1.5 Elitismo

Na Figura 8 é ilustrado, 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. Afinal, 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) [9] em um dos trabalhos pioneiros sobre AGs.

Na Figura 9 é mostrado 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.

1.6 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 1.4. O de 2 pontos é apresentado na Figura 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 11. O crossover de n pontos mais usado tem sido o de 2 pontos por algumas razões mostradas na Seção 1.8.

O crossover uniforme é apresentado na Figura 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 produção 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,

Índice Original	População Intermediária (*Mating pool*)	*Crossover* e Mutação	Primeira Geração (o sublinhado indica mutação)	Ponto de Corte
1	11010000000111110110111		11010000000110000011100	12
5	10011101101111000011100		10011101101111110110111	12
12	00011010011000101011111		00011010011<u>0</u>010010110	12
6	00001100111111010010110		00<u>1</u>01100111100<u>0</u>0101111	12
19	00100011010011001011100		0<u>1</u>10001101000100011101	9
17	01000110010001000111101	•	01000110010011001011100	9
15	10001000111100010000011	•	10001000111100010100010	17
7	00110000001001110100010	•	00<u>0</u>10000001001110000011	17
10	01000000100011110111110		01000000100011111011100	16
8	01111001010000011011100		01111001010000010111110	16
6	00001100111111010010110		00001100111111010010110	21
18	00110100111101001010000		00110100111101001<u>0</u>0000	21
9	01000000001100111010000		0100<u>1</u>0000<u>0</u>10010101111	13
12	00011010011000101011111		00011010011000111010000	13
17	01000110010001000111101		01000110010001000100010	17
7	00110000001001110100010		00110<u>1</u>0001001110111101	17
11	00001001010000000111010	•	00001110010101100100<u>1</u>	3
3	10101110010101100100000	•	10101001010<u>1</u>0000<u>0</u>1101<u>1</u>	3
3	10101110010101100100000	•	10101110010101100101111	19
1	11010000000111110110111		11010001000111101100000	19
15	10001000111100010000011		10001000100110010001010	9
4	10011110000110010001010		10011110011100010000011	9
13	10100001100110000011011		10100001100110000011010	20
6	00001100111111010010110		0<u>1</u>0011<u>1</u>0111111010010110	20
15	10001000111100010000011		10000100001001001100001	3
27	00000100001001001100001		00001000111100010000011	3
8	01111001010000011011100		011<u>00</u>0010100001<u>1</u>1101100	19
19	00100011010011001011100		<u>1</u>0100011010011001010<u>0</u>00	19
8	01111001010000011011100		01111001010001<u>1</u>00111010	5
11	00001001010000000111010		00001001010000<u>1</u>1101100	5

Figura 5: Gerando uma nova população

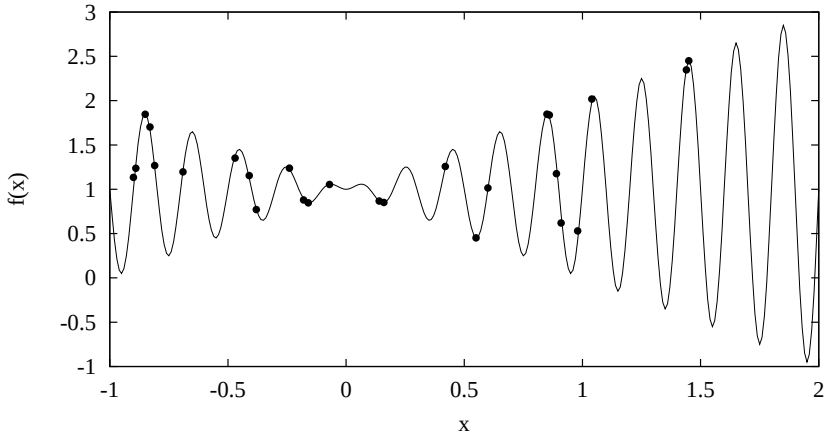


Figura 6: Pontos da primeira geração de cromossomos

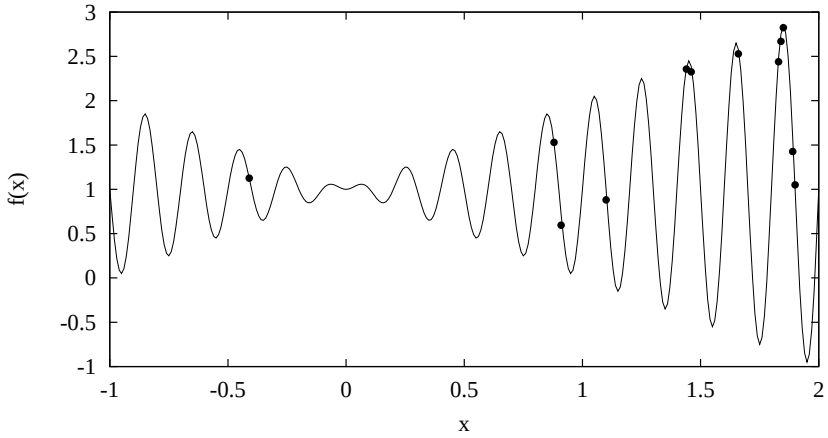


Figura 7: Cromossomos da vigésima quinta geração

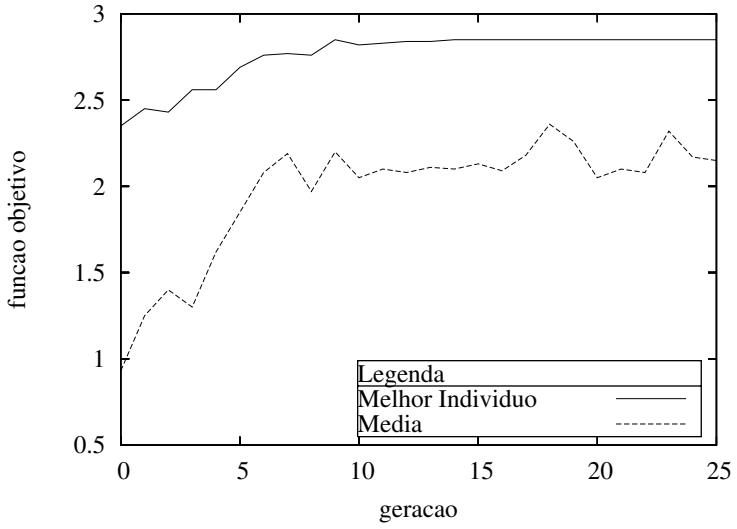


Figura 8: O maior valor e o valor médio da função objetivo em cada geração

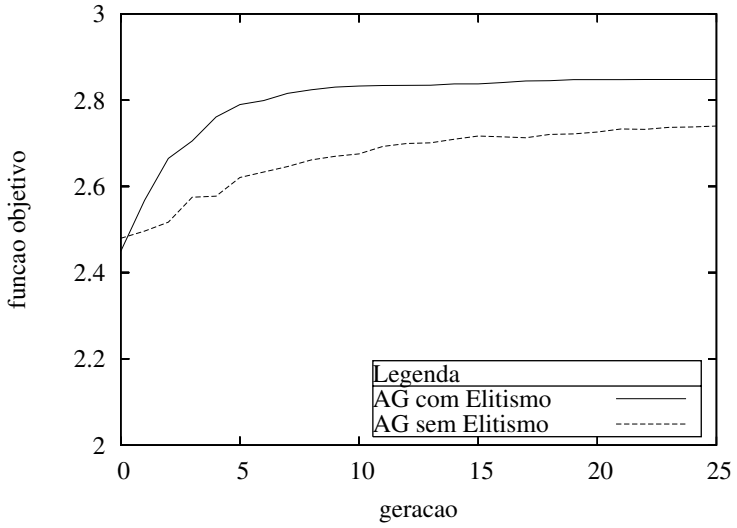


Figura 9: Desempenho do AG com Elitismo

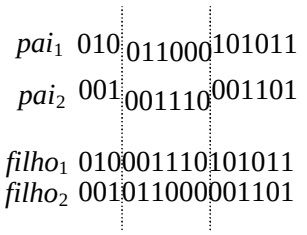


Figura 10: Crossover de 2 pontos

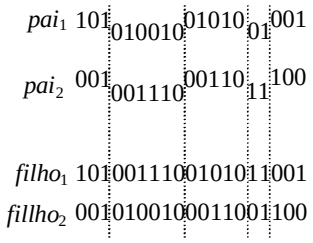


Figura 11: Crossover de 4 pontos

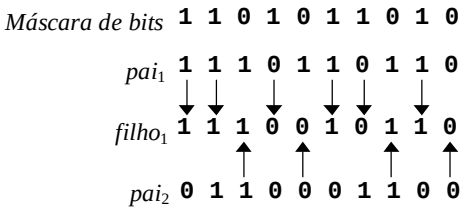


Figura 12: Crossover Uniforme

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 (onde l é o número de bits do cromossomo), uma vez que este sempre leva a metade dos bits de cada pai.

Em [10] é investigada a diferença de desempenho entre vários crossover´s de n pontos e uniforme. A conclusão, conforme relatado em [6], é que não há grandes diferenças de desempenho entre eles. Aliás, de acordo com [17], 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.

1.7 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 diversidade 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. Na Biologia, representa a expressão ou manisfestação física de um gene no organismo (e.g., pele lisa ou rugosa). Nos AGs, representa o objeto ou a estrutura construída 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 influência o valor de outro gene. Problemas com alta Epistasia são de difíceis solução por AGs.

Uma lista completa desta terminologia encontra-se em [19]. A seção seguinte aborda alguns aspectos teóricos de AGs.

	H_1	H_2	H_3
	1****	**10*	*0*01
11001	$\surd$		
11011	$\surd$		
10101	$\surd$	$\surd$	$\surd$

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

Figura 13: Esquemas

1.8 Esquemas

O *Teorema dos Esquemas*, 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 esquema 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. Na Figura 13 é mostrado alguns exemplos.

Conforme mostrado na Figura 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. E os cromossomos 11001, 11011 e 10101 possuem o esquema $1****$. O comprimento de um esquema, representado por $\delta(H)$, 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 contidos no esquema.

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 do esquema H na população intermediária (utilizando seleção proporcional à aptidão) é dado por:

$$m' = \frac{a}{b}m \tag{1.4}$$

Pela Equação 1.4, 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 geração, 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 seguintes, ao passo que esquemas contidos em cromossomos ruins (i.e., aqueles com aptidão abaixo da média) tendem a desaparecer nas gerações seguintes* [16].

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 tal Hipótese são conhecidos como *AG-Deceptivos*. Ocorre isto com blocos que têm forte epistasia. Por exemplo, bloco A era um bom bloco quando parceria com um bloco B no mesmo cromossomo. Quando bloco A foi transferido para outro cromossomo na qual não existia bloco B, bloco A deixou de ser bom. Tal fato confunde a busca do AG. 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 crossover´s. 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 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

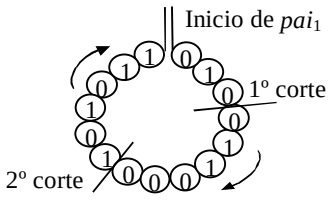


Figura 14: Crossover de 2 pontos visto como um anel

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 [6, 8].

O crossover de n pontos tende a manter juntos os genes que são codificados próximos um do outro. E se existem genes que trabalham bem em parceria é bom que eles estejam juntos no cromossomo para tomar vantagem daquele fato. Existe um operador denominado *Inversão* [21] que busca a seqüência 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.

2. Representação Real

2.1 Representação Binária versus Real

Para ilustrar esta seção, considere a seguinte função a ser maximizada:

$$\begin{aligned} &\text{Maximizar} && f(x_1,x_2) = 0.5 - \frac{\left(\sin\sqrt{x_1^2+x_2^2}\right)^2 - 0.5}{\left(1.0+0.001\left(x_1^2+x_2^2\right)\right)^2} \\ &\text{Sujeita a} && -100 \le x_1 \le 100 \\ &&& -100 \le x_2 \le 100 \end{aligned} \tag{2.1}$$

Esta função é conhecida como F6 na literatura de GA [8]. O máximo global de F6 está no ponto (0.0) cujo valor é $f(0.0) = 1$.

Representação Binária

A maneira usual de aplicar AGs em problemas como Problema (2.1) (otimização numérica) é usar a representação binária. Este problema possui dois parâmetros (genes): x_1 e x_2. Cada gene é codificado em um segmento da cadeia de bits. O comprimento da cadeia depende da precisão numérica requerida. Cada casa decimal requer $3,3$ bits. Se o usuário precisa de oito casas decimais então $8 \times 3,3 = 26,4 \approx 27$ bits são necessários em cada gene. Portanto, o cromossomo requer 27 bits $\times$ 2 genes $= 54$ bits para oito casas decimais. Por exemplo:

$$011010010010011010000010110100011100010000111001011000$$

Para decodificar este cromossomo, primeiramente ele é dividido em dois segmentos de 27 bits:

$$011010010010011010000010110$$
$$100011100010000111001011000$$

Em seguida, estes dois segmentos da cadeia são convertidos da base binária para a base decimal:

$$\begin{aligned} d_1 &= (011010010010011010000010110)_2 = (55129110)_{10} \\ d_2 &= (100011100010000111001011000)_2 = (74518104)_{10} \end{aligned}$$

Finalmente, as duas cadeias são mapeados para o intervalo real do problema usando a Equação (1.2). Assim,

$$x_1 = -100 + (100 - (-100))\frac{55129110}{2^{27} - 1} = -17,851224 \tag{2.2}$$

$$x_2 = -100 + (100 - (-100))\frac{74518104}{2^{27} - 1} = 11,040629 \tag{2.3}$$

Representação Real

Na representação real, o cromossomo é um vetor de números com representação de ponto flutuante na qual cada componente é um parâmetro do problema.

O ponto dado em (2.2) e (2.3) é codificado diretamente no vetor:

$$(-17,851224, 11,040629)$$

na qual é mais amigável ao homem do que uma obscura cadeia de bits. Além disso, a natureza da representação real torna mais fácil a criação de novos operadores genéticos (como será mostrado mais tarde). Além dessas vantagens, a representação

real tem apresentado melhor desempenho do que a binária principalmente em problemas de otimização com parâmetros contínuos [18]. Alguns pesquisadores alegam que a representação binária tende a gerar longos cromosomos[27]. Por exemplo, uma problema com 100 parâmetros requer de uma cadeia de no mínimo 2.700 bits para uma boa precisão numérica. Longos cromossomos torna o espaço de busca grande causando perda de eficiência da busca genética.

A representação binária é historicamente importante. Foi usada pelos trabalhos pioneiros em GA por Holland (1975) e DeJong (1975). É simples para análise teórica. De modo geral, não é possível afirmar qual é a melhor representação por que há argumentos e advogados para ambas as representações (ver [20] para mais discussão).

2.2 Crossovers para Representação Real

Nesta seção será apresentada uma lista (não exaustiva) de operadores para representação real. Considere a seguinte notação utilizada nesta seção. Os cromossomos pais serão representados por:

$$\begin{aligned} \mathbf{p}_1 &= (p_{11}, p_{12}, \ldots, p_{1l}) \\ \mathbf{p}_2 &= (p_{21}, p_{22}, \ldots, p_{2l}) \end{aligned}$$

e o cromossomo filho por

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

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

Considere que o gene c_i do filho $\mathbf{c}$ está dentro do intervalo $[a_i, b_i]$ onde $a_i, b_i \in \Re$. 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. A Notação $r \sim F$ indica que r é um número aleatório extraído da distribuição F (e.g., $r \sim U(x, y)$).

A maioria dos operadores de crossover estão uma desses três categorias [15]:

1. Operadores convencionais.
2. Operadores aritméticos.
3. Operadores baseados na direção.

Operadores Convencionais

Os operadores convencionais são resultados das adaptações dos operadores utilizados para representação binária. Por exemplo:

Crossover de 1-ponto. Dado um ponto de corte na posição k e dois pais:

$$\begin{aligned} \mathbf{p}_1 &= (p_{11},\ p_{12},\ \ldots,\ p_{1k},\ \ldots,\ p_{1l}) \\ \mathbf{p}_2 &= (p_{21},\ p_{22},\ \ldots,\ p_{2k},\ \ldots,\ p_{2l}) \end{aligned}$$

são produzidos dois filhos da seguinte forma:

$$\begin{aligned} \mathbf{c}_1 &= (p_{11},\ p_{12},\ \ldots,\ p_{2k},\ \ldots,\ p_{2l}) \\ \mathbf{c}_2 &= (p_{21},\ p_{22},\ \ldots,\ p_{1k},\ \ldots,\ p_{1l}) \end{aligned}$$

Os operadores convencionais funcionam bem na representação binária, mas na representação real eles basicamente trocam alelos e, portanto, não criam informações novas (i.e., novos números reais), exceto por mutação. Melhor então é usar operadores aritméticos.

Operadores Aritméticos

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

Crossover média [8]. Dado dois cromossomos $\mathbf{p}_1$ e $\mathbf{p}_2$, é produzido um cromossomo $\mathbf{c}$ da seguinte forma:

$$\mathbf{c} = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2}$$

Crossover aritmético [25]. Dado dois cromossomos $\mathbf{p}_1$ e $\mathbf{p}_2$, é produzido dois cromossomos $\mathbf{c}_1$ and $\mathbf{c}_2$ da seguinte forma:

$$\begin{aligned} \mathbf{c}_1 &= r\mathbf{p}_1 + (1-r)\mathbf{p}_2 \\ \mathbf{c}_2 &= (1-r)\mathbf{p}_1 + r\mathbf{p}_2 \end{aligned}$$

onde $r \sim U(0,1)$.

O crossover aritmético é igual ao crossover média se $r = 0,5$. Ele produz aleatoriamente um filho dentro do segmento de reta I ligando os pais $\mathbf{p}_1$ e $\mathbf{p}_2$ (Figure 15). Na ausência de mutação, o crossover aritmético não pode gerar um filho fora dos limites definidos pela população. Portanto, se o ponto de mínimo está fora desses limites, ele não será encontrado pelo crossover aritmético. Além disso, este crossover tende a levar os genes para o meio do seu domínio $[a_i, b_i]$ causando perda de diversidade. Isto pode ser melhorado com o crossover BLX-α que tem capacidade de extrapolação.

Crossover BLX-α ou crossover mistura (do inglês, *blend*) [11]. Dado dois cromossomos $\mathbf{p}_1$ e $\mathbf{p}_2$, é produzido um cromossomo $\mathbf{c}$ da seguinte forma:

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

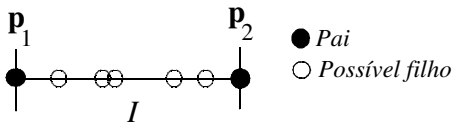


Figura 15: Crossover aritmético

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

Exemplo: considere dois possíveis cromossomos para o Problema (2.1):

$$\begin{aligned} \mathbf{p}_1 &= (30,173; 85,342) \\ \mathbf{p}_2 &= (75,989; 10,162) \end{aligned}$$

Aplicando o BLX com $\alpha = 0.5$ e $\beta = 1,262$ (onde $\beta \sim U(-0,5; 1,5)$), tem-se:

$$\begin{aligned} c_1 &= 30,173 + 1,262(75,989 - 30,173) = 87,993 \\ c_2 &= 85,342 + 1,262(10,162 - 85,342) = -9,535 \end{aligned}$$

assim o filho é dado por:

$$\mathbf{c} = (87,993; -9,535)$$

se o filho $\mathbf{c}$ for infactível, então gerar-se outro filho com novo β. O processo é repetido até obter um filho factível.

Geometricamente, o BLX-α é ilustrado na Figura 16 (no caso de cromosomos unidimensionais) e na Figura 17 (caso multidimensional). No primeiro caso, se $\alpha = 0$, um filho é produzido dentro do segmento de reta I ligando os pais. Mas, se $\alpha > 0$ então o segmento I é estendido αI em cada extremo do segmento e filhos podem ser produzidos fora do segmento. No caso de cromossomos de vetores multidimensionais, BLX-α produz filhos aleatoriamente dentro de um hiperetângulo definido pelos pontos $\mathbf{p}_1$ e $\mathbf{p}_2$. Diferente do crossover aritmético, BLX com $\alpha > 0$ tem capacidade de extrapolar dos limites impostos pela população.

O BLX-$0,5$ balanceia a tendência de gerar filhos próximos ao centro do segmento de reta I porque os filhos tem igual probabilidade para cair tanto dentro como fora do segmento I. Esta tendência ainda pode ser reduzida com a mutação de limite mostrada mais tarde. O BLX-α tem sido usado com sucesso em muitos trabalhos e é talvez o operador mais utilizado para representação real.

Um estudo que vale mencionar é o de Michalewicz [25], 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 simples, crossover aritmético, crossover heurístico, mutação uniforme, mutação de limite, mutação não uniforme e a mutação não-uniforme múltipla. O crossover aritmético foi

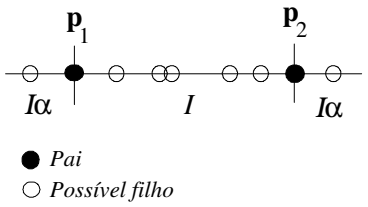


Figura 16: Crossover BLX-α em cromossomos unidimensionais

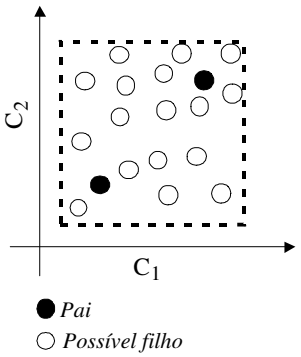


Figura 17: Crossover BLX-α em cromossomos multidimensionais

mencionado acima, os demais serão apresentados nesta e na próxima seção. Outros trabalhos interessantes são [29, 37, 30].

Crossover simples [25]. É similar ao crossover de 1-ponto. Dado dois pais $\mathbf{p}_1$ e $\mathbf{p}_2$ e um ponto de corte k são produzidos dois filhos da seguinte forma:

$$\begin{aligned} \mathbf{c}_1 &= (p_{11},\ p_{12},\ \ldots,\ \alpha p_{2k} + (1-\alpha)p_{1k},\ \ldots,\ \alpha p_{2l} + (1-\alpha)p_{1l}) \\ \mathbf{c}_2 &= (p_{21},\ p_{22},\ \ldots,\ \alpha p_{1k} + (1-\alpha)p_{2k},\ \ldots,\ \alpha p_{1l} + (1-\alpha)p_{2l}) \end{aligned}$$

onde α é inicialmente igual a 1, mas se o filho é infactível então α é reduzido por um fator $1/q$ até torna-lo factível. É idêntico ao crossover de 1-ponto se $\alpha = 1$. Raramente usado, este crossover tem o mesmo problema do crossover de 1-ponto: praticamente não gera novas informações.

Operadores Baseados na Direção

Estes operadores introduzem no crossover a informação da aptidão (ou do gradiente) a fim de determinar a direção da busca. Por exemplo:

Crossover heurístico [37, 25]. Considere que a meta é minimizar a função objetivo $f(\cdot)$. Dado dois pais $\mathbf{p}_1$ e $\mathbf{p}_2$, é produzido um filho $\mathbf{c}$ da seguinte forma:

$$\mathbf{c} = \begin{cases} \mathbf{p}_1 + r(\mathbf{p}_1 - \mathbf{p}_2) & \text{if } f(\mathbf{p}_1) \leq f(\mathbf{p}_2) \\ \mathbf{p}_2 + r(\mathbf{p}_2 - \mathbf{p}_1) & \text{if } f(\mathbf{p}_1) > f(\mathbf{p}_2) \end{cases}$$

onde $r \sim U(0,1)$. Ver Figure 18.

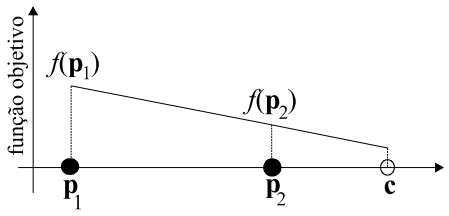


Figura 18: Crossover Heurístico

Se a meta é maximizar $f(\cdot)$ basta inverter as relações $\leq$ e $>$ acima. 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 para sem produzir filhos.

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 [14] ou operadores como o crossover quadrático [2] que usa três pais para fazer um ajuste de curvas quadrático com base no valor da aptidão.

2.3 Mutação para Representação Real

Mutação uniforme [25]: é a simples substituição de um gene por um número aleatório. Ou seja, dado um cromossomo **p** com o j-ésimo gene selecionado para mutação, é produzido um cromossomo **c** da seguinte forma

$$c_i = \begin{cases} r & \text{se } i = j \\ p_i & \text{caso contrário} \end{cases}$$

onde $r \sim U(a_i, b_i)$ na qual a_i e b_i representam os limites direito e esquerdo do domínio do gene c_i.

Mutação gaussiana: é a substituição de um gene por um número aleatório de uma distribuição normal. Ou seja, dado um cromossomo **p** com o j-ésimo gene selecionado para mutação, é produzido um cromossomo **c** da seguinte forma:

$$c_i = \begin{cases} p_i + r & \text{se } i = j \\ p_i & \text{caso contrário} \end{cases}$$

onde $r \sim N(0, \sigma)$. $N(0, \sigma)$ é uma distribuição normal com média zero e desvio padrão σ. Alternativamente, pode-se diminuir o valor de σ, à medida que aumenta a número de gerações criando um efeito que lembra o algoritmo Recozimento Simulado [26].

Mutação creep: adiciona ao gene um pequeno número aleatório de distribuição normal (ou de outras distribuições). É idêntico a mutação gaussiana, exceto que o numero gerado aleatoriamente é pequeno. O número aleatório deve ser pequeno o suficiente para que cause apenas pequena perturbação no cromossomo, porque estando perto do ponto mínimo, tal perturbação gera filhos na vizinhança do mínimo e dai pode mover o AG rapidamente ao mínimo. 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 de limite [25]: 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 do domínio $[a_i, b_i]$ do gene c_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 seus respectivos domínios.

Mutação não-uniforme [25]: 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:

$$c_i = \begin{cases} p_i + \Delta(t, b_i - p_i) & \text{se } i = j \text{ e um dígito binário aleatório é 0} \\ p_i - \Delta(t, p_i - a_i) & \text{se } i = j \text{ e um dígito binário aleatório é 1} \\ p_i & \text{caso contrário} \end{cases}$$

onde t é o número da geração corrente. A função $\Delta(t, y)$ tem a seguinte propriedade: retorna um valor no intervalo $[0, y]$ tal que a probabilidade de $\Delta(t, y)$ retornar um valor próximo a zero aumenta a medida que t também aumenta. Inicialmente (quando t é pequeno) este operador busca a espaço uniformemente. Nas últimas gerações (quando t é grande) ele busca o espaço localmente. Em [25], a seguinte função foi usada:

$$\Delta(t, y) = yr\left(1 - \frac{t}{t_{\max}}\right)^b,$$

onde $r \sim U(0,1)$, $t_{\max}$ é o número máximo de gerações e b é um parâmetro que determina o grau de não-uniformidade. (em [25] é usado $b = 6$ como valor *default*).

Mutação não-uniforme múltipla [25]: é a simples aplicação do operador mutação não-uniforme em todos os genes do cromossomo $\mathbf{p}$.

2.4 Usando Vários Operadores

Uma solução para trabalhar com vários operadores é pondera-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 [8].

3. Algoritmos Genéticos e Otimização Convencional

3.1 Introdução

Em geral, um problema otimização consiste em achar a solução que corresponda ao ponto de máximo ou mínimo de uma função $f(x, y, z, u, \ldots)$ de n parâmetros, $x, y, z, u, \ldots$ 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).

A forma comum de apresentar um problema de otimização numérica é a seguinte:

$$\begin{array}{rl} \text{Minimizar} & f(\mathbf{x}) \\ \text{Sujeita a} & \mathbf{x} \in S \end{array}$$

onde $f : D \rightarrow \Re$ é denominada de função objetivo com domínio D que também é chamado de *espaço de busca*. S é denominado de *conjunto factível* e é um subconjunto de D. Um ponto $\mathbf{x}$ fora do conjunto S é denominado de solução *infactível*. Portanto, o espaço de busca é dividido em regiões factível e infactível (ver Figura 19). Um ponto factível $\mathbf{x}^*$ (i.e., $\mathbf{x}^* \in S$) é dito ser um *mínimo local* se e somente se $f(\mathbf{x}^*) \leq f(\mathbf{x})$ para todo $\mathbf{x}$ *suficientemente* próximo de $\mathbf{x}^*$. Um ponto $\mathbf{x}^*$ é dito ser um *mínimo global* se e somente se $f(\mathbf{x}^*) \leq f(\mathbf{x})$ para todo ponto factível $\mathbf{x}$. Uma função que apresenta apenas um ponto de mínimo é denominada de *função unimodal*. Quando uma função possui mais de um ponto de mínimo, 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.

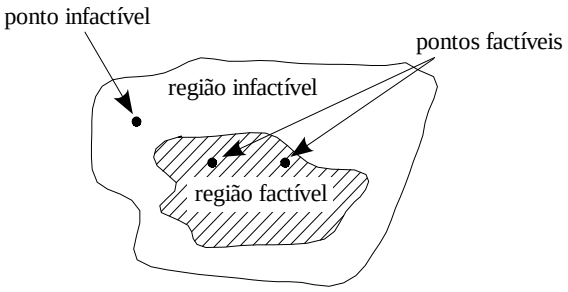


Figura 19: Espaço de Busca

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

$$\begin{array}{rl} \text{Minimizar} & f(x,y) \\ \text{Sujeita a} & x^2+y^2=5 \end{array}$$

ou na forma de desiqualdades:

$$\begin{array}{rl} \text{Minimizar} & f(x,y) \\ \text{Sujeita a} & x^2+2y^2 \leq 5 \end{array}$$

Quando os parâmetros são irrestritos, eles podem assumir qualquer valor do espaço de busca. Por exemplo:

$$\begin{array}{rl} \text{Minimizar} & f(\mathbf{x}) \\ \text{Sujeita a} & \mathbf{x} \in D \end{array}$$

Vários métodos de otimização trabalham melhor com parâmetros irrestritos. Em alguns 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 $x = Min + (Max - Min)sen^2(u)$ 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 um problema de *Programação Linear*. Quando a função objetivo ou as restrições são funções não lineares dos parâmetros, o problema é conhecido como um 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*.

3.2 Métodos de Otimização

Visando situar os Algoritmos Genéticos no contexto de otimização em geral, considerem-se algumas 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 encontradas.

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 (do inglês, *hill climbing*) investigam os pontos adjacentes do espaço de busca e movem-se na direção que melhora o valor da função. Em geral, estes métodos começam em um ponto arbitrário do espaço de busca e usam a informação do gradiente (derivadas) para encontrar a direção de subida. Mas não garantem encontrar o máximo global. Por exemplo, na Figura 20, o método poderia começar aleatoriamente no ponto A ou C. Se começa no C, o método terá êxito, pois subirá até o ponto D que é o ponto de máximo global. Se, por acaso, iniciar no ponto A então o método falhará pois subirá até o B que é um máximo local. Contudo, métodos de Subida de Encosta são geralmente bem mais rápidos do que outros métodos de otimização. Grande número de técnicas importantes de otimização são métodos de Subida de Encosta. O estado da arte desses métodos incluem a família dos métodos do gradiente conjugado (e.g, o algoritmo de Fletcher-Reeves), a família dos métodos quase-newton (e.g, o algoritmo BFGS) e o Algoritmo Simplex de Nelder e Mead que, apesar de ser um pouco mais lento, é interessante porque não necessita de derivadas. Vários livros texto descrevem estes métodos como por exemplo [31].

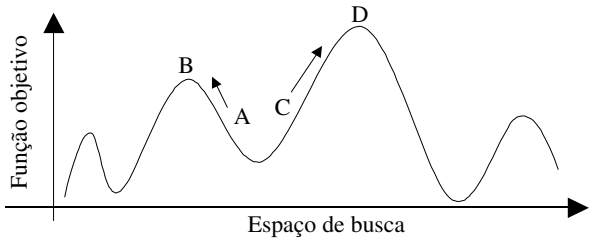


Figura 20: Subida de Encosta

3.3 Algoritmos Genéticos

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 multimodais complexas e complicadas, reduzindo a incidência em 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 problemas simples que podem ser resolvidos por métodos da Subida de Encosta. Não raro, os AGs ainda estão avaliando a população inicial enquanto os 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 relacionados alguns aspectos do mecanismo de busca dos AG´s.

Seleção e Deriva Genética

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. Portando, a freqüência de genes na população tende a variar devido a seleção que favorece uns genes e descarta outros.

Um outro fator, além da mutação e seleção, pode afetar a freqüência de genes na população: a *deriva genética* (do inglês, *genetic drift*). A deriva genética é o fenômeno de variação da freqüência de genes devido puramente ao acaso. Afeta principalmente as populações pequenas e 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*. Com taxas adequadas de mutação, é possível recuperar a freqüência dos genes perdidos mantendo uma boa de diversidade de genes da população.

3.4 Exploitation e Exploration

Um algoritmo de otimização eficiente deve usar estas duas técnicas (denominadas de *exploration* e *exploitation*, respectivamente) para encontrar o ótimo global da função objetivo [5]. Ambas as técnicas têm a mesma tradução para o português: exploração. Entretanto,

- *Exploration* diz respeito à exploração no sentido de visitar pontos desconhecidos no espaço de busca à procura por novas soluções. Esta é a técnica usada pelo método Gerar-e-Testar.

- *Exploitation* significa exploração no sentido de extrair informações (como por exemplo, o gradiente) presentes nas soluções já encontradas e usa-las para obter as próximas soluções. Esta é a técnica usada pelo método Subida da Encosta.

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, baseado na aptidão, em direção aos melhores pontos encontrados até então (*exploitation*).

Um fator que influencia a quantidade de *exploration* e *exploitation* é a *pressão seletiva*. Este termo é largamente usado para caracterizar a forte (alta pressão seletiva) ou a fraca (baixa pressão seletiva) ênfase na seleção dos melhores indivíduos da população.

Quando a pressão seletiva é muito alta, o AG assume o comportamento dos métodos de Subida de Encosta (muito *exploitation*). Os primeiros bons indivíduos (sub-ótimos) gerados pela AG ganham uma aptidão muita alta e tendem a dominar a população espalhando seus genes através das gerações com alta probabilidade. Conseqüentemente, o AG converge rapidamente, possivelmente subindo a encosta do primeiro pico local que encontra. Isto acontece porque o AG não teve tempo para explorar regiões desconhecidas do espaço do busca (*exploration*).

Por outro lado, quando a pressão seletiva é muito baixa, o AG assume o comportamento dos métodos Gerar-e-Testar (muito *exploration*) porque a aptidão é aproximadamente igual para todos os indivíduos. Aqui o AG converge lentamente explorando aleatoriamente o imenso espaço de busca. Neste sentido, o AG torna-se muito similar ao bem conhecido processo estocástico do passo ao acaso (do inglês, *random walk*).

A pressão seletiva pode ser regulada arbitrariamente pelo usuário. Aumentar a pressão seletiva, também aumenta a velocidade de convergência do AG. Contudo, isto pode levar o AG a encontrar máximos ou mínimos locais. Na prática, é difícil conhecer qual a pressão seletiva que fornece o equilíbrio ideal entre *exploitation* e *exploration*.

Além dos Algoritmos Genéticos, existem outros métodos de otimização global (i.e., métodos que levam ao ótimo global) que combinam *exploration* e *exploitation*, sendo a maioria deles recentes. Os métodos antigos de otimização são quase todos métodos de Subida de Encosta. Alguns exemplos de métodos de otimização global são Recozimento Simulado [26, 34], Busca Tabu [26, 34] e *Branch and Bound* [32].

4. Aspectos Avançados

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

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, ou seja, 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 2.

Tabela 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 denomina *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.

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 [18] sugerem, para lidar com tais funções objetivo, algumas dicas para evitar avaliar cromossomos idênticos mais de uma vez, reutilizando deste modo a avaliação feita anteriormente.

Isto pode 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 (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, dever-se-ão 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).

4.3 Substituições Geracional e de Estado Uniforme

Os AGs trabalham substituindo as populações antigas por novas. Alguns tipos de substituição são:

- Substituições geracional;
- Substituições geracional com elitismo;
- Substituição de estado uniforme (do inglês, *steady-state*);
- Substituição de estado uniforme sem duplicatas.

Na substituição geracional (utilizada na seção 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 a pressão seletiva.

Na substituição de estado uniforme, 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 de estado uniforme, são criados n filhos que substituem os n piores pais. Em geral,

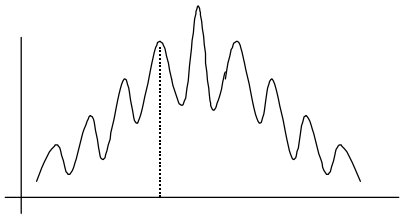


Figura 21: Convergência prematura

a taxa de crossover é maior (≈ 1) no AG de estado uniforme do que no AG geracional. Esta abordagem que utiliza os filhos para crossover tão logo eles sejam gerados, tem apresentado bons resultados [8]. 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 do mundo real, este custo extra não é significativo, pois o custo computacional está quase todo concentrado no cálculo da aptidão.

A substituição de estado uniforme sem duplicatas é uma abordagem alternativa que não substitui o indivíduo da população caso uma duplicata do filho já exista na população. A idéia é evitar o grande número de filhos duplicatas gerados pelo AG de estado uniforme [8].

4.4 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. Como conseqüência, o algoritmo converge para um máximo ou mínimo local, Figura 21.

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.

Um outro problema que pode ocorrer nos AGs é a supercompressão do intervalo da aptidão explicado pelo seguinte exemplo. Considere substituir a função objetivo

utilizada no Problema (2.1) pela seguinte nova função objetivo:

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

Tal substituição torna o valor da função objetivo praticamente o mesmo para toda a população, como na Tabela 3.

Tabela 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 praticamente zera a pressão seletiva (pois qualquer cromossomo tem a mesma probabilidade de seleção). Ocorre problema semelhante nas gerações finais do algoritmo, mas por outro motivo. Grande parte da população convergiu e os cromossomos têm aptidões praticamente iguais. A pressão seletiva é quase eliminada. 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 da função objetivo.

4.5 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 (o algoritmo da roleta não funciona), o intervalo da aptidão torna-se supercomprimido (elimina a pressão seletiva), 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 [4, 35]:

$$f_i = \min + (\max - \min)\frac{N-i}{N-1}$$

em que i é o índice do cromossomo na população em ordem decrescente de valor da função objetivo. Normalmente é utilizado $1 \leq \max \leq 2$ e $\max + \min = 2$. Vale notar que deste modo a aptidão representa o número de filhos esperados do cromossomo e max serve de parâmetro para controlar a pressão seletiva. Um exemplo é mostrado na Tabela 4, que expande o intervalo dos valores da função objetivo que estão muitos próximos (resolvendo assim o problema mostrado na Tabela 3).

Tabela 4: Ordenamento Linear

Cromossomo	Função objetivo	Posição	Aptidão	Probabilidade de seleção
A	2.000,999588	1	2,0	40%
B	2.000,826877	2	1,5	30%
C	2.000,655533	3	1,0	20%
D	2.000,400148	4	0,5	10%
E	2.000,102002	5	0,0	0%

A Figura 22 mostra o controle da pressão seletiva utilizando o ordenamento linear. Na Figura 22(a), a alta pressão seletiva favorece fortemente os melhores cromossomos, direcionando a busca às melhores soluções encontradas até então (muito *exploitation*). Na Figura 22(b), a baixa pressão seletiva favorece um pouco mais os cromossomos de baixa aptidão, direcionando a busca para regiões desconhecidas do espaço de busca (muito *exploration*).

No método ordenamento exponencial, a aptidão é dada por [25]:

$$f_i = q(1-q)^{i-1} \tag{4.1}$$

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 Equação 4.1 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:

$$f = ag + b$$

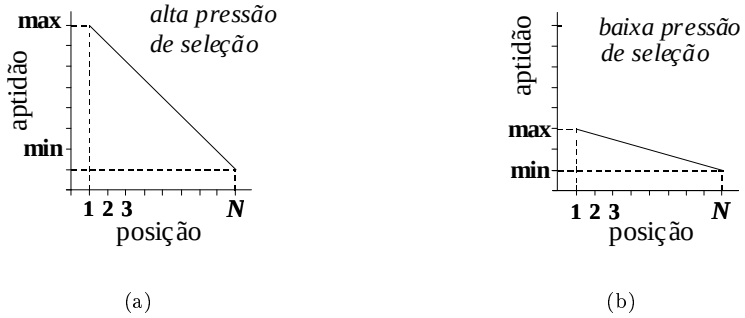


Figura 22: Posição do cromossomo versus pressão da seleção

em que g é o valor da função objetivo (ver Figura 24). 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., $f_{\max} = C\bar{f}$). Tipicamente, o valor de C está entre 1,2 e 2,0. O aumento de C provoca uma aumento na pressão seletiva. 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 Figura 23 (o teste $g_{\min} > (C\bar{g} - g_{\max})/(C-1)$, verifica se ocorre aptidão negativa).

4.6 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

$\bar{g} \leftarrow \frac{1}{N}\sum_{i=1}^{N} g_i$
SE $g_{\min} > (C\bar{g} - g_{\max})/(C-1)$ **ENTÃO**
    $\Delta \leftarrow g_{\max} - \bar{g}$
    $a \leftarrow (C-1)\bar{g}/\Delta$
    $b \leftarrow \bar{g}(g_{\max} - C\bar{g})/\Delta$
SENÃO
    $\Delta \leftarrow \bar{g} - g_{\min}$
    $a \leftarrow \bar{g}/\Delta$
    $b \leftarrow -\bar{g}g_{\min}/\Delta$
FIM SE
RETORNE a e b

Figura 23: Cálculo dos coeficientes a e b do escalonamento linear

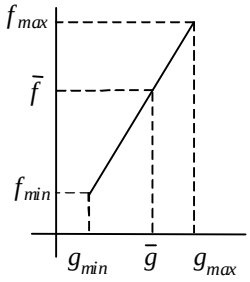


Figura 24: Gráfico de escalonamento da aptidão

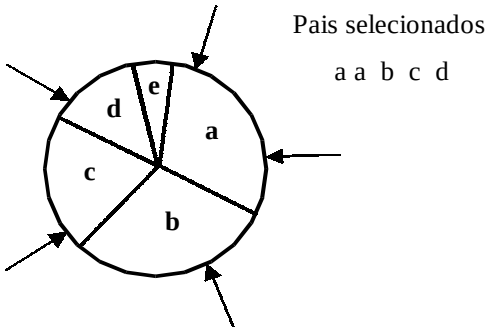


Figura 25: Amostragem Universal Estocástica

com aptidão acima da média terão mais chances de serem selecionados.

4.7 Amostragem Estocástica Universal

O algoritmo 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*) [4] 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 25).

5. O Problema do Caixeiro Viajante

O objetivo deste capítulo 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 conhecido problema de otimização combinatória: o problema do caixeiro viajante (PCV).

Este problema é NP-Completo [24], 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-Completo 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. É um fenômeno chamado explosão combinatória.

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:

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

onde (x_i, y_i) é a coordenada de uma cidade e (x_{i+1}, y_{i+1}) representa a próxima cidade a ser visitada.

Uma maneira de resolver o PCV via AGs é representar cada cromossomo por uma lista de cidades. Por exemplo, considere a lista de cidades representadas pelas letras $A, B, \ldots, G$ na Figura 26, como um possível caminho a ser seguido pelo caixeiro. Tal caminho seria representado por um cromossomo como:

Cromossomo: $A \quad B \quad C \quad D \quad F \quad E \quad G$

Note 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.

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

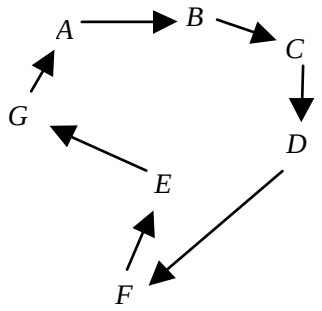


Figura 26: Uma possível rota de viagem para o vendedor do PCV

permutações dos elementos A, B e C. 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 [33] e o Problema Coloração de Grafos [8]. Existem vários operadores genéticos que realizam permutações [16, 33], sendo alguns deles descritos a seguir.

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

$$\text{Cromossomo} \quad : \quad A \quad B \quad C \quad D \quad F \quad E \quad G$$

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:

$$\begin{array}{rcccc|ccc|cc} \text{Cromossomo} & : & A & B & C & D & F & E & G \\ \text{Após a mutação} & : & A & B & F & C & D & E & G \end{array}$$

Existem vários operadores de crossover para permutação. Alguns deles [16, 33] 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 pai_1, nas posições selecionadas, o mesmo ordenamento que estes mesmos elementos apresentam em pai_2. Em seguida, o novo ordenamento nas posições selecionadas de pai_1 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. $filho_2$ 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:

pai_1	:	A	B	$\vert$ C	D	F	$\vert$ E	G
pai_2	:	C	E	$\vert$ G	A	D	$\vert$ F	B

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.

$$\begin{array}{lcc|ccc|cc} pai_1 & : & A & B & G & D & F & E & C \\ pai_2 & : & G & E & C & A & D & F & B \end{array}$$

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.

$$\begin{array}{lcc|ccc|cc} pai_1 & : & D & B & G & A & F & E & C \\ pai_2 & : & G & E & C & D & A & F & B \end{array}$$

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:

$$\begin{array}{lcc|ccc|cc} filho_1 & : & D & B & G & F & A & E & C \\ filho_2 & : & G & E & C & D & F & A & B \end{array}$$

Crossover CX

No crossover CX, o filho copia o seu i-ésimo elemento do i-ésimo elemento de um dos dois pais pelo seguinte procedimento. Inicia-se copiando o primeiro elemento de pai_1 para $filho_1$ (alternativamente, pode-se começar copiando um elemento qualquer da lista). Conseqüentemente, C é copiado para $filho_2$ para completar a troca.

$$\begin{array}{lcccccccc} pai_1 & : & A & B & C & D & F & E & G \\ pai_2 & : & C & E & G & B & D & F & A \end{array}$$

$$\begin{array}{lcccccccc} filho_1 & : & A & \square & \square & \square & \square & \square & \square \\ filho_2 & : & C & \square & \square & \square & \square & \square & \square \end{array}$$

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

$$\begin{array}{lcccccccc} filho_1 & : & A & \square & C & \square & \square & \square & \square \\ filho_2 & : & C & \square & G & \square & \square & \square & \square \end{array}$$

Do mesmo modo, G de pai_1 é copiado para $filho_1$ e A é copiado para $filho_2$.

$$\begin{array}{lcccccccc} filho_1 & : & A & \square & C & \square & \square & \square & G \\ filho_2 & : & C & \square & G & \square & \square & \square & A \end{array}$$

Seguindo o mesmo processo, A de pai_1 deve ser copiado para pai_1. Porém, como A já foi copiado para pai_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:

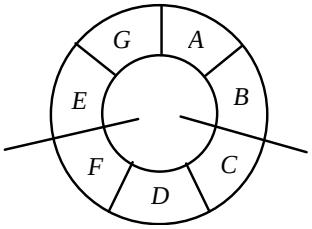
$$\begin{array}{lcccccccc} filho_1 & : & A & E & C & B & D & F & G \\ filho_2 & : & C & B & G & D & F & E & A \end{array}$$

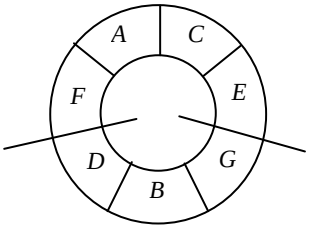
Crossover OX

O crossover OX inicia com dois cortes escolhidos aleatoriamente.

$$\begin{array}{lcccc|ccc|cc} pai_1 & : & A & B & | & C & D & F & | & E & G \\ pai_2 & : & C & E & | & G & B & D & | & F & A \end{array}$$

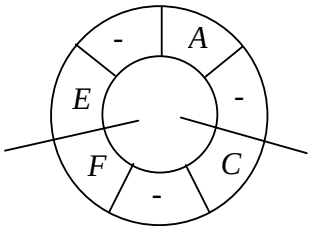
Seu funcionamento é melhor ilustrado interpretando o cromossomo como um círculo, onde o primeiro e o último elementos se juntam.





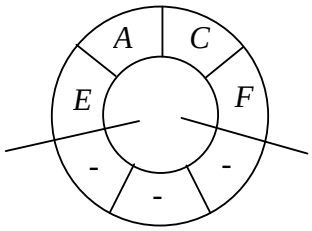
$$\begin{array}{cccc|ccc|cc} A & B & | & C & D & F & | & E & G \\ \multicolumn{4}{c}{(pai_1)} & & & & & \end{array} \qquad \begin{array}{cccc|ccc|cc} C & E & | & G & B & D & | & F & A \\ & & & & \multicolumn{3}{c}{(pai_2)} & & \end{array}$$

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



$$\begin{array}{lcccc|ccc|cc} pai_1 & : & A & \square & | & C & \square & F & | & E & \square \end{array}$$

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:



$$pai_1 \quad : \quad C \quad F \mid \square \quad \square \quad \square \mid E \quad A$$

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

$$filho_1 \quad : \quad C \quad F \mid G \quad B \quad D \mid E \quad A$$

Por processo similar é obtido $filho_2$:

$$filho_2 \quad : \quad G \quad B \mid C \quad D \quad F \mid A \quad E$$

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.

6. Fontes Adicionais de Consulta

Há vários livros de AG na literatura, desde de clássicos como Goldberg [16], passando por bons livros texto básicos tais como Michalewicz [25], Mitchell [28], Haupt e Haupt [18], até livros avançados, como Bäck et al. [3]. Há também muitos tutoriais, como por exemplo Beasley et al. [5, 6], Whitley [36] e Janikow e Clair [23]. Na web um dos sites mais conhecidos é o ENCORE [1].

Referências

[1] Encore: The Evolutionary COmputation REpository network. (http://alife.santafe.edu/ joke/encore/)., 1998.

[2] A. A. Adewuya. New methods in genetic search with real-valued chromosomes. Master's thesis, M.I.T., 1996.

[3] T. Back, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[4] J. Baker. Reducing bias and inefficiency in the selection algorithm. In J. Grefenstette, editor, *Proc. of the Second International Conference on Genetic Algorithms and Their Applications*, pages 14–21, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

[5] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993. Avaliable by ftp on ENCORE in file: GA/papers/over92.ps.gz.

[6] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algortihms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993. Avaliable by ftp on ENCORE in file: GA/papers/over93-3.ps.gz.

[7] C. A. C. Coello. A survey of constraint handling techniques used with evolutionary algorithms. url = citeseer.nj.nec.com/202945.html.

[8] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[9] K. DeJong. *The analysis and behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[10] L. J. Eshelman, R. A. Caruna, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 10–19, San Mateo, CA, 1989. Morgan Kaufmann.

[11] L. J. Eshelman and D. J. Shaffer. Real-coded genetic algorithms and interval-schemata. In D. L. Whitley, editor, *Foundations of Genetic Algorithms 3*, pages 187–203. San Mateo, CA: Morgan Kaufman, 1992.

[12] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995. (http://www.lania.mx/ ccoello/EMOO/EMOObib.html).

[13] C. M. Fonseca and P. J. Fleming. Multiobjective optimization. In T. Back, D.B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C4.5:1–9. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[14] M. Gen and R. Cheng. *Genetic Algorithms and Engineering Design*. Wiley Series in Engineering Design and Automation. John Wiley & Sons, 1997.

[15] M. Gen and R. Cheng. *Genetic algorithms and engineering optimization*. Wiley, 2000.

[16] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[17] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE trans SMC*, 16:122–128, 1986.

[18] R. L. Haupt and S. E. Haupt. *Pratical Genetic Algorithms*. Wiley-Intercience, 1998.

[19] J. Heitkoetter and D. Beasley. The Hitch-Hiker´s guide to evolutionary computation: a list of frequently asked questions., 1998. (Disponível por ftp no ENCORE).

[20] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[21] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[22] J. Horn and N. Nafpliotis. Multiobjective optimization using the niched pareto genetic algorithm. Technical Report IlliGAL 93005, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1992. (http://gal4.ge.uiuc.edu/illigal.home.html).

[23] C. Z. Janikow and D. S. Clair. Genetic algorithms simulating nature´s methods of evolving the best design solution. *IEEE Potentials*, 14:31–35, 1995.

[24] H. R. Lewis and C. H. Papadimitriou. *Elementos de Teoria da Computação*. Bookman, 2000.

[25] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

[26] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 1999.

[27] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[28] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[29] H. Muhlenbein and D. SchlierkampVoosen. Predictive models for the breeder genetic algorithm: Continuous parameter optimization. *Evolutionary Computation*, 1, 1993.

[30] I. Oho and S. Kobayashi. A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. In T Back, editor, *Proc. of the 7th ICGA*, pages 246–253, 1997.

[31] W. H. Press, S. A. Teukolsky B. P. Flannery, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[32] J. Stolfi and L. H. Figueiredo. Métodos numéricos auto-validados e aplicações. In 21º *Cóloquio Brasileiro de Matemática*. IMPA, 1997.

[33] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 332–349. Van Nostrand Reinhold, 1991.

[34] V. Viana. *Meta-Heurísticas e Programação Paralela em Otimização Combinatória*. UFC Edições, 1998.

[35] D. Whitley. The genitor algorithm and selection pressure: Why rankbased allocation of reproductive trials is best. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, Calif., 1989. Morgan Kaufmann.

[36] D. L. Whitley. A genetic algorithm tutorial. Technical Report CS93103, Colorado State University, 1992. Avaliable by ftp on ENCORE in file: GA/papers/tutor92.ps.

[37] A. Wright. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218, 1991.