

# Algoritmos Genéticos (GAs)

João Batista Mendes

Montes Claros, MG

## Formalização matemática

Dado um problema de otimização descrito por:

Minimize  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

Sujeito a:

$$g_i(x) \geq 0 \quad \forall i \in \{1, \dots, q\}, \quad x \in X.$$

$$g_i(x) = 0 \quad \forall i \in \{q + 1, \dots, m\}, \quad x \in X.$$

$$x_j^{min} \leq x_j \leq x_j^{max} \quad j \in \{1, \dots, K\}$$

Onde:

$\nabla f(x)$ : Função objetivo;

$\nabla X$ : Espaço de busca;

$\nabla g_i, \forall i \in \{1, \dots, q\}$ : Conjunto de restrições do problema  $\rightarrow$  *Soluções viáveis (ou factíveis) x soluções não viáveis.*

**Introdução** Os algoritmos genéticos (GAs) são um dos tipos de EAs mais conhecidos.

Proposto por John Holland (década de 60).

Década de 70: Estudado/implementado por Holland e seus alunos (destaque para Goldberg).

1975 - Holland: *Adaptation in Natural and Artificial Systems*.

Descreve o GA como uma abstração da evolução biológica.

Elementos principais:

- População de "cromossomos- string de '0's e '1's. O número de indivíduos da população corresponde ao tamanho da população;

- Operador de Seleção baseado no fitness (aptidão);
- Operador de Cruzamento para geração de novas soluções (filhos);
- Operadores de Mutação e Inversão.

Observações:

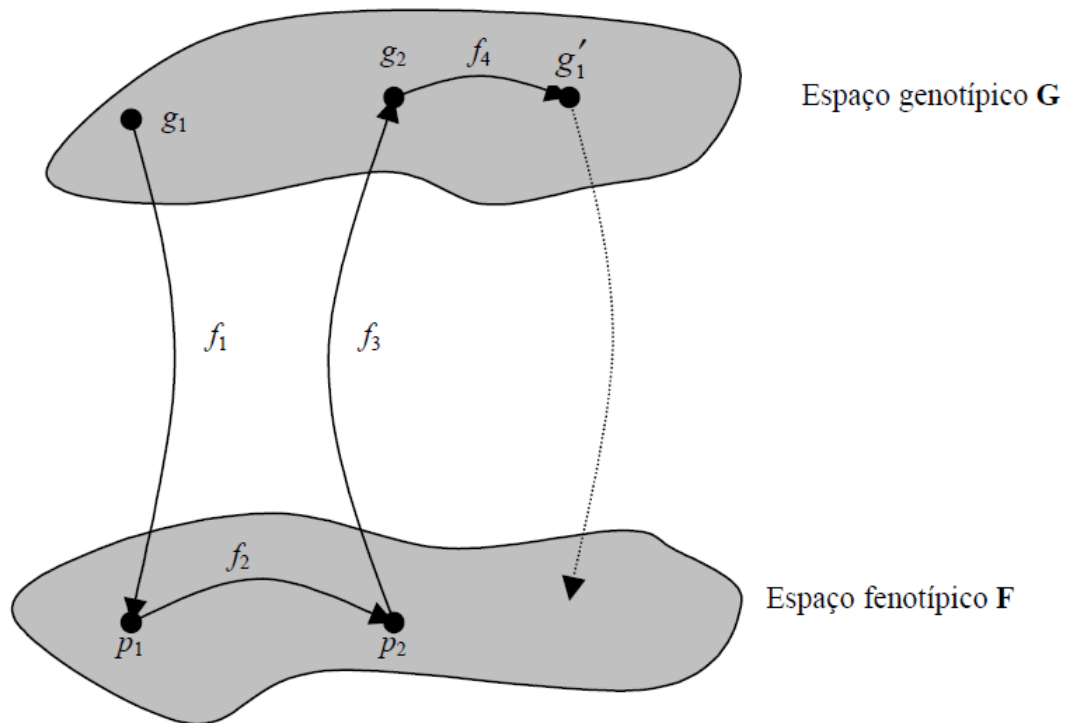
- Cada cromossomo compreende um conjunto de genes (bits) e cada gene é uma instância de um alelo (0 ou 1) e representa uma solução candidata no espaço de busca de soluções candidatas;
- O operador de *seleção* identifica os cromossomos da população para reprodução. Geralmente, o cromossomo com melhor aptidão produz mais filho que o de pior aptidão;

- O operador de *cruzamento* troca conteúdo (partes, pedaços) de dois cromossomos;
- A *mutação* modifica o estado de alguns alelos de um cromossomo;
- A reversão inverte o ordenamento de uma sequência de um cromossomo.

## **Definições:**

*Genótipo*: Codificação do conjunto de soluções candidatas. Podem existir diversas representações para um mesmo problema. **G**: Espaço de estados genotípicos (codificação);

*Fenótipo*: Interpretação do genótipo. **F**: Espaço fenotípico (comportamental).



Mapeamentos:

- ◇  $f_1$ : Mapeia elementos de **G** em **F**. Mapeamento do tipo muitos para um;
- ◇  $f_2$ : Descreve os processos de seleção e migração de indivíduos dentro da população local;

- ◇  $f_3$ : Descreve os efeitos da operação de seleção e migração no espaço  $\mathbf{G}$ ;
- ◇  $f_4$ : Descreve as regras de mutação e recombinação de  $\mathbf{G}$  em  $\mathbf{G}$ .

## Algoritmo Genético Simples

---

**Algorithm 1** GA\_Simples ()

---

```
 $t = 0;$   
Inicializa ( $P(t)$ ,  $M$ ,  $I$ );  
Avalia-se ( $P(t)$ );  
while Critério de parada não atendido do  
     $t \leftarrow t + 1;$   
     $S(t) \leftarrow \text{Selecao } (P(t-1));$   
     $Q(t) \leftarrow \text{Cruzamento } (S(t));$   
     $Q(t)' \leftarrow \text{Mutacao } (Q(t));$   
     $Q(t)' \leftarrow \text{Avalia-se } (Q(t));$   
     $P(t) \leftarrow \text{Substituicao } ( P(t-1) \cup Q(t)' );$   
end while  
Return  $P(t);$ 
```

---

Numa primeira etapa, gera-se (normalmente de forma aleatória) uma população de  $M$  indivíduos (soluções) -  **$P(0)$**  e avalia-se (determina-se o fitness) cada indivíduo da população.

A etapa seguinte constitui-se num processo re-



petivo e corresponde a uma geração do algoritmo.

Em cada geração, temos:

† Alguns indivíduos da população são selecionados para cruzamento -  $\mathbf{S}(t)$  - segundo o seu fitness (melhor fitness, maior probabilidade de cruzamento).

★ Em seguida, aplica-se o operador de cruzamento, segundo uma probabilidade  $P_c$ , nas soluções de  $\mathbf{S}(t)$ , resultando na população  $\mathbf{Q}(t)$ .

\* O operador de mutação, segundo uma probabilidade  $P_m$ , é aplicado nas soluções de  $\mathbf{Q}(t)$  resultando no conjunto de soluções  $\mathbf{Q}(t)'$  - *população filha* - que é avaliada.

□ População da próxima geração -  $\mathbf{P}(t) \rightarrow M$  soluções são retiradas do conjunto  $\mathbf{P}(t-1) \cup \mathbf{Q}(t)'$ .

O processo se repete por diversas gerações, até que uma condição de parada seja atendida. Normalmente, a cada geração, soluções relativamente “boas” se reproduzem, enquanto que soluções relativamente “ruins” são eliminadas.

## **Representação (Codificação) do conjunto de soluções**

Normalmente, uma solução é representada por uma lista ou vetor de atributos de tamanho fixo.

$Sol_1$	$Sol_2$	$\dots$	$S_{n-1}$	$S_n$
---------	---------	---------	-----------	-------

Representações possíveis para um indivíduo (solução):

1). *Binária*: Sequência (string) de bits: Representação mais usada (adotada pelo GA padrão) diante da facilidade de implementação e busca por similaridades.

Motivação: *HOLLAND (1992)* argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao GA, e prova que um alfabeto binário maximiza o paralelismo implícito.

Cada parte do cromossomo é chamada de *gene*. Cada gene possui um valor (chamado de *alelo*) e sua localização (chamado de *locus*).

Para a representação binária de um número real com  $l$  genes, o número real  $x$  é determinado por:

$$x = \frac{\sum_{i=0}^{l-1} a_i 2^i}{2^l} \times (\bar{x} - \underline{x}) + \underline{x}$$

◇  $a_i$ : Alelo do locus  $i$ ;

◇  $\bar{x}$  e  $\underline{x}$  correspondem, respectivamente, os limites superior e inferior para o número real.

## Exemplo

Data representação binária (genótipo) seguinte com  $l = 12$ ,  $\bar{x} = 2$  e  $\underline{x} = -1$ .

Locus	11	10	9	8	7	6	5	4	3	2	1	0
Alelo	1	0	1	0	0	1	1	0	1	0	1	1

Ela corresponde a que número real (fenótipo)

???

A decodificação da representação:

$$x = \frac{1.2^{11} + 1.2^9 + 1.2^6 + 1.2^5 + 1.2^3 + 1.2^1}{2^{12}} \times (2 + 1) - 1$$

$$x = 0,9534.$$

Podemos usar as duas representações para o mesmo indivíduo.


Se a função objetivo é descrita por:

$$f(x) = x \cdot \sin(10\pi x) + 2.0$$

$$f(0,9534) = 0,9534 \times \sin(10\pi \times 0,9534) + 2.0 = 1,0520.$$

## Representação Binária

01100011111001010000...11111



The diagram shows a binary string "01100011111001010000...11111". Below the string, three blue brackets are used to group parts of it. The first bracket is under "01100" and labeled  $x_1$ . The second bracket is under "01111" and labeled  $x_2$ . The third bracket is under the final "11111" and labeled  $x_N$ .

## Representação Inteira

12151016...31

Nos GAs, os operadores de cruzamento e mutação permitem mover uma população em torno de uma superfície, definida pela sua função fitness.

A representação binária possui suas desvantagens:

◇ *Hamming Cliff*: Distância Hamming ( $D_h$ ) corresponde ao total de posições onde as soluções diferem.

Distância entre as soluções  $s_1$  e  $s_2$  é de 1:

$$s_1 = 'A \ G \ G \ M \ C \ G \ B \ L'$$

$$s_2 = 'M \ G \ G \ M \ C \ G \ B \ L'$$

Hamming Cliffs são grandes diferenças de bits nas cadeias de caracteres que codificam dois inteiros adjacentes.

Na representação binária, a solução  $s_1 = ( \mathbf{0 \ 1 \ 1 \ 1} )$  é vizinha de  $s_2 = ( \mathbf{1 \ 0 \ 0 \ 0} )$  no domínio discreto dos inteiros: Passamos do valor inteiro 7 para o valor 8.

Porém,  $D_h(s_1, s_2) = 4$ . Para chegar ao valor 8, partindo do vizinho 7, tem-se que mudar todo conteúdo da solução.

◊ *Codificação Gray*: Solução para o Hamming Cliff. Nesta representação, a distância Hamming entre dois inteiros consecutivos (adjacentes) é de 1 bit somente.

Decimal	Binária	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010

Seja um cromossomo  $b_i = (b_1, b_2, \dots, b_{l_i})$  no código binário e seja  $g_i = (g_1, g_2, \dots, g_{l_i})$  sua representação correspondente no código Gray.



A conversão entre o código binário e o código Gray, ou vice-versa, pode ser definida com base no operador "Adição módulo 2" ou no "OU exclusivo"  $\approx 0 \otimes 0 = 0, 0 \otimes 1 = 1 \otimes 0 = 1$  e  $1 \otimes 1 = 0$

Algoritmo para conversão de binário para gray\*

---

**Algorithm 2** Binario\_para\_Gray(*b*)

---

*n* - Número de bits da cadeia de caracteres;

*G* - Vetor de caracteres codificado em Gray;

*b* - Vetor de caracteres codificado em binário;  $\{XOR(1,0)=XOR(0,1)=1; XOR(0,0)=XOR(1,1)=0\}$

*i*  $\leftarrow$  1;

*G*[*i*]  $\leftarrow$  *b*[*i*];

**while** *i*  $\neq$  *n* **do**

*i*  $\leftarrow$  *i* + 1;

*G*[*i*]  $\leftarrow$  XOR(*b*[*i* - 1], *b*[*i*]);

**end while**

**RETURN** *G*;

---

\*Material do Professor João Antônio de Vasconcelos -  
cpdee.ufmg.br/ joao

Algoritmo para conversão de Gray para binário\*

---

**Algorithm 3** Gray\_para\_Binario( $G$ )

---

$n$  - Número de bits da cadeia de caracteres;

$G$  - Vetor codificado em Gray;

$b$  - Vetor codificado em binário;

$\{\text{XOR}(1,0)=\text{XOR}(0,1)=1; \text{XOR}(0,0)=\text{XOR}(1,1)=0\}$

$i \leftarrow 1;$

$b[i] \leftarrow G[i];$

**while**  $i \neq n$  **do**

$i \leftarrow i + 1;$

$b[i] \leftarrow \text{XOR}(b[i - 1], G[i]);$

**end while**

**RETURN**  $b;$

---

\*Material do Professor João Antônio de Vasconcelos -  
cpdee.ufmg.br/ joao

Conversão de inteiro codificado em Gray para número real entre limites mínimo e máximo.

---

**Algorithm 4** Gray\_para\_real( $G, x_i^{min}, x_i^{max}$  )

---

$G$  - Vetor de caracteres codificado em Gray;

$b$  - Vetor de caracteres (binário);

$l_i$  - Número de bits da solução  $i$ ;

$b_{i,1} \leftarrow G_{i,1}$ ;

$b_{i,j} \leftarrow XOR(b_{i,j-1}, G_{i,j}), j = 2, \dots, l_i$ ;

$$x_i \leftarrow x_i^{min} + \frac{x_i^{max} - x_i^{min}}{2^{l_i} - 1} \sum_{j=1}^{l_i} b_{i,j} 2^{l_i-j};$$

**RETURN**  $x_i$ ;

---

## Exercício (2 pontos)

Dados os indivíduos  $x_1$  ,  $x_2$  e  $x_3$ :

$1\ 1\ 0\ 0\ 1\ | \underline{0\ 1\ 0\ 1\ 1}\ | \mathbf{0\ 0\ 0\ 1\ 1}$

1. Transformar o valor de  $x_2$  do código Gray para base 10, sendo  $x_2^{min} = 0$  e  $x_2^{max} = 31$ .
2. Transformar o valor de  $x_3$  do código Gray para base 10, sendo  $x_3^{min} = -2$  e  $x_3^{max} = 2,5$ .

## Vizinhança

O conceito de vizinhança na codificação binária não é a mesma para codificação real. Na codificação binária, define-se a vizinhança de duas soluções em função da distância de Hamming → Se a vizinhança entre duas cadeias binárias não é maior do que, por exemplo 1 (um), elas são soluções vizinhas.

Assim,  $s_1 = (0 \ 0 \ 0)$  e  $s_2 = (1 \ 0 \ 0)$  são vizinhos. Porém, o fenótipo de  $s_1$  e  $s_2$  são 0 e 4 respectivamente. Elas são distantes entre si no espaço das decisões.

2). *Real*: Vetor de números reais que engloba valores reais. Interessante para problemas onde se trabalha com variáveis reais.

Exemplo:

$$s_1 = [0.525, 10.456, \dots, 0.123]$$

2). *Inteira* : Vetor de números inteiros.

## **Problemas**

A codificação é uma das etapas críticas de um GA. Uma codificação inadequada pode:

- ◇ Levar a convergência prematura do algoritmo;
- ◇ Impedir a exploração de certas regiões do espaço de busca;
- ◇ Alguns autores argumentam que o GA binário apresenta um desempenho ruim quando aplicado a problemas numérico de alta dimensionalidade, onde exige-se alta precisão.

*Problema de 100 variáveis com domínio no intervalo de  $[-500...500]$  com 6 dígitos de precisão após a casa decimal.*

## Função de Aptidão (Fitness Function)

Problema Original:

Minimize  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

Sujeito a:

$$g_i(x) \geq 0 \quad i = \{1, \dots, q\}$$

$$g_i(x) = 0 \quad i = \{q + 1, \dots, m\}$$

$$x_j^{min} \leq x_j \leq x_j^{max} \quad j \in \{1, \dots, K\}$$

Problema Transformado:

Minimize  $h(x, r, s) : f(x) + r \sum_{i=1}^q [g_i(x)]_+^2 + s \sum_{i=q+1}^m [g_i(x)]^2$

Onde:

- ◇  $r, s$  : Parâmetros de penalidade;
- ◇  $[]_+$  : Restrições de desigualdades violadas.

## Função de desempenho - $d(x)$

- *Método da Inversão*:  $d(x) = \frac{1}{h(x) + \lambda}$

$\lambda$  : Constante tal que  $h(x^*) + \lambda = \epsilon \equiv 10^{-n}$ .

- *Método do Deslocamento*:

- $d(x) = C_{max} - h(x)$  se  $h(x) < C_{max}$ ;
- $d(x) = 0$  se  $h(x) \geq C_{max}$ .

$C_{max}$  pode ser constante ou variar ao longo das gerações.  $C_{max} = K \sum_{i=1}^{N_{pop}} h_i(x) \frac{1}{N_{pop}}$ . Por exemplo  $k=1.2$



## Algoritmo Genético Simples

### Geração da População Inicial

---

**Algorithm 5** GA\_Simples ()

---

$t = 0$ ;

Inicializa ( $P(t)$ ,  $M$ ,  $I$ );

Avalia-se ( $P(t)$ );

**while** Critério de parada não atendido **do**

$t \leftarrow t + 1$ ;

$S(t) \leftarrow \text{Selecao } (P(t-1))$ ;

$Q(t) \leftarrow \text{Cruzamento } (S(t))$ ;

$Q(t)' \leftarrow \text{Mutacao } (Q(t))$ ;

$Q(t)' \leftarrow \text{Avalia-se } (Q(t))$ ;

$P(t) \leftarrow \text{Substituicao } ( P(t-1) \cup Q(t)' );$

**end while**

**Return**  $P(t)$ ;

---

## Geração da população inicial

Mais implementado: Geração aleatória do conjunto de soluções candidatas.

Porém, se existe algum conhecimento inicial sobre o problema, ele pode ser utilizado na inicialização da população:

- No caso de codificação binária, se é sabido que a solução final vai apresentar mais 0's do que 1's, esta informação pode ser útil.
- ◇ Em problemas com restrições, deve-se tomar cuidado para não gerar indivíduos inválidos \*.

Mecanismo simples para gerar uma solução inicial (binária) de forma aleatória: Para cada gene do cromossomo, seu alelo assume o valor 1 com probabilidade 0,5 e vice-versa.

\*Von Zuben: Computação Evolutiva - Uma abordagem prática

---

**Algorithm 6** inicializa (*popSize*, *nbits*)

---

```
1:  $P \leftarrow \emptyset$ ;  
2: for  $i \leftarrow 1 : popSize$  do  
3:   for  $j \leftarrow 1 : nbits$  do  
4:      $P[i, j] \leftarrow 0$ ;  
5:     if  $random() < 0,50$  then  
6:        $P[i, j] \leftarrow 1$ ;  
7:     end if  
8:   end for  
9: end for  
10: RETURN  $P$ 
```

---

Outra técnica: Dividir o espaço de busca em diversas grids. Seleciona-se uma grid aleatoriamente e gera-se (também aleatoriamente) uma solução para aquela grid.

## Algoritmo Genético Simples

### Seleção

---

#### Algorithm 7 GA\_Simples ()

---

$t = 0$ ;

Inicializa ( $P(t)$ ,  $M$ ,  $I$ );

Avalia-se ( $P(t)$ );

**while** Critério de parada não atendido **do**

$t \leftarrow t + 1$ ;

$S(t) \leftarrow \text{Selecao } (P(t-1))$ ;

$Q(t) \leftarrow \text{Cruzamento } (S(t))$ ;

$Q(t)' \leftarrow \text{Mutacao } (Q(t))$ ;

$Q(t)' \leftarrow \text{Avalia-se } (Q(t))$ ;

$P(t) \leftarrow \text{Substituicao } ( P(t-1) \cup Q(t)' );$

**end while**

**Return**  $P(t)$ ;

---

## Operador de Seleção

Processo de escolha de soluções para reprodução/sobrevivência - Implementado por um EA.

Primeiramente, computa-se a aptidão do conjunto de soluções. Posteriormente, faz-se a seleção dos indivíduos.

*Seleção Natural* - Maior probabilidade de reprodução para os indivíduos com *maior (melhor)* aptidão.

**Pressão de Seleção:** Caracteriza os operadores de seleção e indica o quão os melhores indivíduos são selecionados → Quanto maior a pressão de seleção, mais os melhores indivíduos são selecionados.

Alguns autores enfatizam que a seleção atua no sentido de eliminar componentes menos apropriados da população: Os recursos disponíveis são limitados (Competição pela sobrevivência).

Vários métodos de seleção baseiam-se numa probabilidade computada com base na aptidão (geralmente descrita pela função objetivo) do indivíduo.

Normalmente, cria-se um arquivo (*mating pool*), de tamanho *popsiz*e, onde são mantidas as *soluções selecionadas* para reprodução.

*Seleção proporcional*: Método proposto no GA de Holland (1975) onde se gera um conjunto de soluções com base na aptidão das soluções.

A probabilidade de seleção do indivíduo  $i$  ( $p_i$ ) é proporcional ao seu valor de fitness:

$$p_i = \frac{f_i}{\sum f_i}$$

Onde:

$\forall p_i$  : Probabilidade de seleção da solução  $i$ ;

- $f_i$  : Valor do fitness da solução  $i$ ;

♡  $N$  : Tamanho da população;

★  $M$  : Fitness médio da população.

Observações:

□ Todos os valores de fitness devem ser maior do que zero;

♠ A probabilidade de seleção depende fortemente da escala da função de fitness:

Tem-se uma população com 10 indivíduos com os seguintes valores de fitness:

$$\vec{\phi} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).$$

Neste caso,  $p_b$  do melhor indivíduo é de  $\approx 18,2\%(\frac{10}{55})$  e  $p_w$  do pior indivíduo é de  $\approx 1,8\%(\frac{1}{55})$ .



Se adicionarmos 100 a cada valor de fitness:  
 $p_b \approx 10,4\%$  e  $p_w \approx 9,6\%$

Outros métodos:

$$p_i = \frac{Fitness(s_i)}{K},$$

Onde

$$k = \sum_{j \leftarrow 1:M} Fitness(s_j)$$

ou

$k$  = Fitness da pior solução.

*Escalonamento do fitness*

Durante a execução de um EA, algumas vezes a população torna-se dominada por indivíduos com alto desempenho e com intervalo reduzido no espaço dos objetivos.

Neste caso, calcular a aptidão empregando os métodos vistos anteriormente resulta em um fitness semelhante para o conjunto de soluções da população → Perde-se a pressão de seleção em relação às melhores soluções.

O escalonamento de fitness acentua as diferenças reduzidas nos valores dos objetivos e permite aumentar a pressão de seleção.

1. *Fitness linear*: A função objetivo assume valores que variam ao longo do tempo:

$$\Phi(a_i(t)) = \alpha f(a_i(t)) - \beta(t)$$

onde:

♣  $\Phi$  mapeia o valor da função objetivo num intervalo não negativo;

□  $\alpha = +1(-1)$  para problemas de maximização (minimização);

♠  $\beta(t)$  corresponde ao pior valor encontrado nas últimas gerações.

2. *Power law*:  $\Phi(a_i(t)) = f(a_i(t))^k$

onde  $k$  depende do problema que está sendo resolvido.

3. *Variável no tempo*:  $\Phi(a_i(t)) = \exp(f(a_i(t))/T)$

$T$  serve para controlar a pressão de seleção durante o processamento do algoritmo.

### *Ranqueamento do fitness*

A probabilidade de seleção é computada com base em um *rank* atribuído ao conjunto de soluções → dispensa o escalonamento do fitness.

Rank mapeia a função objetivo  $f$  para a função aptidão  $\Phi$ :

$$\Phi(a_i) = \alpha f(a_i)$$

□  $\alpha = +1(-1)$  para problemas de maximização (minimização).

- Evita a convergencia prematura - O super indivíduo destaca se pouco em relação aos demais: Pior indivíduo tem fitness **1**; segundo pior **2** e o melhor tem fitness **N** (N - Tamanho da população);
  - Problemas onde o cálculo da função objetivo envolve preferências a soluções alternativas - Deve se dedicar pouca atenção aos valores da função objetivo.
1. *Rank linear - RL*: Atribui uma probabilidade de seleção para cada indivíduo que é proporcional ao rank (posição do indivíduo dentro da população) do indivíduo. O indivíduo menos apto tem rank 0.

$$P(i)_{RL} = \frac{\alpha_{rank}[\text{rank}_i/(N-1)](\beta_{rank} - \alpha_{rank})}{N}$$

onde:

$\alpha_{rank}$  representa o número de filhos alocados à pior solução;

$\beta_{rank}$  representa o número de filhos alocados à melhor solução;

$N$  corresponde ao tamanho da população;

$\text{rank}_i$ : Posição (rank) do indivíduo dentro da população. O pior indivíduo tem aptidão igual a zero.

Normalmente

$$\alpha_{rank} = 2 - \beta_{rank} \text{ e } 1 \leq \beta_{rank} \leq 2.$$

Exemplo\*

\*[http://www.dca.ufrn.br/fane/metaheuristicas/ag\\_pratica.pdf](http://www.dca.ufrn.br/fane/metaheuristicas/ag_pratica.pdf)

$S_i$	$f(S_i)$	rank	Aptidão	$p_i$
$S_1$	200,999588	1	2,0	40%
$S_2$	200,826877	2	1,5	30%
$S_3$	200,655533	3	1,0	20%
$S_4$	200,400148	4	0,5	10%
$S_5$	200,102002	5	0,0	0%

Após computar o fitness das soluções, precisamos computar a probabilidade de seleção das soluções.

Existem vários procedimentos para seleção das soluções para reprodução. Vamos a eles.

## Randômica

Um pai é selecionado aleatoriamente da população.

## Roleta

Os indivíduos com maior aptidão ocuparão uma fatia maior da roleta. A roleta é girada diversas vezes, e, em cada giro, um indivíduo é selecionado para participar do processo de cruzamento.

	$S_i$	$f(S_i)$	$p_i$
$S_1$	10110	2,23	0.14
$S_2$	11000	7,27	0.47
$S_3$	11110	1,05	0.07
$S_4$	01001	3,35	0.21
$S_5$	00110	1,69	0.11

*Implementação:*

$S_i$	$p_i$	Acumulada	$rand$
$S_1$		0,14	0,00 .. 0,13
$S_2$		0,61	0,14 .. 0,60
$S_3$		0,68	0,61 .. 0,67
$S_4$		0,89	0,68 .. 0,88
$S_5$		1,00	0,89 .. 0,99

---

**Algorithm 8** roulette ()

---

```
1:  $t \leftarrow rand$ ;  
2: if  $t < 0,14$  then  
3:   RETURN  $s_1$ ;  
4: else if  $t < 0,61$  then  
5:   RETURN  $s_2$ ;  
6: else if  $t < 0,68$  then  
7:   RETURN  $s_3$ ;  
8: else if  $t < 0,89$  then  
9:   RETURN  $s_4$ ;  
10: else  
11:   RETURN  $s_5$ ;  
12: end if
```

---



O algoritmo *roulette* é executado *popsiz*e vezes.

Independente de ser ou não selecionada, se a solução pega aleatoriamente é retornada para a população, podendo ser escolhida novamente, tem-se uma *seleção com substituição* (*selection with replacement*).

Caso contrário, tem-se uma *seleção sem substituição* (*selection without replacement*).

## **Amostragem estocástica universal**

Os indivíduos são mapeados em segmentos contínuos de uma reta, tal que, onde o tamanho do segmento de cada indivíduo é proporcional à sua aptidão (semelhante à roleta).

Pontos (Número de indivíduos para seleção) equidistantes são posicionados no segmento de reta. Seja  $NP$  total de indivíduos para seleção

→ A distância entre os pontos é de  $\frac{1}{NP}$  e a posição do primeiro ponto é determinada por um número aleatório no intervalo de  $[0, \frac{1}{NP}]$ .

## Problemas com métodos baseados no fitness

1. Se a população possui uma *supersolução* (solução que possui fitness muito superior ao fitness do restante da população), esta será escolhida na maioria das vezes → Perda da diversidade e convergência prematura para um ótimo local;
2. A maioria das soluções possui fitness semelhante. Neste caso, o processo de seleção se assemelha a um procedimento aleatório.

Uma solução para o problema levantado

## Torneio

Em um torneio de  $q$  soluções, a melhor solução é escolhida (de forma determinística ou probabilística)  $\rightarrow$  Torneio binário ( $q = 2$ ) é o mais comum.

Este método pode ser aplicado em problemas de minimização e maximização e em problemas onde a função objetivo pode assumir valores negativos.

## Considerações

Se o operador de seleção tem pressão de seleção:

- *Alta*: a população perde diversidade rapidamente. Uma saída é trabalhar com populações grandes (ou numerosas) ou implementar operadores de mutação e cruzamento disruptivos;

- *Baixa*: Convergência lenta, porém, explora o espaço de busca mais apropriadamente.

Assim, a pressão de seleção de um operador de seleção depende dos operadores de cruzamento e mutação.

## Algoritmo Genético Simples

### Cruzamento

---

#### Algorithm 9 GA\_Simples ()

---

```
 $t = 0;$   
Inicializa (P(t), M, I);  
Avalia-se (P(t));  
while Critério de parada não atendido do  
     $t \leftarrow t + 1;$   
    S(t)  $\leftarrow$  Selecao (P(t-1));  
    Q(t)  $\leftarrow$  Cruzamento (S(t));  
    Q(t)'  $\leftarrow$  Mutacao (Q(t));  
    Q(t)'  $\leftarrow$  Avalia-se (Q(t));  
    P(t)  $\leftarrow$  Substituicao ( P(t-1) U Q(t)' );  
end while  
Return P(t);
```

---

## Operador de Cruzamento (Crossover)

Objetivo: Troca de informações entre soluções distintas.

Principal operador de exploração do espaço de busca do GA. Envolve 03 etapas:

- Segundo uma probabilidade de cruzamento ( $P_c$ ), 02 soluções são escolhidas de forma aleatória;
- Um ponto de corte, limitado pelo *comprimento* das soluções, é escolhido aleatoriamente;
- O código genético das duas strings é trocado segundo o ponto de corte.

$P_c$  é um parâmetro que indica a frequência com que o operador de cruzamento será aplicado.

Se não existe cruzamento, as soluções filhas são cópias fiéis dos seus pais. Caso contrário, as soluções filhas são constituídas da combinação do material genético dos seus pais.

Se  $P_c = 100\%$ , todos os filhos são gerados a partir do cruzamento das soluções. Se  $P_c = 0\%$ , as soluções da nova geração são cópias exatas das soluções da população anterior (Não significa dizer que a nova geração é a mesma).

02 formas de escolha das soluções para cruzamento:

★ O mating pool é embaralhado e formam-se os seguintes pares:  $1 \rightarrow 2$ ,  $3 \rightarrow 4$ , etc...

- Dois números aleatórios são gerados (um par). O processo é repetido  $\frac{popsize}{2}$  vezes.

---

**Algorithm 10** CrossOverOperator ( $MP, P_c, l$ )

---

```
1:  $Q \leftarrow \emptyset$ ;  
2: for  $i \leftarrow 1 : MP.size()/2$  do  
3:   if  $random() \leq P_c$  then  
4:      $Q \leftarrow Q \cup crossOver(MP, MP.size(), l)$ ;  
5:   end if  
6: end for  
7: RETURN  $Q$ ;
```

---



## Tipos de cruzamento

*1. Um ponto de corte:* Duas soluções "Pais"

Pai 1: 1 0 1 0 | 0 1 1 1 0 1

Pai 2: 0 1 1 1 | 0 1 0 1 1 0

Após o cruzamento, 02 soluções filhas são criadas:

Filho 1: 0 1 1 1 | 0 1 1 1 0 1

Filho 2: 1 0 1 0 | 0 1 0 1 1 0

---

**Algorithm 11** onePC ( $P, N, l$ )

---

```
1:  $p_1 \leftarrow \text{int.random}(N)$ ;  
2:  $p_2 \leftarrow \text{int.random}(N)$ ;  
3:  $q \leftarrow \text{int.random}(l)$ ;  
4:  $F_1, F_2 \leftarrow \emptyset$ ;  
5: for  $i \leftarrow 1 : l$  do  
6:   if  $i \leq q$  then  
7:      $F_1[i] \leftarrow P[p_2, i]$   
8:      $F_2[i] \leftarrow P[p_1, i]$   
9:   else  
10:     $F_1[i] \leftarrow P[p_1, i]$   
11:     $F_2[i] \leftarrow P[p_2, i]$   
12:   end if  
13: end for  
14: RETURN  $[F_1, F_2]$ ;
```

---

*2. Dois pontos de corte:* Duas soluções são cruzadas segundo dois pontos (p e q), definidos aleatoriamente, gerando 02 soluções filhas.

Pai 1: **1 0 1** | 0 0 1 1 | 1 0 1

Pai 2: 0 1 1 | **1 0 1 0** | 1 1 0

Após o cruzamento:

Filho 1: **1 0 1** | **1 0 1 0** | 1 0 1

Filho 2: 0 1 1 | 0 0 1 1 | 1 1 0

---

**Algorithm 12** twoPC ( $P, N, l$ )

---

```
1:  $p_1 \leftarrow \text{int.random}(N)$ ;  
2:  $p_2 \leftarrow \text{int.random}(N)$ ;  
3:  $p, q \leftarrow \text{int.random}(l)$ ;  $\{p < q\}$   
4:  $F_1, F_2 \leftarrow \emptyset$ ;  
5: for  $i \leftarrow 1 : l$  do  
6:   if  $(i \leq p)$  or  $(i > q)$  then  
7:      $F_1[i] \leftarrow P[p_1, i]$   
8:      $F_2[i] \leftarrow P[p_2, i]$   
9:   else  
10:     $F_1[i] \leftarrow P[p_2, i]$   
11:     $F_2[i] \leftarrow P[p_1, i]$   
12:   end if  
13: end for  
14: RETURN  $[F_1, F_2]$ ;
```

---

**3. Cruzamento Uniforme:** Para cada bit do primeiro filho, determina-se qual pai vai contribuir para aquela posição segundo uma probabilidade fixa  $p$ .

Pai 1: 1 1 1 0 0 1 1 0 0 0 1 1

Pai 2: 0 0 0 1 0 1 0 0 1 0 0 1

Vetor aleatório

(0.55 0.03 0.67 0.77 0.30 0.25 0.99 0.78 0.10 0.58 0.87 0.73)

Filho 1: 0 1 0 1 0 1 0 0 0 0 0 1

Filho 2: 1 0 1 0 0 1 1 0 1 0 1 1

### Outra formulação

Cada gene do filho é gerado a partir da cópia do gene correspondente do  $pai_1$  ou  $pai_2$ , segundo o conteúdo de uma máscara binária de cruzamento (com mesmo comprimento das soluções), criada aleatoriamente: Se tiver bit

**1** na máscara, copia se o conteúdo de  $pai_1$  para  $filho_1$  e  $pai_2$  para  $filho_2$  e vice-versa para bit **0**. Gera-se uma máscara para cada par de soluções.

Pai 1 :    1 0 1 1 0 0 1 1

Pai 2 :    0 0 0 1 1 0 1 0

Máscara: 1 1 0 1 0 1 1 0

Filho 1:  1 0 0 1 1 0 1 0

Filho 2:  0 0 1 1 0 0 1 1

Quando tem-se *cruzamento uniforme parametrizado*, do tipo "dois pais, um filho", define-se uma probabilidade  $p$  para o filho herdar genes do primeiro pai e  $1 - p$  para herdar genes do segundo pai.

Para  $p = 0.5$ , tem-se o *cruzamento uniforme discreto*.

$p$  também pode ser computado segundo o peso relativo de dois pais, isto é,  $p = \frac{fp_1}{fp_1 + fp_2}$  para problemas de maximização.

*4. Cruzamento com 03 Pais:* 03 pais são escolhidos aleatoriamente. Cada bit do primeiro pai é comparado com o bit do segundo pai. se ambos são iguais, o bit é copiado para o filho, caso contrário, o bit da 3o. solução é copiado para o filho.

*5. Surrogate reduzido:* Restringe se os pontos de cruzamento nos locais onde os valores dos genes são distintos.

*6. Cruzamento embaralhado:* Seleciona-se um ponto de cruzamento (Similar ao cruzamento com um ponto de corte). Antes da realização do cruzamento, as variáveis dos pais são embaralhadas. Após a recombinação, as variáveis da solução filha são desembaralhadas (???)

Considerações:

★ Cada operador de cruzamento é particularmente eficiente para uma determinada classe

de problemas e extremamente ineficiente para outras.

♡ Para representação em ponto flutuante (inteira), outros operadores podem ser implementados.

♠ Existem operadores específicos para problemas de natureza combinatória.



## Algoritmo Genético Simples

### Mutação

---

#### Algorithm 13 GA\_Simples ()

---

$t = 0$ ;

Inicializa (P(t), M, I);

Avalia-se (P(t));

**while** Critério de parada não atendido **do**

$t \leftarrow t + 1$ ;

    S(t)  $\leftarrow$  Selecao (P(t-1));

    Q(t)  $\leftarrow$  Cruzamento (S(t));

    Q(t)'  $\leftarrow$  Mutacao (Q(t));

    Q(t)'  $\leftarrow$  Avalia-se (Q(t));

    P(t)  $\leftarrow$  Substituicao ( P(t-1) U Q(t)' );

**end while**

**Return** P(t);

---

## Operador de Mutação

Envolve uma solução somente.

Objetivo: Introduzir uma variabilidade extra na *população filha*, resultante do cruzamento (*mating pool*), sem alterar significativamente os resultados obtidos até então.

O indivíduo mutado é chamado de *mutante*.

**Mutação padrão:** Cada bit (gene) de uma solução é modificado segundo uma probabilidade ( $P_m$ ), normalmente pequena. Se o valor do bit é 1, mudamos seu valor para 0 e vice-versa.

**1-bit:** Uma posição da string tem o valor do seu bit invertido.

Pai : 1 0 1 1 0 1 0 1

Filho : 1 0 1 1 1 1 0 1

**Alternância:** Duas posições da string são escolhidas e seus bits são trocados.

Pai :    1 0 1 1 0 1 0 1

Filho : 0 0 1 1 1 1 0 1

**Reversão:** Uma posição aleatória é escolhida e os bits próximos são invertidos.

Pai :    1 0 1 1 0 1 0 1

Filho : 1 0 1 1 0 1 1 0

O operador de Mutação se aplica a todas as soluções geradas no cruzamento.

---

**Algorithm 14** mutationOperator ( $Q, P_m, l$ )

---

```
1: for  $i \leftarrow 1 : Q.size()$  do  
2:    $Q[i] \leftarrow mutation(Q[i], P_m, l);$   
3: end for  
4: RETURN  $Q;$ 
```

---

Implementação da Mutação

---

**Algorithm 15** mutation ( $s, P_m, l$ )

---

```
1: for  $i \leftarrow 1 : l$  do  
2:   if  $random() \leq P_m$  then  
3:      $s[i].invertBit();$   
4:   end if  
5: end for  
6: RETURN  $s;$ 
```

---

## Algoritmo Genético Simples

### Substituição

---

#### Algorithm 16 GA\_Simples ()

---

$t = 0$ ;

Inicializa (P(t), M, I);

Avalia-se (P(t));

**while** Critério de parada não atendido **do**

$t \leftarrow t + 1$ ;

    S(t)  $\leftarrow$  Selecao (P(t-1));

    Q(t)  $\leftarrow$  Cruzamento (S(t));

    Q(t)'  $\leftarrow$  Mutacao (Q(t));

    Q(t)'  $\leftarrow$  Avalia-se (Q(t));

    P(t)  $\leftarrow$  Substituicao ( P(t-1) U Q(t)' );

**end while**

**Return** P(t);

---

## Operador de Substituição

*Estratégia simples:* Gera-se um (ou dois) filho(s) a cada geração que substitui a pior (ou duas piores) solução(ções) da população.

Outra estratégia: São escolhidas as duas melhores soluções do conjunto

$$(pai_i, pai_{i+1}, filho_i, filho_{i+1}).$$

Atenção: Alguns trabalhos implementam um tipo de cruzamento 2PC que produz um filho somente.

## Elitismo

A(O) melhor solução (conjunto de soluções) é escolhida (o) para compor o conjunto solução da próxima geração.

Taxa de elitismo: Percentual das melhores soluções da geração corrente para participar da geração seguinte.

## Condição de Parada

O conjunto de operações de seleção, cruzamento, mutação e substituição correspondem a uma iteração (geração) do GA.

Assim, o processamento do GA é concluído quando:

- ◇ Número de gerações ( $maxgen$ ) ou de avaliações da função objetivo;
- ◇ Após  $t$  unidades de tempo de processamento;
- ◇ Após  $k$  iterações sem evoluir o melhor fitness do conjunto de soluções.

## Representação real

Na codificação real, cada solução (indivíduo) da população é representada por um vetor de número reais.

$X = [x_1, x_2, \dots, x_{nvar}]$ . O número de variáveis ( $nvar$ ) determina a dimensão do espaço de busca.

Exemplo:  $X = [0,259 \quad 10,231 \quad \dots \quad 0,333]$

Alguns operadores foram propostos para codificação real. Vamos a eles ....

### 1. Cruzamento

- **Média** - A solução filha é resultante da média aritmética das soluções pais;
- **Média Geométrica** - Toma-se a raiz quadrada do produto dos dois valores;



- **Extensão** - Tira-se a diferença entre os dois valores e adiciona (ou subtrai) o valor da diferença do maior ou do menor valor.
- **Cruzamento polarizado** - Sejam  $k$  - Número aleatório que identifica a variável no intervalo  $[1, \dots, nvar]$ ;  $\alpha_{pol}$  - Coeficiente de multiplicação *polarizado* no intervalo  $]0,5 \ 1.0]$ ;  $\alpha$  - Coeficiente de multiplicação no intervalo  $[-0,1 \dots 1,1]$ ;  $dir$  - Variável binária aleatória que indica a direção do cruzamento: Do ponto de corte até a última variável ( $dir = nvar$ ) ou da primeira variável até o ponto de corte ( $dir = 1$ );  $i$  e  $j$  dois indivíduos selecionados para cruzamento sendo a aptidão de  $i$  melhor que a de  $j$ .

As variáveis fora do intervalo são copiadas diretamente do progenitor.

Para  $dir = 1$

$$F_{1...k}^1 = \alpha_{pol} X_{1...k}^i + (1 - \alpha_{pol}) X_{1...k}^j$$

$$F_{1...k}^2 = (1 - \alpha) X_{1...k}^i + \alpha X_{1...k}^j$$

Para  $dir = nvar$

$$F_{k...nvar}^1 = \alpha_{pol} X_{k...nvar}^i + (1 - \alpha_{pol}) X_{k...nvar}^j$$

$$F_{k...nvar}^2 = (1 - \alpha) X_{k...nvar}^i + \alpha X_{k...nvar}^j$$

## 2. Mutação

- **Substituição aleatória** - Substitua o valor por um número aleatório;
- **Creep** - Adicione (ou subtraia) um valor pequeno (gerado aleatoriamente) da solução;
- **Creep geométrico** - Multiplique por um valor aleatório próximo de 1.
- **Mutação real** - Similar ao cruzamento polarizado, faz-se a mutação ou não de acordo com  $k$ . A mutação consiste em somar ao indivíduo um vetor ( $\gamma$ ):

1.  $dir = 1 \rightarrow \gamma_{1...k} = 0,05 \beta \theta_{1...k}$
2.  $dir = nvar \rightarrow \gamma_{k...nvar} = 0,05 \beta \theta_{k...nvar}$

Sendo:  $k$  - Ponto de referência para realização da mutação;  $-1 \leq \beta \leq 1$  - Número aleatório;  $\theta$  - Amplitude da faixa definida pelo limite máximo e mínimo de cada variável ( $X_{max} - X_{min}$ );  $dir$  - Direção para realização da mutação.

## Exercícios

1. Trabalho do professor Renato Dourado;
2. Maximize  $f(x) = x.\sin(10\pi x) + 2.0$

Sujeito a:  $-1 \leq x \leq 2$ .

Parâmetros:

$popsize = 10$ ,  $maxgen = 10$ ,  $p_c = 0.80$  e  $p_m = 0.01$ .

Construa um gráfico ilustrando a evolução da função objetivo ao longo das gerações após 50 execuções do GA.

Teste os vários operadores de seleção, cruzamento e mutação e desenvolva um documento mostrando a combinação que produziu os melhores resultados para este problema. Adicione no documento um gráfico ilustrando a evolução da função objetivo ao

longo das gerações após 50 execuções do GA para cada uma das combinações.

Implemente a codificação gray e binária e compare os resultados obtidos. Utilize o gráfico de evolução da função objetivo para comparar os resultados obtidos com ambas as representações.

3. Implemente um algoritmo genético para minimizar a função seguinte

$$f(x) = x^2 + y^2$$

Sujeito a:

$$x + y = 7$$

$$1 \leq x \leq 5 \text{ e } y \geq 3.$$

Parâmetros:

$$popsize = 10, \text{ maxgen} = 10, p_c = 0.80 \text{ e } p_m = 0.01.$$

Construa um gráfico ilustrando a evolução da função objetivo ao longo das gerações após 50 execuções do GA.

4. Minimize  $f(x) = An + \sum_{i=1}^n [x_i^2 - A \cos(2\pi x_i)]$   
Onde  $A = 10$  e  $x_i \in [-5, 12 \quad 5, 12]$

Verifique se os parâmetros anteriores são suficientes para resolução deste problema. Caso não sejam, faça os devidos ajustes para que se chegue à solução do problema. Trabalhe com  $n = 2, 5$  e  $10$ .

## Referências Bibliográficas

1. Back, T.; Fogel, D.B; Michalewicz, Z. **Evolutionary Computation 1 - Basic Algorithms and Operators**. Institute of Physics Publishing. Bristol and Philadelphia. 2000.
2. Sivanandam, S.N.; Deepa, S.N. **Introduction to Genetic Algorithms**. Springer-Verlag Berlin Heidelberg. 2008.
3. Homepage: Fernando J. Von Zuben (UNICAMP).

## Problemas discretos

### Vehicle Routing Problem\*

Tem-se uma frota de veículos para entregar produtos em diversos clientes (com demandas variadas), partindo de um depósito. Deve-se determinar um conjunto *ótimo* (com base na distância ou custo) de rotas para os veículos.

Restrições principais: demanda dos clientes e capacidade dos veículos.

### Operadores de cruzamento para o VRP

- *Order crossover (OX)*: Dois pontos de cortes são gerados aleatoriamente. O material genético, localizado entre os pontos de

\* *Comparison of eight evolutionary crossover operators for the vehicle routing problem.* Krunoslav Puljic and Robert Manger. 2013(18). MATHEMATICAL COMMUNICATIONS. 359-375.



corte do 1o. pai, é copiado para o filho. O restante do material é copiado do 2o. pai, partindo do 2o. ponto de corte (busca cíclica quando se chega ao fim do 2o. pai);

Pais:

$$p_1 = ( 1 \ 2 \ 3 \mid 5 \ 4 \ 6 \ 7 \mid 8 \ 9 )$$

$$p_2 = ( 4 \ 5 \ 2 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 3 )$$

Filhos:

$$f_1 = ( 2 \ 1 \ 8 \mid 5 \ 4 \ 6 \ 7 \mid 9 \ 3 )$$

Procedimento semelhante é utilizado para construção do segundo filho:

$$f_2 = ( 3 \ 5 \ 4 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 2 )$$

- *Partially mapped crossover (PMX)*: Semelhante ao OX: O material genético entre os dois pontos de corte  $p_1$  é copiado para  $f_1$ . O código restante do filho é preenchido

com material do 2o. pai respeitando, o máximo possível, as posições absolutas do código genético.

Pais:

$$p_1 = ( 1 \ 2 \ 3 \mid 5 \ 4 \ 6 \ 7 \mid 8 \ 9 )$$

$$p_2 = ( 4 \ 5 \ 2 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 3 )$$

Filhos:

$$f_1 = ( * \ * \ * \mid 5 \ 4 \ 6 \ 7 \mid * \ * )$$

$$f_2 = ( * \ * \ * \mid 1 \ 8 \ 7 \ 6 \mid * \ * )$$

Observamos a correspondência existente entre  $p_1$  e  $p_2$ :  $5 \rightarrow 1$ ,  $4 \rightarrow 8$ ,  $6 \rightarrow 7$  e  $7 \rightarrow 6$ . Esta correspondência é usada no preenchimento de  $f_1$  com material proveniente de  $p_2$ .

Tem-se assim:

$$f_1 = ( 8 \ 1 \ 2 \mid 5 \ 4 \ 6 \ 7 \mid 9 \ 3 )$$

e

$$f_2 = ( \ ? \ ? \ ? \ | \ 1 \ 8 \ 7 \ 6 \ | \ ? \ ? \ )$$

- *Edge recombination crossover (ERX)*: Interpreta o indivíduo como um conjunto de arestas não dirigidas. A idéia é que o filho herde o máximo possível de arestas dos pais - arestas comuns a ambos têm prioridade.

A rotina para geração de uma solução filha utiliza o conceito de vizinhança: O vizinho de um vértice  $i$  é outro vértice  $j$  que é adjacente à  $i$  no ciclo do primeiro ou do segundo pai.

A rotina parte de um vértice arbitrário e executa os seguintes passos: Em cada passo, o próximo vértice é escolhido entre os vizinhos do vértice anterior. Se existem diversas opções, escolhe-se o vértice com menor vizinhança. O vizinho é escolhido aleatoriamente quando existem duas ou mais

soluções vizinhas candidatas com lista de vizinhos de mesmo tamanho.

Pais:

$$p_1 = ( 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 )$$

$$p_2 = ( 4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5 )$$

Vamos construir o vizinho  $c$  partindo do vértice 1. O próximo vértice deve ser escolhido entre os vizinhos de 1 (2, 4 e 9) com menor vizinhança. 2, 4 e 9 têm, respectivamente, 3, 3 e 4 vizinhos. Seleccionamos o vértice 4 aleatoriamente.

$$c = ( 1 \ 4 \ * \ * \ * \ * \ * \ * \ * ).$$

Os vértices vizinhos de 4 são: 1, 3 e 5. Mas 1 já está na solução filha. Logo, nos restam 3 e 5. O vértice 5 é escolhido.

$$c = ( 1 \ 4 \ 5 \ * \ * \ * \ * \ * \ * ).$$

Prosseguindo, teremos ao final a seguinte solução filha:

$$c = ( 1 \ 4 \ 5 \ 6 \ 7 \ 8 \ 2 \ 3 \ 9 ).$$

- *Cycle crossover (CX)*: Funcionamento: Um vértice deve ser copiado em um filho a partir de um pai específico. Porém, a posição de inserção é herdada do outro pai:

Pais:

$$p_1 = ( 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 )$$

$$p_2 = ( 4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5 )$$

Filho - Iniciamos com o 1o. vértice de  $p_1$ :

$$f_1 = ( 1 \ * \ * \ * \ * \ * \ * \ * \ * ).$$

A seguir identificamos o vértice de  $p_2$  na posição 1 - vértice 4. 4 é copiado para  $f_1$ , porém na posição em que está localizado em  $p_1$ :

$$f_1 = ( 1 \ * \ * \ 4 \ * \ * \ * \ * \ * ).$$

Seleciona-se o vértice de  $p_2$  na posição de 4  
→ Vértice 8. Copiamos 8 em  $f_1$  na posição  
em que ele se encontra em  $p_1$ :

$$f_1 = ( 1 * * 4 * * * 8 * ).$$

Executando mais 2 passos teremos:

$$f_1 = ( 1 2 3 4 * * * 8 * ).$$

O último vértice inserido foi o 2 na posição  
2 de  $f_1$ . Porém, na posição 2 de  $p_2$  está  
o vértice 1 que já foi inserido na solução  
filha. Neste caso, somente  $p_2$  é usada para  
completar  $f_1$ .

Ao final teremos:

$$f_1 = ( 1 2 3 4 7 6 9 8 5 ).$$

A composição do filho  $f_2$  será:

$$f_2 = ( 4 1 2 8 5 6 7 3 9 ). \text{ Favor verificar se está correto!}$$

- *Alternating edges crossover (AEX):*

Interpreta uma solução como sendo um conjunto de arestas dirigidas. A solução filha é formada pela escolha de arcos do primeiro e do segundo pai, com escolhas aleatórias no caso de inviabilidade da seleção:

$$p_1 = ( 5 \ 1 \ 7 \ 8 \ 4 \ 9 \ 6 \ 2 \ 3 )$$

$$p_2 = ( 3 \ 6 \ 2 \ 5 \ 1 \ 9 \ 8 \ 4 \ 7 )$$

Escolhemos o 1o. arco  $5 \rightarrow 1$  de  $p_1$

$$c = ( 5 \ 1 \ * \ * \ * \ * \ * \ * \ * ).$$

A seguir, escolhe-se o arco de  $p_2$  que parte de 1, i.e.  $1 \rightarrow 9$ :

$$c = ( 5 \ 1 \ 9 \ * \ * \ * \ * \ * \ * ).$$

Em seguida, pega-se o arco de  $p_1$  que se inicia no vértice 9. Após alguns passos chega-se a seguinte construção:

$$c = ( 5 \ 1 \ 9 \ 6 \ 2 \ 3 \ * \ * \ * ).$$

O arco que parte do vértice 3 deve partir de  $p_2$ . Porém, esta escolha inviabiliza a solução (fecha o ciclo). A saída é selecionar aleatoriamente um dos vértices que ainda não foram visitados.

- *Heuristic greedy or random or probabilistic crossover (HGreX or HRndX or HProX)*: O HGreX se assemelha ao AEX. A solução filha é construída escolhendo o vértice *mais barato (ou econômico)* dos respectivos arcos pais para cada vértice.

$$p_1 = c_{51} = 2, c_{17} = 2, c_{78} = 6, c_{84} = 8, c_{49} = 3, c_{96} = 6, c_{62} = 4, c_{23} = 4, c_{35} = 3$$

$$p_2 = c_{36} = 6, c_{62} = 4, c_{25} = 2, c_{51} = 2, c_{19} = 3, c_{98} = 8, c_{84} = 8, c_{47} = 3, c_{73} = 5$$

Inicialmente, escolhe-se um vértice aleatoriamente, por exemplo 5. Ambos os arcos das soluções pai que partem do nó 5 são



consideradas. Introduzimos o nó  $5 \rightarrow 1$  na solução filha:

$$c = ( 5 \ 1 \ * \ * \ * \ * \ * \ * \ * ).$$

Em seguida, escolhemos o arco mais econômico em  $p_1$  e  $p_2$  que parte do nó 1.  $1 \rightarrow 7$  é adicionado na solução filha:

$$c = ( 5 \ 1 \ 7 \ * \ * \ * \ * \ * \ * ).$$

No passo seguinte seleciona-se o arco mais interessante entre  $7 \rightarrow 3$  e  $7 \rightarrow 8$ :

$$c = ( 5 \ 1 \ 7 \ 3 \ * \ * \ * \ * \ * ).$$

Na próxima etapa, avaliamos os arcos  $3 \rightarrow 5$  e  $3 \rightarrow 6$ . O primeiro é mais interessante, porém, ele inviabiliza a solução filha:

$$c = ( 5 \ 1 \ 7 \ 3 \ 6 \ * \ * \ * \ * ).$$

Passo seguinte:

$$c = ( 5 \ 1 \ 7 \ 3 \ 6 \ 2 \ * \ * \ * ).$$

Agora devemos considerar os arcos  $2 \rightarrow 3$  e  $2 \rightarrow 5$ . Porém, ambos inviabilizam a solução filha. Nesta situação, o operador gera aleatoriamente um número específico de arcos viáveis ( $2 \rightarrow 4$ ,  $2 \rightarrow 8$  e  $2 \rightarrow 9$ ) e pega o mais barato. Admitamos que  $2 \rightarrow 8$  seja o mais barato, teremos ao final a seguinte solução:

$$c = ( 5 \ 1 \ 7 \ 3 \ 6 \ 2 \ 8 \ 4 \ 9 ).$$

HRndX e HProX são variantes do HGreX. Eles utilizam critérios distintos para seleção de um arco em uma fase específica do algoritmo. Enquanto o HGreX utiliza um princípio guloso e sempre escolhe o arco mais barato entre as alternativas existentes, HRndX e HProX fazem escolhas aleatórias. Consequentemente, não existe garantia que eles vão produzir os mesmos filhos para os mesmos pais.

## Operadores de Mutação para o VRP

- *Remove-And-Reinsert (RAR)*<sup>†</sup>: Um elemento é retirado de sua posição corrente e é reinsertido em outra posição da solução.

$$s = ( 1 \ 2 \ 3 \ \underline{4} \ 7 \ 6 \ 9 \ 8 \ 5 )$$



$$s = ( 1 \ 2 \ 3 \ 7 \ 6 \ 9 \ \underline{4} \ 8 \ 5 ).$$

<sup>†</sup>Gendreau, Laporte e Potvin. Metaheuristics for the vehicle routing problem, 1999