**Instituto Politécnico de Tomar**



**Realizado por :**

Paulo Chorinca

Carlos Neto

Diogo António

Bruno Mendes

Gonçalo Dias

Fernando Saraiva

Índice

[Resumo 4](#_Toc326852199)

[O que são? 4](#_Toc326852200)

[Para que seve? 4](#_Toc326852201)

[Características gerais 4](#_Toc326852202)

[Roleta – Seleção 5](#_Toc326852203)

[Uniforme –Crossover –Recombinação 8](#_Toc326852204)

[Esquema 1 8](#_Toc326852205)

[Esquema 2 8](#_Toc326852206)

[Observação /Problema 9](#_Toc326852207)

[Operador Truncation – Substituição 9](#_Toc326852208)

[Mochila :Problema 10](#_Toc326852209)

[Reparação: aleatório 11](#_Toc326852210)

[Anexos 12](#_Toc326852211)

[Reparação:Psenda - Aleatória 13](#_Toc326852212)

[Problema do caixeiro viajante 14](#_Toc326852213)

[Descrição 14](#_Toc326852214)

[Esquemas: 14](#_Toc326852215)

[Cálculo do Fitness: 15](#_Toc326852216)

[Reparar /Mitigar Programa 16](#_Toc326852217)

[Order Crossover 17](#_Toc326852218)

[Descrição 17](#_Toc326852219)

[Esquema 17](#_Toc326852220)

[SUS – Problema de Minimização 19](#_Toc326852221)

[Esquema 19](#_Toc326852222)

[Comparação de valores: 19](#_Toc326852223)

[SwapGenes: Mutação 20](#_Toc326852224)

[Descrição 20](#_Toc326852225)

[Esquemas 20](#_Toc326852226)

[Anexos 21](#_Toc326852227)

[Função de avaliação 22](#_Toc326852228)

[Valor do indivíduo 22](#_Toc326852229)

[Seleção 22](#_Toc326852230)

[Recombinação 23](#_Toc326852231)

[Mutação 23](#_Toc326852232)

[Intermediate recombination 24](#_Toc326852233)

[Anexos 26](#_Toc326852234)

[Fluxogramas 27](#_Toc326852235)

[Fitness Penalização 27](#_Toc326852236)

[Reparação Aleatória 27](#_Toc326852237)

[Reparação Pseudo Aleatória 28](#_Toc326852238)

[Operador Roleta 28](#_Toc326852239)

[Operador Crossover 29](#_Toc326852240)

[Operador SUS 30](#_Toc326852241)

[Operador Truncation 30](#_Toc326852242)

[Crossover 31](#_Toc326852243)

[Invertion Mutation 32](#_Toc326852244)

[Mutation 33](#_Toc326852245)

[PMX 34](#_Toc326852246)

[Crossover 35](#_Toc326852247)

[Mutação Binaria 36](#_Toc326852248)

[P.S.O 37](#_Toc326852249)

[Recombinação Binaria 38](#_Toc326852250)

[Pseudo-codigo 39](#_Toc326852251)

[Fitness Penalização 39](#_Toc326852252)

[Reparação aleatória 39](#_Toc326852253)

[Reparação aleatória 40](#_Toc326852254)

[Operador crossover 40](#_Toc326852255)

[Operador Roleta 41](#_Toc326852256)

[Operador SUS 41](#_Toc326852257)

[Operador Truncation 42](#_Toc326852258)

[Crossover 42](#_Toc326852259)

[Invertion Mutation 43](#_Toc326852260)

[Mutation 43](#_Toc326852261)

[PMX 44](#_Toc326852262)

[SUS Minimização 44](#_Toc326852263)

[Crossover 45](#_Toc326852264)

[Mutação binaria 45](#_Toc326852265)

[PSO 45](#_Toc326852266)

[Recombinacao\_binaria 46](#_Toc326852267)

# Resumo

Desenvolveu-se neste trabalho a implementação computacional de vários algoritmos genéticos que agem sobre uma população através de diversos operadores, de forma a criar novas populações, até se obter uma populacão próxima da popuação “ideal”.

Estes algoritmos correm diversas vezes de forma sucessiva, sendo escolhidos os melhores indivíduos para fazerem parte da nova população.

O operador de *seleção* foi implementado em dois algoritmos básicos: *roleta* e *torneio*.

Na roleta, cada indivíduo tem uma determinada chance de ser escolhido, tendo como base a sua qualidade.

No torneio, é feito um torneio para escolher os melhores indivíduos da população.

## O que são?

* Os Algoritmos Genéticos são uma classe de procedimentos, com passos distintos bem definidos.
* Essa classe se fundamenta em analogias a conceitos biológicos já testadas à exaustão.
* Cada passo distinto pode ter diversas versões diferentes

## Para que seve?

* Busca e Otimização
* Amplamente utilizados, com sucesso, em problemas de difícil manipulação pelas técnicas tradicionais
* Eficiência X Flexibilidade

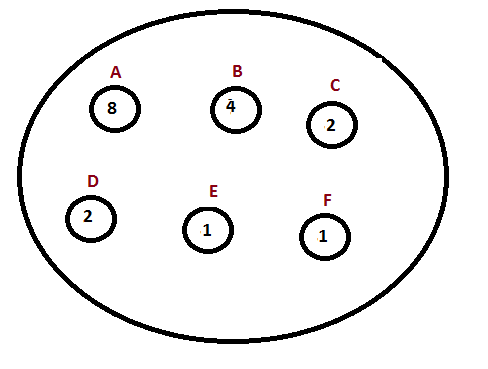
## Características gerais

* Utilizam uma codificação do conjunto de
* Parâmetros (indivíduos )e não comos próprios parâmetros(estados );
* Vasculham várias regiões do espaço de busca de cada vez;
* Utilizam informações diretas de qualidade ,em contraste com as derivadas utilizadas nos métodos tradicionais de otimização;
* Utilizam regras de transição probabilísticas e não regras determinísticas

## Roleta – Seleção

1. Atribuir uma percentagem a cada individuo de uma população com base no fitness e no total de fitness da população ;
2. Juntar os indivíduos em linha de forma a criar percentagens acumuladas, com base na ordem dos indivíduos;
3. Gerar um numero real aleatório entre 0 e1
4. Selecionam o individuo para onde o número aponta;
5. Repetir os passos 3º e 4º ate ter o número de indivíduos pretendidos.

Exemplo: Selecionam 4 indivíduos numa população de 6 indivíduos.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |

|  |  |  |  |
| --- | --- | --- | --- |
| 0,2 | 0,1 | 0,69 | 0,5 |
| A | A | C | B |

1. Juntos todos os indivíduos de uma população pela ordem que se encontram, de forma a fazer uma linha com os indivíduos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Indi A** | **Indi B** | **Indi C** | **Indi D** | … **N Individuos** |

1. Fazer a soma do fitness do individuo anterior ao ser, com base na ordem em que se encontram na linha;
2. Obter o total do somatório de todos os fitness, o fitness do ultimo indivíduo, pois é o acumulado de todos os indivíduos;
3. Gerar um ponto aleatório, inteiro ou real, dentro do intervalo 0 e total do fitness de todos os indivíduos, para ser o nosso ponto de partida;
4. Definir qual vai ser o offset que se vai acrescentar ao ponto de partido. Calcula através

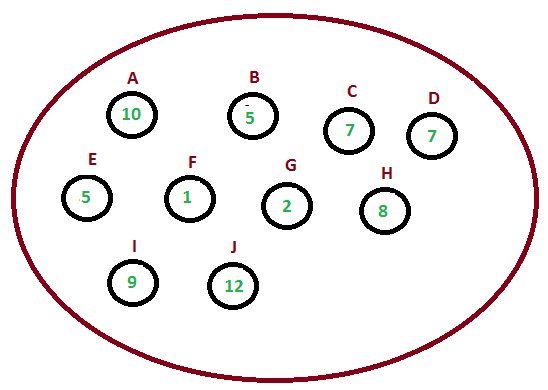
Total fitness

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

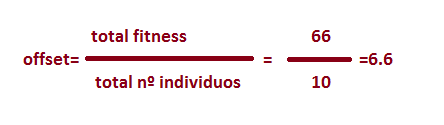
Total nº indivíduos

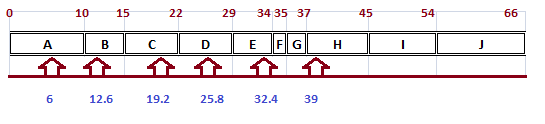
1. Selecionar o individuo para onde o ponto aponta;
2. Somar ao ponto o offset;
3. Repetir os pontos 6º e 7º ate obter o numero de indivíduos selecionados pretendidos;
4. Caso se atinja o fim da linha de indivíduos então volta se ao inicio da linha

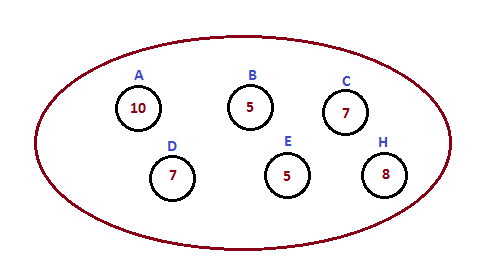
Exemplo: população 10 indivíduos e devemos escolher 6 indivíduos para reprodução



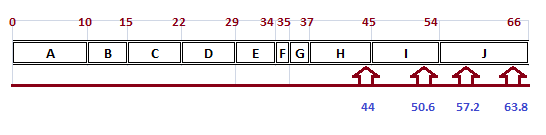
***População***

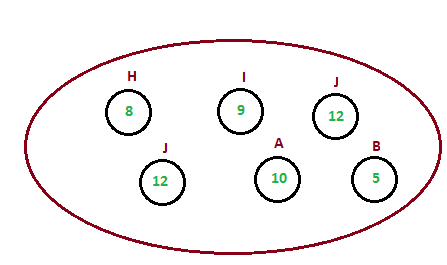






**Indivíduos selecionados**



**Indivíduos Selecionados**

# Uniform–Crossover – Recombinação

1. Ter dois indivíduos, um “pai” e uma “mãe ”
2. Gerar uma cadeia de bits, ou mascara, para determinar quais os bits que não ser traçados;

Nota: A mascara tem que ter as mesmas dimensões do gene dos indivíduos pais

1. Para cada bit a “1” na mascara vai haver uma troca

Exemplo: Dois pais com gemes de tamanho de 10 bits e gerar dois filhos usando o operador uniform – crossover .

### Esquema 1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pai | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  | pais |
| Mãe | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | mascara |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Filho1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  | filhos |
| Filho2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |  |

### Esquema 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pai | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  | pais |
| Mãe | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | mascara |
|  |  |  |  |  |  |  |  |  |  |  |  |
| filho | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |  |
|  |  |  |  |  |  |  |  |  |  |  | filhos |
| filha | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |  |

## Observação /Problema

Se a mascara for toda a “0” o indivíduos filho vai ser igual ao pai e o individuo filha vai ser igual a mãe.

Solução

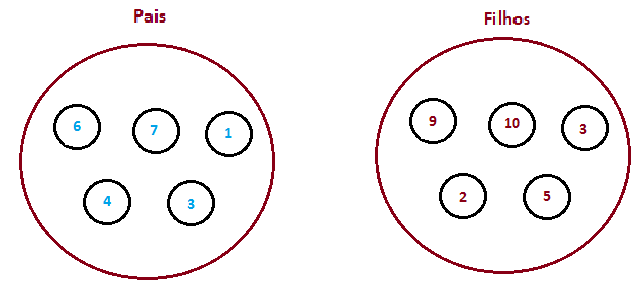
Garantir que 50% dos bits da mascara são a “1” e os restantes a “0”.

Os 50% referidos anterior mente pode ser um parâmetro deste operada, sendo 50% o valor recomendado por defeito

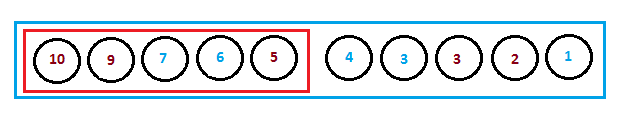
## Operador Truncation – Substituição

1. Ter duas populações, ou mais para aplicar o operador;
2. Juntar os varias populações numa só
3. Ordenar, com base no fitness de cada individuo, de forma descendente
4. Selecionar os indivíduos que surgem primeiro e criar uma nova população

Exemplo: Duas populações com 5 indivíduos cada, onde se quer criar uma nova população com 5 indivíduos



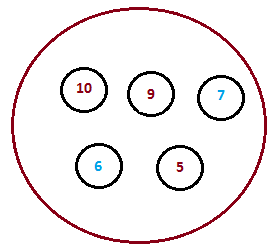
Indivíduos ordenados

 Os indivíduos das

duas populações

juntos e ordenados

Os 5 primeiros indivíduos



Nova os 5 indivíduos

População com melhor fitness

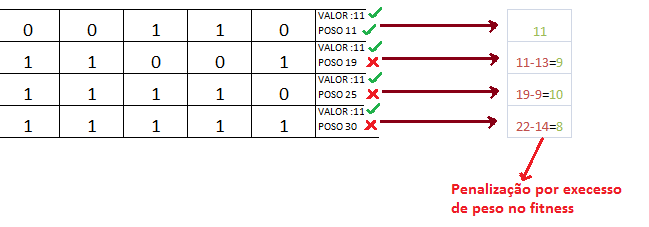
selecionado

# Mochila :Problema

Problema: levar o maior número de peças na mochila, que tem um limite de peso, mas ao mesmo tempo levar o maior valor na mochila.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Peso | 10 | 4 | 1 | 10 | 5 |
| Valor | 5 | 3 | 10 | 1 | 3 |

Máximo peso possível na mochila: 16

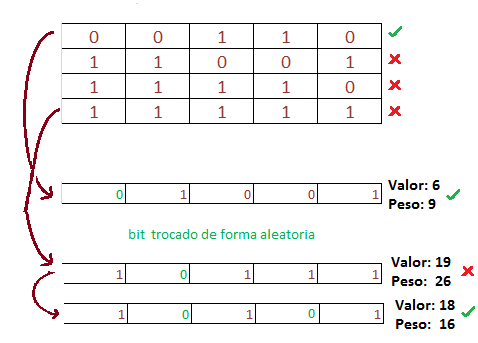


Tipos de penalizações

Linear: -31=3 (usada no exemplo anterior)

Quadrática: -32=9

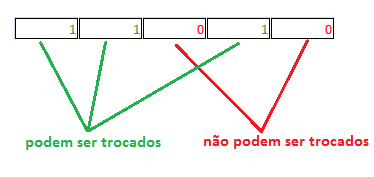
### Reparação: aleatório



Repetir o processo ate ter um peso aceitável

Só troca os bits que estão a “1”, pois são esses que estão a fazer a indivíduos / mochila ter peso a mais.

Exemplo: dos bits que podem ser trocados na reparação:



## [Anexos](file:///C:\Users\mendes\Desktop\relat\2\inovative\versao02)

[Mochila Fitness Penalização](versao02/Mochila%20FitnessPenalizacao.jpg)

[Mochila Reparacao Aleatoria](versao02/Mochila%20ReparacaoAleatoria.jpg)

[Mochila Reparacao Pseudo Aleatoria](versao02/Mochila%20ReparacaoPseudoAleatoria.jpg)

[Mochila-part1](versao02/Mochila-part1.jpg)

[Mochila-part2](versao02/Mochila-part2.jpg)

[Mochila-part3](versao02/Mochila-part3.jpg)

[Mochila-part4](versao02/Mochila-part4.jpg)

[Operador Roleta](versao02/Operador%20Roleta.jpg)

[Operador SUS](versao02/Operador%20SUS.jpg)

[Operador Truncation](versao02/Operador%20Truncation.jpg)

[Operator Truncation](versao02/OperatorTruncation.jpg)

[pseudo-codigo\_mochila fitnessPenalizacao](versao02/pseudo-codigo_mochila%20fitnessPenalizacao.txt)

[pseudo-codigo\_mochila reparacao aleatoria](versao02/pseudo-codigo_mochila%20reparacao%20aleatoria.txt)

[pseudo-codigo\_mochila reparacao pseudo-aleatoria](versao02/pseudo-codigo_mochila%20reparacao%20pseudo-aleatoria.txt)

[pseudo-codigo\_Operador crossover](versao02/pseudo-codigo_Operador%20crossover.txt)

[pseudo-codigo\_Operador Roleta](versao02/pseudo-codigo_Operador%20Roleta.txt)

[pseudo-codigo\_Operador SUS](versao02)

[pseudo-codigo\_Operador Truncation](versao02/pseudo-codigo_Operador%20Truncation.txt)

[Relatorio \_Final](versao02/Relatorio%20_Final.docx)

[Roleta](versao02/Roleta.jpg)

[SUS-part1](versao02/SUS-part1.jpg)

[SUS-part2](versao02/SUS-part2.jpg)

[UniformCrossover-part1](versao02/UniformCrossover-part1.jpg)

[UniformeCrossover-part2](versao02/UniformeCrossover-part2.jpg)

[Esquemas 1](versao02/v0.2_01.jpg)

[Esquemas 2](versao02/v0.2_02.jpg)

[Esquemas 3](versao02/v0.2_03.jpg)

[Esquemas 4](versao02/v0.2_04.jpg)

[Esquemas 5](versao02/v0.2_05.jpg)

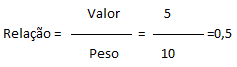
[Esquemas 6](versao02/v0.2_06.jpg)

[Versao2](versao02/Versao2.vpp)

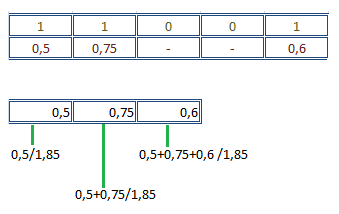
# Reparação:Psenda - Aleatória

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Peso | 10 | 4 | 1 | 10 | 5 |
| Valor | 5 | 3 | 10 | 1 | 3 |
| Valor/Peso | 0,5 | 0,75 | 10 | 0,1 | 0,6 |

Calculo de Relação



Exemplo:



Atenção: Não podemos

usar dessa forma tão

direta porque as peças

com melhor relação

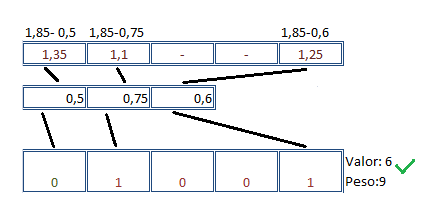
Valor /peso ficam com a

maior probabilidade de

serem escolhidos para

saírem da mochila.

Então:



# Problema do caixeiro viajante

Descrição**:**

Problema de complexidade n!. Procura-se encontrar um, ou mais caminhos, para percorrer todas as cidades e voltar ao ponto de origem, sem nunca passar numa cidade duas vezes e obter o caminho com menos custo.

### Esquemas:

* Tamanho de gene igual ao numero de cidades;
* As cidades são indexadas para que as suas representações sujam números inteiros;
* As ordem com que os alelos aparecem no gene n2 a mesma ordem por qual passamos nas cidades.

**Representação Caminhos Custo**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** |
| **A** | **-** | **2** | **4** | **1** |
| **B** | **2** | **-** | **1** | **5** |
| **C** | **4** | **1** | **-** | **3** |
| **D** | **1** | **5** | **3** | **-** |

2

4

5

3

**Representação Gene**

|  |  |  |  |
| --- | --- | --- | --- |
| **0** | **1** | **2** | **3** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cidade | A | B | C | D |
| Índex | 0 | 1 | 2 | 3 |

Ordem:

Cidade A

Cidade B

Cidade C

Cidade D

Cidade A

## Cálculo do Fitness:

* Exemplo 1

A

B

C

D

A

2

1

3

1

+

+

+

=

7

Fitness

Representação do caminho

* Exemplo 2

D

B

A

C

D

5

2

4

3

+

+

+

=

14

Fitness

Representação do caminho

**São sempre o mesmo caminho,**

**começam é em pontos diferentes**

## Reparar /Mitigar Programa

**Pegar na cidade e por como primeira cidade através de uma rotação, para o mesmo caminho só ter uma representação.**

**Transforma-se numa única**

**representação**

**fazemos isso, só partir o gene em dois no ponto onde aparece o , depois criamos um novo gene com a( 2º parte) em primeiro e a ( 1º parte) em segundo, e assim fica o sempre como cidade de partida.**

# Order Crossover

### Descrição

Uma parte do pai é mapeado para uma porção da mãe. A partir da porção substituída, o resto dos genes são preenchidos mas omitindo os genes já presentes e respeitando a ordem com que eles se encontram.

## Esquema

**Pai**

**Ordem pela qual se lê quando**

**se chaga ao fim do gene**

**Mãe**

**Divisão feita num ponto Aleatório**

**Os genes do lado esquerdo são**

**iguais aos dos pais**

**Filho**

**Filha**

**Não podem ser escolhidos logo escolhe se os**

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **3** | **0** | **1** |

**porque já existe genes com que ainda não**

**esses valores aparecem no individuo**

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **3** | **0** | **2** |

**Código**

Entra (Individuo pai, mae Individuo)

dim: = TAMANHO (pai.genes [])

corte: = aleatorio ([1; dim-1])

Pará i: = 0 comeu i <corte

filho.gene [i]: = pai.gene [i]

filha.gene [i]: = mae.gene [i]

i: = i +1

n J: = corte comeu j <dim

filho.gene [j]: = Procura (j, mae.gene [], filho.gene [])

filha.gene [j]: = Procura (j, pai.gene [], filha.gene [])

j: = j +1

sai (Individuo Filho, Individuo FILHA)

Procura (pos, progenitor.gene [], descendente.gene [])

p: = pos-1

q: = pos

enquanto (p> = 0)

se (descendente.gene [p] = progenitor.gene [q]) entao

q: = q +1

q: = q% TAMANHO (progenitor.gene [])

p: = pos

p: p-1 =

Retorna progenitor.gene [q]

# SUS – Problema de Minimização

## Esquema

População

**I**

**J**

**H**

**G**

**F**

**E**

**D**

**A**

**C**

**B**

Max(população) = 12

Min(população) = 1

Pseudo-código: Cálculo do novo Fitness:

inteiro converterFitness (população, fitness)

MaxFitness := Max(população) + 1

return MaxFitness – fitness

## Comparação de valores:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I | J |
| Fitness | 10 | 5 | 7 | 7 | 5 | 1 | 2 | 8 | 9 | 12 |
| Novo Fitness | 3 | 8 | 6 | 6 | 8 | 12 | 11 | 5 | 4 | 1 |

Como se pode ver pelos valores da tabela anterior, os valores originais com maior fitness, passam a ser os que têm menor fitness neste momento e os que tinham menor, passam a ter os maiores valores de fitness.

Antes

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J |
|  |  |  |  |  |  |  |  |  | 66 |

Depois

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J |
|  |  |  |  |  |  | 64 | | | |

Nota: Não esquecer que neste SUS procuramos o valor mais pequeno de fitness, mas o SUS “não devolve os valores mais pequenos”, logo tivemos que transformar os valores grandes em valores pequenos e fazer o contrário com os valores pequenos.

# SwapGenes: Mutação

## Descrição

**Temos um individuo onde escolhemos dois delos , de forma aleatória, e os trocamos de sitio.**

## Esquemas

**Dois pontos aleatórios que apontam para**

**Dois deles**

**Depois é só fazer o troca de posições**

**Código**

Entra (População individuos [])

dimPop: = TAMANHO (individuos [])

dimGene: = TAMANHO (. individuos [] genes [])

Pará i: = 0 i = comeu dimPop

alelo1: = aleatorio ([0; dimGene [)

Fazer

alelo2: = aleatorio ([0; dimGene [)

enquanto (alelo1 = alelo2)

aleloTmp:. = individuos [i] genes [alelo1]

individuos [i] genes [alelo1]:. individuos. = [i] genes [alelo2]

individuos [i] genes [alelo2]:. = aleloTmp

i: = i +1

sai (População individuos [])

## [Anexos](versao03)

[**DemoTSP**](versao03/DemoTSP)

[**TSP**](versao03/TSP)

[**Apresentação Caixeiro Viajante**](versao03/CaixeiroViajante_apresentacao.pdf)

[**CaxeiroViajante-Part1**](versao03/CaxeiroViajante-Part1.jpg)

[**CaxeiroViajante-Part2**](versao03/CaxeiroViajante-Part2.jpg)

[**CaxeiroViajante-Part3**](versao03/CaxeiroViajante-Part3.jpg)

[**cycleCrossover(fluxograma)**](versao03/cycleCrossover(fluxograma).jpg)

[**cycleCrossover**](versao03/cycleCrossover.jpg)

[**pseudo-codigo cycleCrossover**](versao03/cycleCrossover.txt)

[**InversionMutation**](versao03/InversionMutation.jpg)

[**InvertionMutation**](versao03/InvertionMutation.jpg)

[**pseudo-codigo InvertionMutation**](versao03/InvertionMutation.txt)

[**Mutation**](versao03/Mutation.jpg)

[**pseudo-codigo Mutation**](versao03/Mutation.txt)

[**PMX**](versao03/PMX.jpg)

[**PMX**](versao03/PMX.txt)

[**PMX\_1**](versao03/PMX_1.jpg)

[**PMX\_2**](versao03/PMX_2.jpg)

[**Relatorio\_Final**](versao03/Relatorio_Final.docx)

[**SUS Minimizacao**](versao03/SUS%20Minimizacao.jpg)

[**pseudo-codigo SUS\_Minimizacao**](versao03/SUS_Minimizacao.txt)

[**SUS\_Minimizacao-Part1**](versao03/SUS_Minimizacao-Part1.jpg)

[**SUS\_Minimizacao-Part2**](versao03/SUS_Minimizacao-Part2.jpg)

[**SwapGenes fluxograma**](versao03/SwapGenes%20fluxograma.jpg)

[**Swapgenes**](versao03/Swapgenes.jpg)

[**pseudo-codigo Swapgenes**](versao03/Swapgenes.txt)

[**Versao3**](versao03/Versao3.vpp)

# Função de avaliação

f(x1,x2)=21.5 + x1 m(4π\*x1)+x2 m(20 π\*x2)

com, -3,0 ≤ x1 ≤12,1 e 4,1 ≤ x2 ≤5,8

Representação binaria para 4 casas decimais :

Dx1=15,1=12,1 –(-3,0)

15,1 \*10 000 = 15100 217 ≤ 151000 ≤ 218

Dx2=1,7=5,8 – 4,1

1,7 \*10 000 =17000 214 ≤ 17000 ≤ 215

Indivíduo i é composto de 2 genes (x1 e x2)

X1 =18 alelos

X2 =15 alelos

Exemplo:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **0** |

X1= =250552d

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |  |  |

X2= =4800d

X´1=

X´2=

## Valor do indivíduo

Avaliação =f(11,4323 ; 4,3490) = 13,1733

## Seleção

Ordena-se a população (por exemplo: através da roleta) colocando neste caso (maximização ) no topo dar lista os indivíduos que obtiverem maior valor calculado através da função de avaliação.

## Recombinação

1. Escolhem-se 2 indivíduos do topo da lista obtida (que ainda não tivessem sido escolhidos)
2. Calcula-se um numero aleatório entre [0;1[

3.1 Se o numero aleatório por maior do que 0,65, os dois indivíduos escolhidos passam diretamente para a descendência.

3.2 se o número aleatório por menor ou igual a 0.65,calcula-se aleatoriamente um ponto de corte para os primeiros genes dos dois indivíduos escolhidos recombinando-os e procedendo de igual forma para os segundos genes de ambos. Passando para a discordância os dois indivíduos cujos novos genes são constituídos até aos cortes, pelos seus próprios e são contas em diante pelos do indivíduos com o qual recombinaram.

4 volta-se ao ponto 1 até terminassem os indivíduos da lista ordenada.

Exemplo gene x1=a+b

Pontos de corte aleatória

gene x2=c+d

Indivíduos:i1 e i2

i1= x1 & x2 ⬄ i1= a+b & c+d

i2= x1 & x2 ⬄ i1= A+B & C+D

Recombinando fica:

i´1 =a +B & c +D

i´2 =A +b & C +d

## Mutação

1. É escolhida uma percentagem de mutação (por exemplo 1%)
2. É escolhido um individuo
3. É escolhido o primeiro gene: e para cada um dos alelos é calculada aleatoriamente uma percentagem, se essa percentagem for menos do que 1% o bit em causa passa a “1”.

3.1 É efetuada a mesma operação no segundo gene .

1. Volta-se ao ponto 2 até terminar a população.

Exemplo:

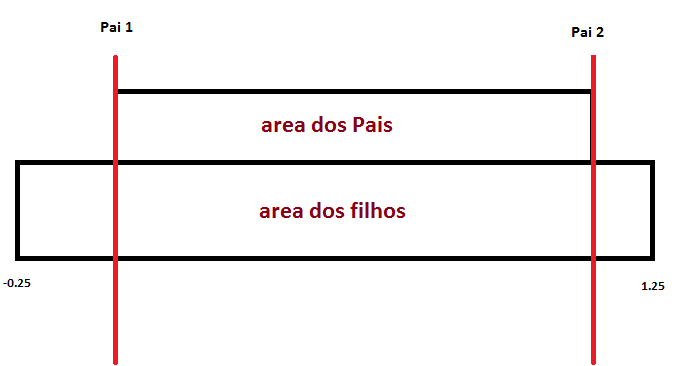
i´3= 101101101000111010 & 111001000101110

# Intermediate recombination

1. Selecionam-se 2 indivíduos, P1 e P2, assumindo o papel de “Pais” (P1 “Pai1” )
2. Selecionam-se os genes GenesiP1 e GenesiP2 , dos indivíduos P1 e P2 respetivamente , em que i (1,2,3…. numero genes)
3. Gera-se um número aleatório a; pertencente ao intervalo [ -0.25 ; 1.25 ]
4. Calcula-se o gene GenesiF do novo indivíduo F “filho ”, através da seguinte formula.

GenesiF = GenesiP1 \* ai+ GenesiP2 \* ai

1. Volta-se ao ponto 2 até que todos os genes dos indivíduos ” Pais ” P1 e P2 sejam percorridos (i=numero genes ).



Exemplo

Considerar os seguintes indivíduos, com 3 genes cada

|  |  |  |
| --- | --- | --- |
| **12** | **25** | **5** |

Pai 1

|  |  |  |
| --- | --- | --- |
| **123** | **4** | **34** |

Pai 2

Genes1P2 Genes2P2  Genes3P2

Considerar os seguintes valores de a para esta exemplo

a1  a2 a3

0.5 1.1 -0.1

O novo filho calculado

Genes1F Genes2F  Genes3F

|  |  |  |
| --- | --- | --- |
| **67.5** | **1.9** | **2.1** |

Pai 2

GenesiF = GenesiP1 \* ai+ GenesiP2 \* (1- ai)

=12 \* 0.5 + 123 \* (1-0.5)

= 6 + 61.5

=67.5

## [Anexos](file:///C:\Users\mendes\Desktop\relat\2\inovative\versao03)

[Demo](versao04/Demo)

[GeneticAlgorithm\_real\_coded](versao04/GeneticAlgorithm_real_coded)

[PSO\_DE\_GA](versao04/PSO_DE_GA)

[Differential Evolution](versao04/Differential%20Evolution.pdf)

[intermediate\_crossover](versao04/intermediate_crossover.txt)

[intermediate\_crossover\_parte1](versao04/intermediate_crossover_parte1.jpg)

[intermediate\_crossover\_parte2](versao04/intermediate_crossover_parte2.jpg)

[IntermediateCrossover](versao04/IntermediateCrossover.jpg)

[Mutation Binaria](versao04/Mutation%20Binaria.jpg)

[optimizacao\_funcaoMat\_PDF](versao04/optimizacao_funcaoMat_PDF.pdf)

[pseudo-codigo](versao04/Mutation_binaria.txt) [Mutation\_binaria](versao04/Mutation_binaria.txt)

[P.S.O](versao04/P.S.O.jpg)

[PSO](versao04/PSO.pdf)

[pseudo-codigo PSO](versao04/PSO.txt)

[Recombinacao Binaria](versao04/Recombinacao%20Binaria.jpg)

[pseudo-codigo Recombinacao\_binaria](versao04/Recombinacao_binaria.txt)

[Relatorio\_Final4](versao04/Relatorio_Final4.docx)

[Esquemas 1](versao04/Scan_Pic0001.jpg)

[Esquemas 2](versao04/Scan_Pic0002.jpg)

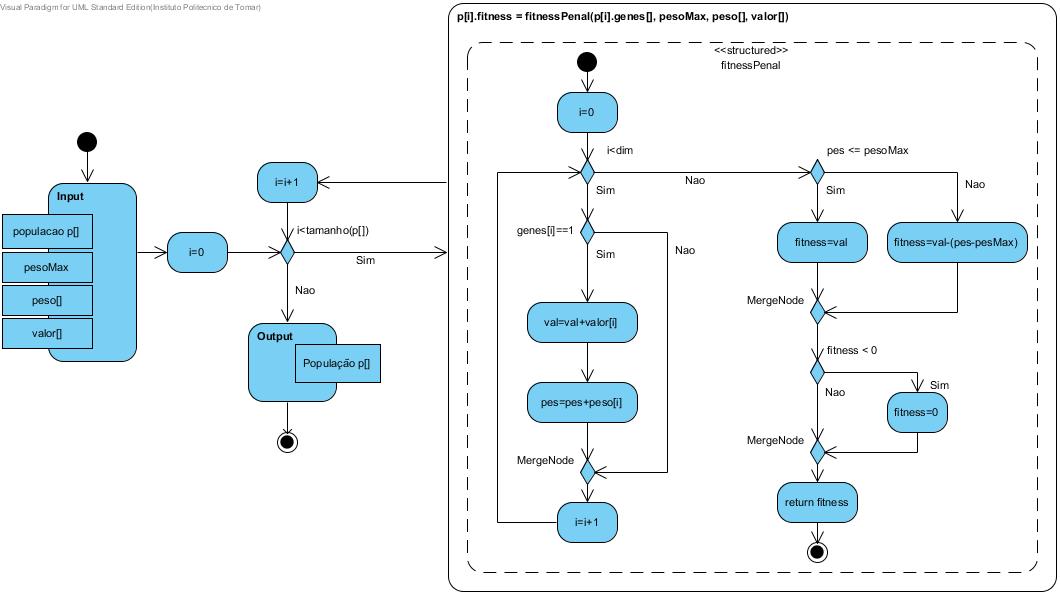
[Esquemas 3](versao04/Scan_Pic0003.jpg)

[solucaoDemo](versao04/versao0_4.pptx)

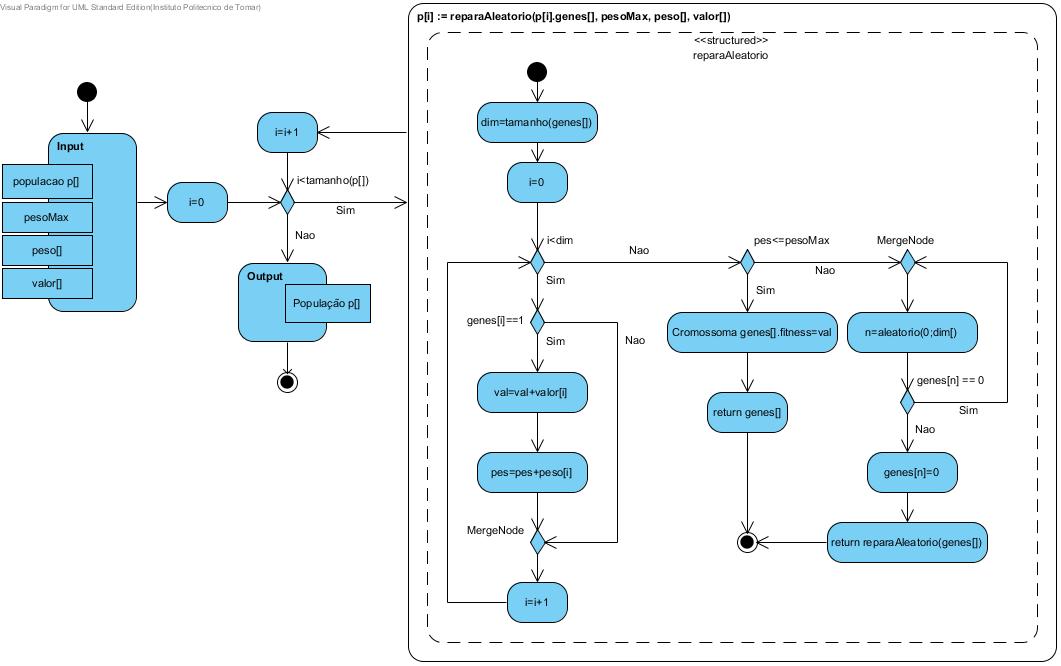
[Versao4](versao04/Versao4.vpp)

# Fluxogramas

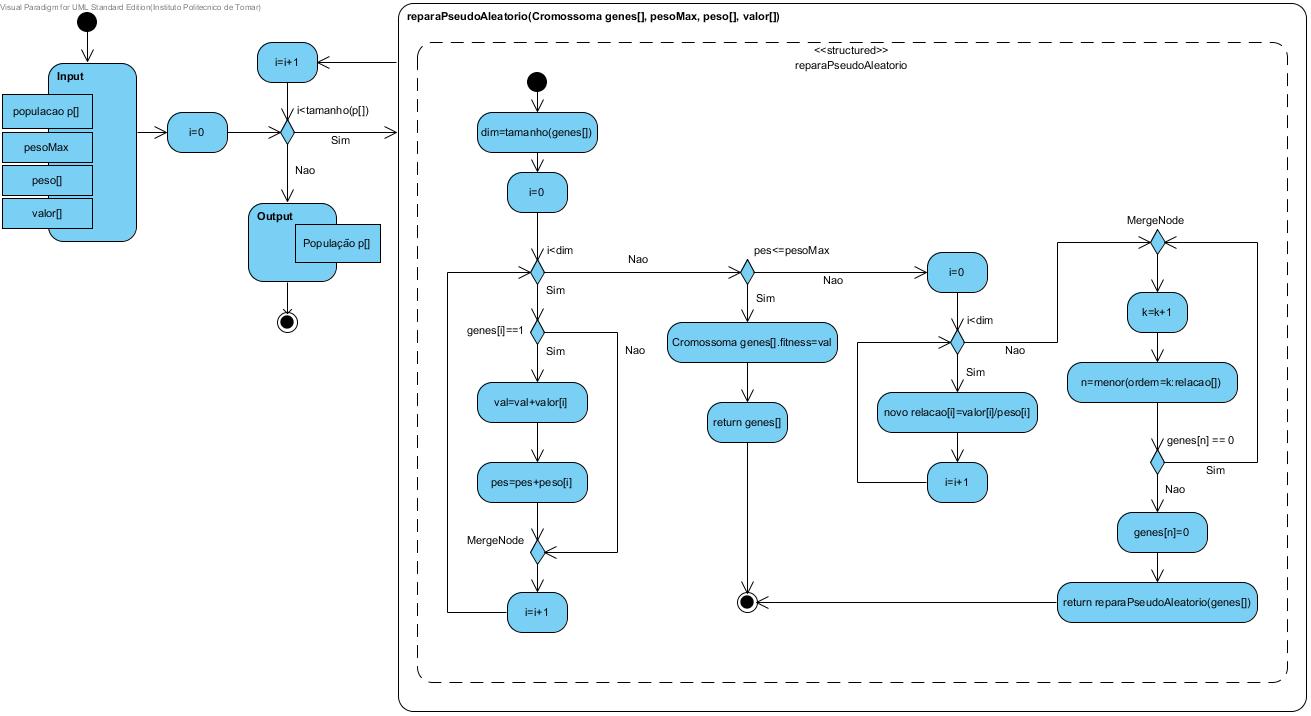
## Fitness Penalização



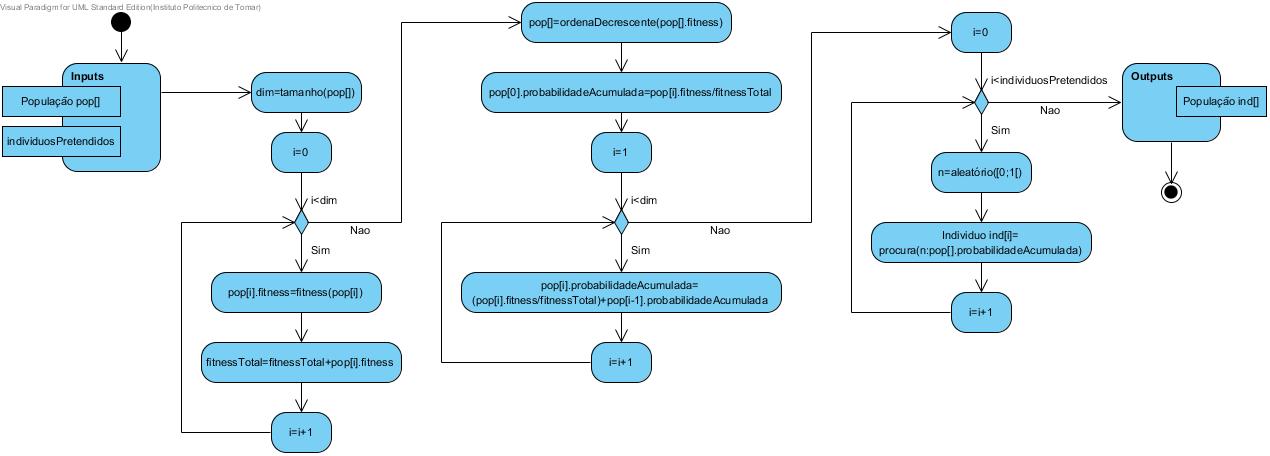
## Reparação Aleatória



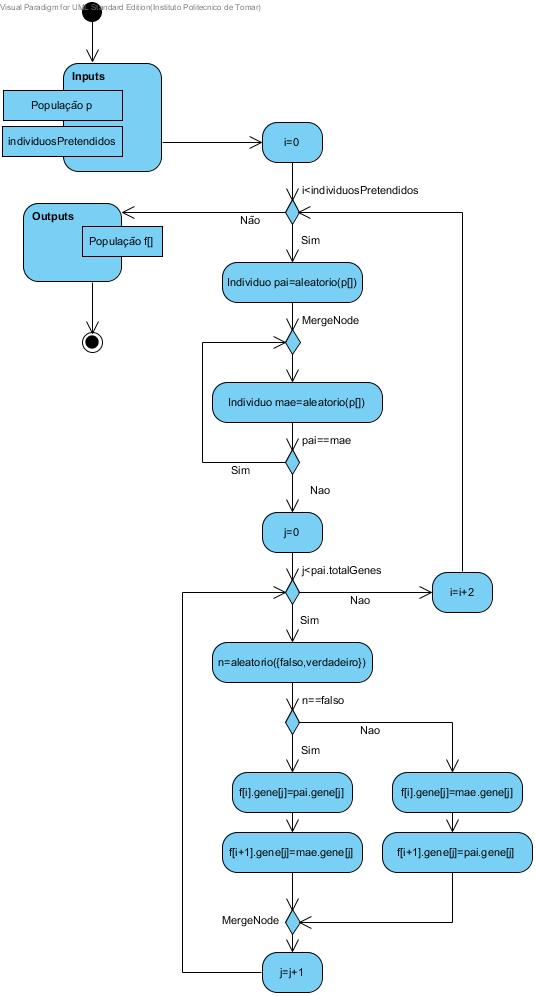
## Reparação Pseudo Aleatória



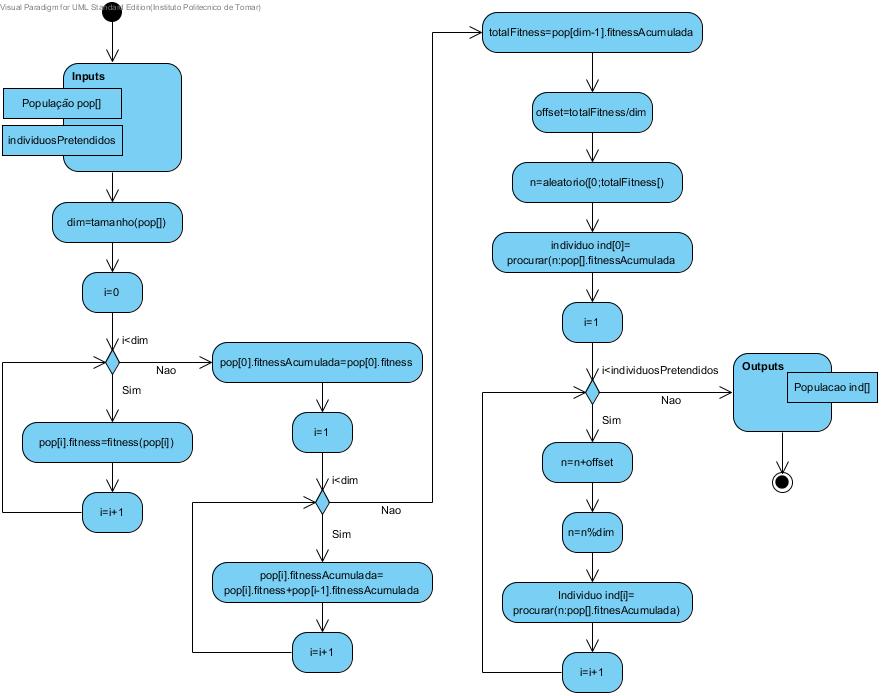
## Operador Roleta



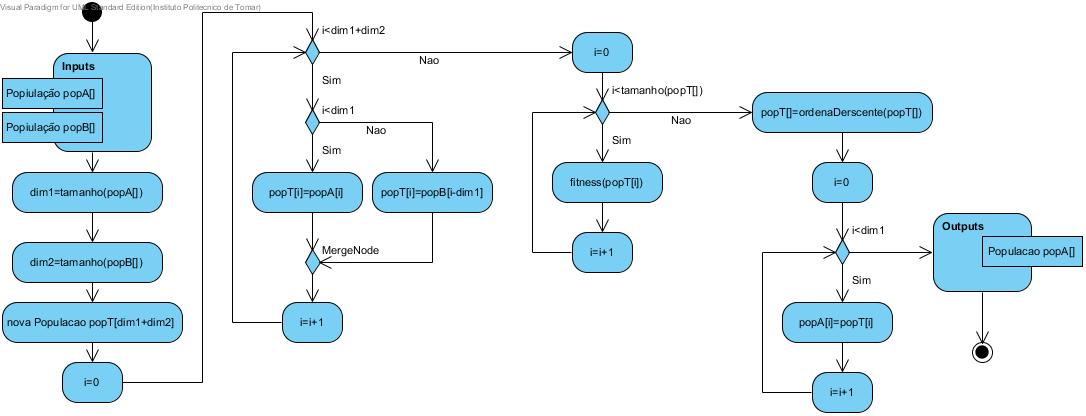
## Operador Crossover



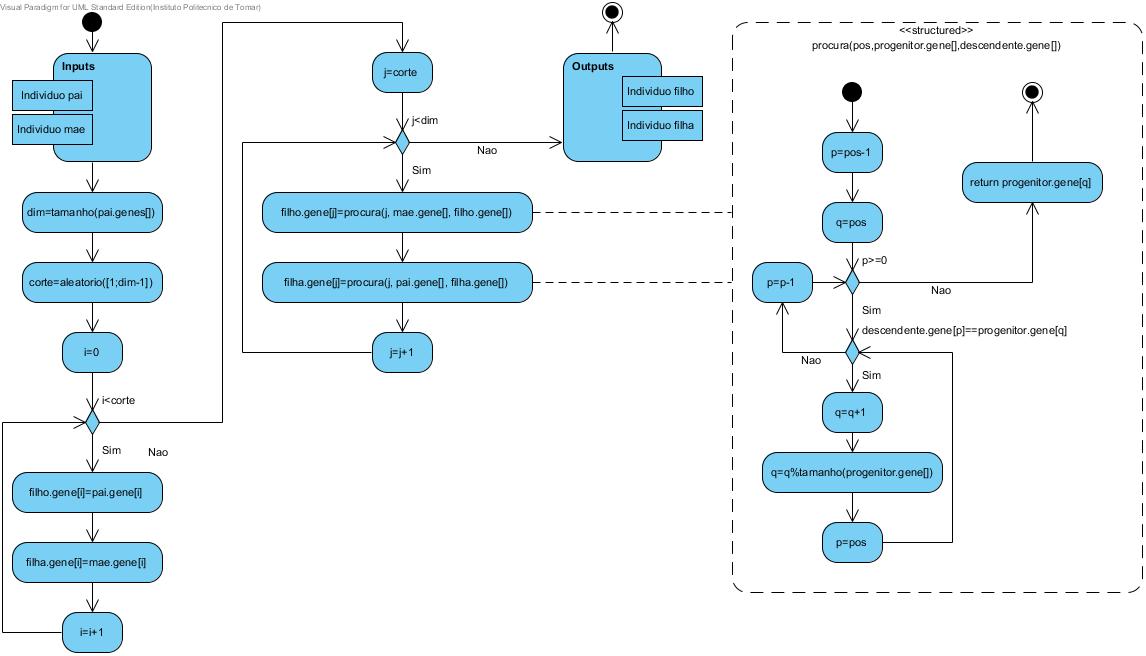
## Operador SUS



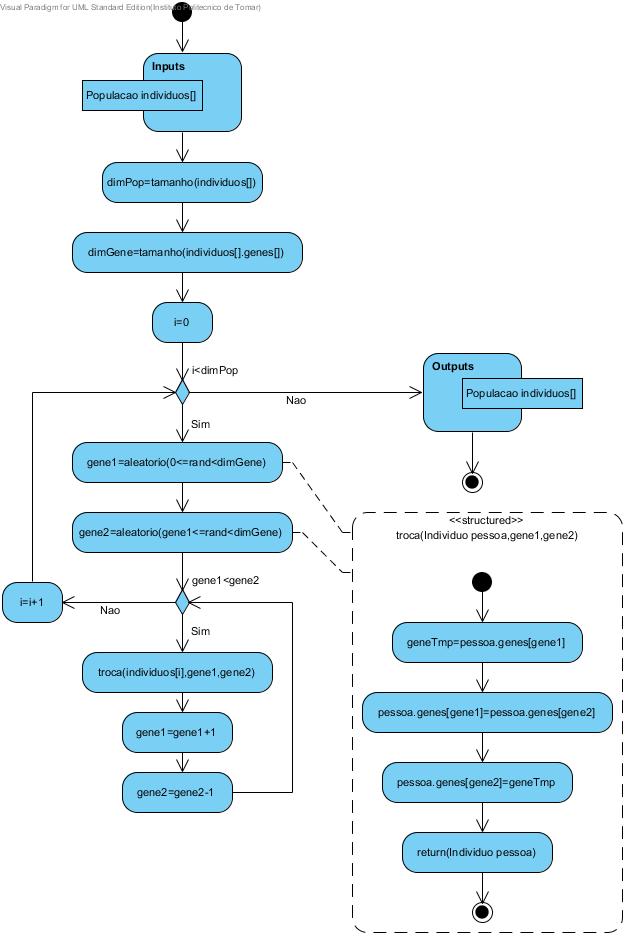
## Operador Truncation



## Crossover

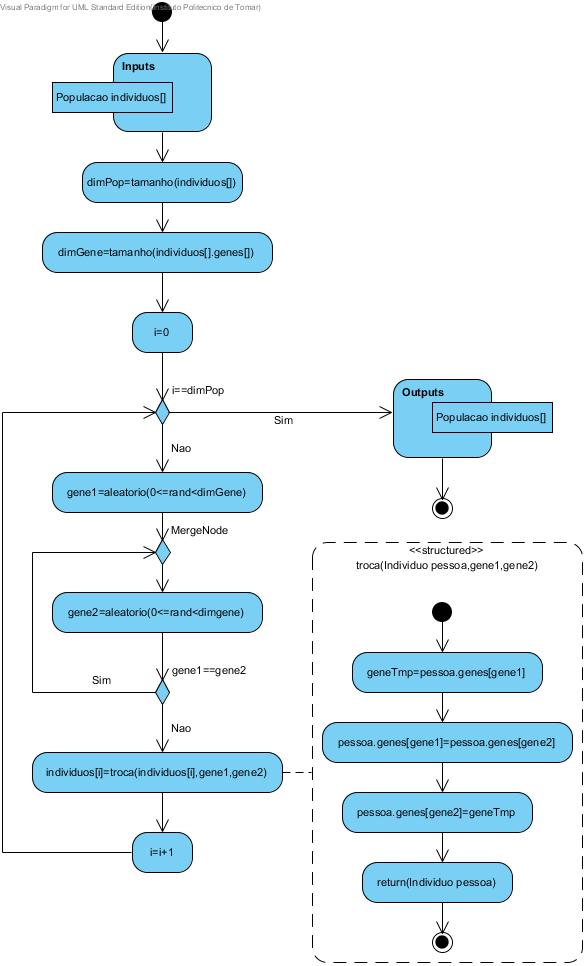
****

## Invertion Mutation

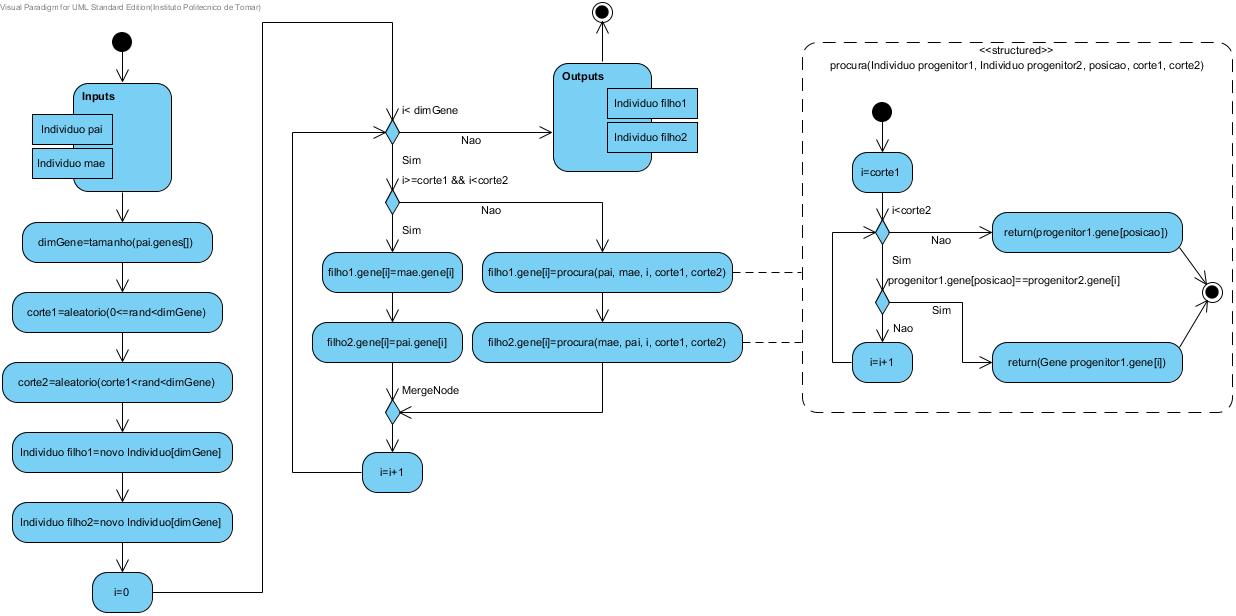
****

**.**

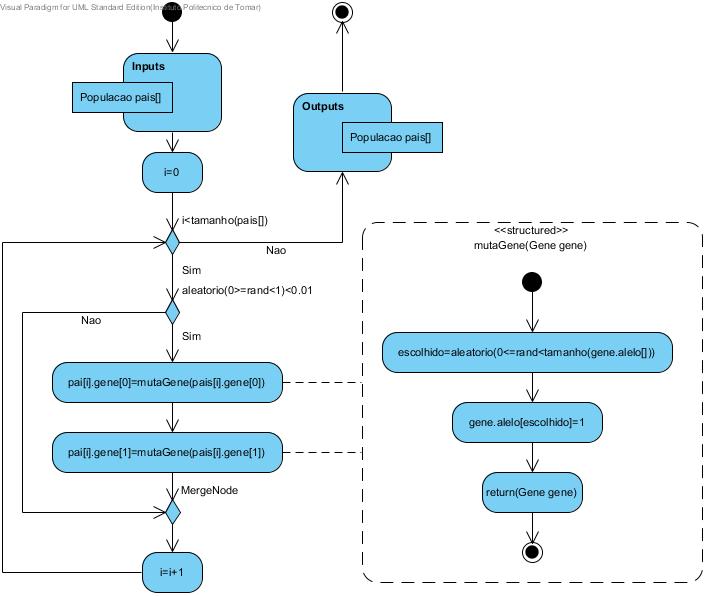
## Mutation

****

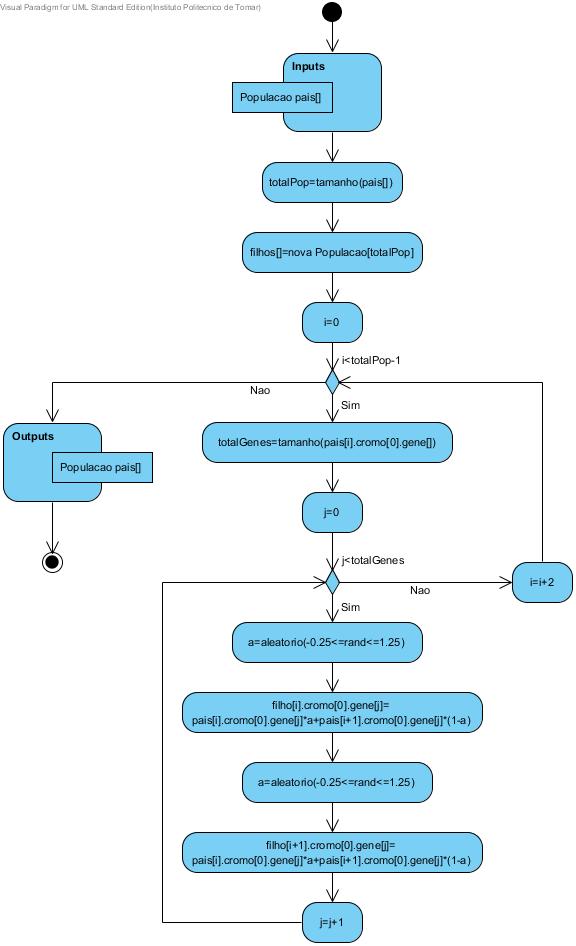
## PMX

****

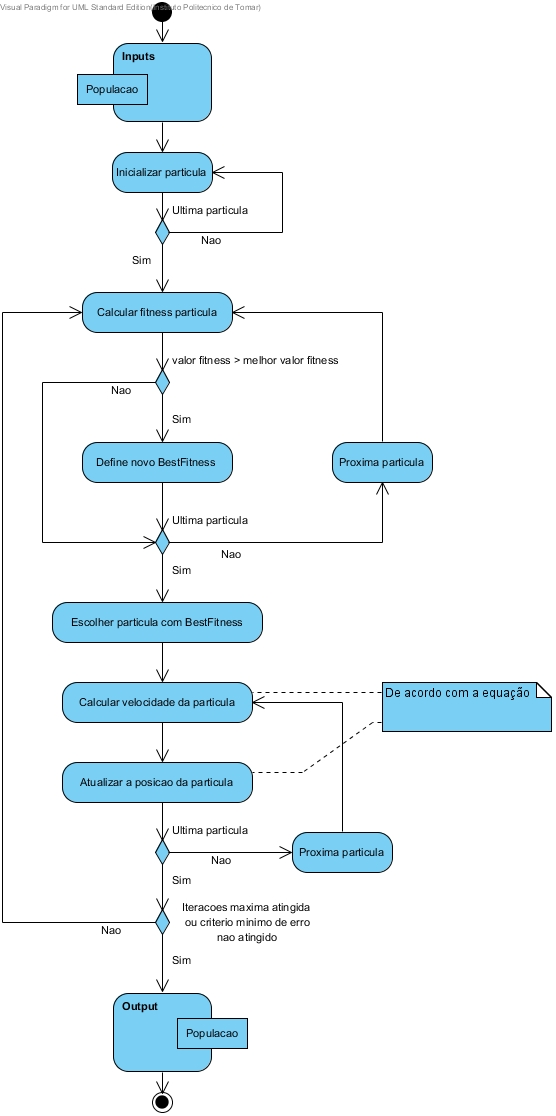
## Crossover



## Mutação Binaria



## P.S.O



## Recombinação Binaria



# Pseudo-codigo

## Fitness Penalização

entra(População p[], pesoMax, peso[], valor[])

para i := 0 ate i < tamanho(p[])

p[i].fitness := fitnessPenal(p[i].genes[], pesoMax, peso[], valor[])

i := i+1

sai(População p[])

fitnessPenal(Individuo genes[], pesoMax, peso[], valor[])

para i := 0 ate i < tamanho(peso[])

se genes[i] = 1 entao

val := val+valor[i]

pes := pes+peso[i]

i := i+1

se pes <= pesoMax entao

fitness := val

senao

fitness := val-(pes-pesoMax)

se fitness < 0 entao

fitness := 0

retorna fitness

## Reparação aleatória

entra(População p[], pesoMax, peso[], valor[])

para i := 0 até i < tamanho(p[])

p[i] := reparaAleatorio(p[i].genes[], pesoMax, peso[], valor[])

i := i+1

sai(População p[])

reparaAleatorio(Individuo genes[], pesoMax, peso[], valor[])

dim := tamanho(genes[])

para i := 0 ate i < dim

se genes[i] = 1 entao

val := val+valor[i]

pes := pes+peso[i]

i := i+1

se pes <= pesoMax entao

Individuo genes[].fitness := val

retorna genes[]

senão

fazer

n := aleatorio([0;dim[)

enquanto (genes[n] = 0)

genes[n] := 0

retorna reparaAleatorio(genes[])

## Reparação aleatória

entra(População p[], pesoMax, peso[], valor[])

para i := 0 até i < tamanho(p[])

p[i] := reparaPseudoAleatorio(p[i].genes[], pesoMax, peso[], valor[])

i := i+1

sai(População p[])

reparaPseudoAleatorio(Cromossoma genes[], pesoMax, peso[], valor[])

dim := tamanho(genes[])

para i := 0 até i < dim

se genes[i] = 1 entao

val := val+valor[i]

pes := pes+peso[i]

i := i+1

se pes <= pesoMax então

Cromossoma genes[].fitness := val

retorna genes[]

senão

para i := 0 até i < dim

novo relacao[i] := valor[i]/peso[i]

fazer

k := k+1

n := menor(ordem:=k : relacao[])

enquanto (genes[n] = 0)

genes[n] := 0

retorna reparaPseudoAleatorio(genes[])

## Operador crossover

entra(População p[], individuosPretendidos)

para i := 0 até i < individuosPretendidos

Individuo pai := aleatório(p[])

fazer

Individuo mae := aleatório(p[])

enquanto (pai = mae)

para j := 0 ate j < pai.totalGenes

n := aleatório({falso,verdadeiro})

se n = falso então

f[i].gene[j] := pai.gene[j]

f[i+1].gene[j] := mae.gene[j]

senao

f[i].gene[j] := mae.gene[j]

f[i+1].gene[j] := pai.gene[j]

j := j+1

i := i+2

sai(População f[])

## Operador Roleta

entra(População pop[], individuosPretendidos)

dim := tamanho(pop[])

para i := 0 ate i < dim

pop[i].fitness := fitness(pop[i])

fitnessTotal := fitnessTotal+pop[i].fitness

i := i+1

pop[] := ordenaDecrescente(pop[].fitness)

pop[0].probabilidadeAcumulada := pop[i].fitness/fitnessTotal

para i := 1 ate i < dim

pop[i].probabilidadeAcumulada := (pop[i].fitness/fitnessTotal)+pop[i-1].probabilidadeAcumulada

i := i+1

para i := 0 ate i < individuosPretendidos

n := aleatório([0;1[)

Individuo ind[i] := procura(n : pop[].probabilidadeAcumulada)

i := i+1

sai(População ind[])

## Operador SUS

entra(Populacao pop[], individuosPretendidos)

dim := tamanho(pop[])

para i := 0 ate i < dim

pop[i].fitness := fitness(pop[i])

i := i+1

pop[0].fitnessAcumulada := pop[0].fitness

para i := 1 ate i < dim

pop[i].fitnessAcumulada := pop[i].fitness + pop[i-1].fitnessAcumulada

i := i+1

totalFitness := pop[dim-1].fitnessAcumulada

offset := totalFitness/dim

n := aleatorio([0;totalFitness[)

individuo ind[0] := procurar(n : pop[].fitnessAcumulada)

para i := 1 ate i < individuosPretendidos

n := n+offset

n := n%dim

Individuo ind[i] := procurar(n : pop[].fitnessAcumulada)

i := i+1

sai(Populacao ind[])

## Operador Truncation

entra(Populacao popA[], População popB[])

dim1 := tamanho(popA[])

dim2 := tamanho(popB[])

nova Populacao popT[dim1+dim2]

para i := 0 ate i < dim1+dim2

se i < dim1 então

popT[i] := popA[i]

senao

popT[i] := popB[i-dim1]

i := i+1

para i := 0 ate i < tamanho(popT[])

fitness(popT[i])

i := i+1

popT[] := ordenaDecrescente(popT[])

para i := 0 ate i < dim1

popA[i] := popT[i]

i := i+1

sai(Populacao popA[])

## Crossover

entra(Individuo pai, Individuo mae)

dim := tamanho(pai.genes[])

corte := aleatorio([1;dim-1])

para i := 0 até i < corte

filho.gene[i] := pai.gene[i]

filha.gene[i] := mae.gene[i]

i := i+1

para J := corte até j < dim

filho.gene[j] := procura(j, mae.gene[], filho.gene[])

filha.gene[j] := procura(j, pai.gene[], filha.gene[])

j := j+1

sai(Individuo filho, Individuo filha)

procura(pos, progenitor.gene[], descendente.gene[])

p := pos-1

q := pos

enquanto (p >= 0)

se (descendente.gene[p] = progenitor.gene[q]) entao

q := q+1

q := q%tamanho(progenitor.gene[])

p := pos

p := p-1

retorna progenitor.gene[q]

## Invertion Mutation

entra(Populaçao individuos[])

dimPop := tamanho(individuos[])

dimGene := tamanho(individuos[].genes[])

para i := 0 ate i < dimPop

gene1 := aleatorio(0<=rand<dimGene)

gene2 := aleatorio(gene1<=rand<dimGene)

enquanto(gene1 < gene2) fazer

troca(individuos[i], gene1, gene2)

gene1 := gene1+1

gene2 := gene2-1

i := i+1

sai(Populaçao individuos[])

troca(Individuo pessoa, gene1, gene2)

geneTmp := pessoa.genes[gene1]

pessoa.genes[gene1] := pessoa.genes[gene2]

pessoa.genes[gene2] := geneTmp

retorna(Individuo pessoa)

## Mutation

entra(Populaçao individuos[])

dimPop := tamanho(individuos[])

dimGene := tamanho(individuos[].genes[])

para i := 0 ate i = dimPop

gene1 := aleatorio(0<=rand<dimGene)

fazer

gene2 := aleatorio(0<=rand<dimGene)

enquanto(gene1 = gene2)

individuos[i] := troca(individuos[i], gene1, gene2)

i := i+1

sai(Populaçao individuos[])

troca(Individuo pessoa, gene1, gene2)

geneTmp := pessoa.genes[gene1]

pessoa.genes[gene1] := pessoa.genes[gene2]

pessoa.genes[gene2] := geneTmp

retorna(Individuo pessoa)

## PMX

entra(Individuo pai, Individuo mae)

dimGene := tamanho(pai.genes[])

corte1 := aleatorio(0<=rand<dimGene)

corte2 := aleatorio(corte1<rand<dimGene)

Individuo filho1 := novo Individuo[dimGene]

Individuo filho2 := novo Individuo[dimGene]

para i := 0 ate i < dimGene

se(i >= corte1 e i < corte2) entao

filho1.gene[i] := mae.gene[i]

filho2.gene[i] := pai.gene[i]

senao

filho1.gene[i] := procura(pai, mae, i, corte1, corte2))

filho2.gene[i] := procura(mae, pai, i, corte1, corte2))

i := i+1

sai(Individuo filho1, Individuo filho2)

procura(Individuo progenitor1, Individuo progenitor2, posicao, corte1, corte2)

para i := corte1 ate i < corte2

se(progenitor1.gene[posicao] = progenitor2.gene[i])

retorna(Gene progenitor1.gene[i])

i := i+1

retorna(Gene progenitor1.gene[posicao])

## SUS Minimização

entra(Populaçao individuos[])

maxFitness := maiorFitness(individuos[])+1

para i := 0 ate i < tamanho(individuos[])

individuos[i].fitness := maxFitness-individuos[i].fitness

i := i+1

sai(Populaçao individuos[])

maiorFitness(individuos[])

max := 0

para i := 0 ate i < tamanho(individuos[])

se (max < individuos[i].fitness) entao

max := individuos[i].fitness

i := i+1

retorna max

## Crossover

entra(Populacao pais[])

totalPop := tamanho(pais[])

filhos[] := nova Populacao[totalPop]

para i := 0 ate i < totalPop-1 fazer

totalGenes := tamanho(pais[i].cromo[0].gene[])

para j := 0 ate j < totalGenes fazer

a = aleatorio(-0.25<=rand<=1.25)

filho[i].cromo[0].gene[j] := pais[i].cromo[0].gene[j] \* a + pais[i+1].cromo[0].gene[j] \* (1-a)

a = aleatorio(-0.25<=rand<=1.25)

filho[i+1].cromo[0].gene[j] := pais[i].cromo[0].gene[j] \* a + pais[i+1].cromo[0].gene[j] \* (1-a)

j := j+1

i := i+2

sai(Populacao pais[])

## Mutação binaria

entra(Populacao pais[])

para i := 0 até i < tamanho(pais[]) *//percorre a populaçao progenitora*

se (aleatorio(0<=rand<1) < 0.01) entao *//se factor de ponderaçao for menor,*

pais[i].gene[0] := mutaGene(pais[i].gene[0]) *//que 1% entao*

pais[i].gene[1] := mutaGene(pais[i].gene[1]) *//muta o alelo em causa*

i := i+1 *//nova iteraçao*

sai(Populacao pais[])

mutaGene(Gene gene)

escolhido := aleatorio(0<=rand<tamanho(gene.alelo[])) *//alelo escolhido aleatoriamente*

gene.alelo[escolhido] := 1 *//toma o valor de verdadeiro*

retorna(Gene gene)

## PSO

entra(Populacao pop[])

Para cada pop[].partícula

Inicializar pop[i].partícula

Fazer

Para cada pop[].partícula

calcularValor(pop[i].partícula.fitness)

Se (valor de fitness > melhor valor de fitness (pBest))

definaValor(pBest)

gBest := partícula com o melhor valor de fitness

Para cada pop[].partícula

v[] := v[] + c1 \* rand() \* (pbest[] - present[]) + c2 \* rand() \* (gbest[] - present[])

present[] := persent[] + v[]

Enquanto (iterações máximas não forem atingidas ou critérios mínimos erro não forem atingidos)

sai(Populacao pop[])

## Recombinacao\_binaria

entra(Populacao pais[])

dim := tamanho(pais[]) *//numero de individuos da populaçao progenitora*

dimGene0 := tamanho(pais[].gene[0].alelo[]) *//dimensao do primeiro gene*

dimGene1 := tamanho(pais[].gene[1].alelo[]) *//dimensao do segundo gene*

Populacao filhos[] := nova Populacao[dim] *//criaçao de uma geraçao vazia do tamanho da populaçao progenitora*

para i := 0 até i < dim *//percorrer a populaçao progenitora*

pais[i].fitness := fitness(pais[i]) *//calculo do fitness dos individuos da populaçao progenitora*

i := i+1 *//nova iteraçao*

pais[] := ordenaDecrescente(pais[].fitness) *//ordena a populaçao progenitora pelo valor do fitness decrescente*

para i := 0 até i < dim-1 *//ciclo para recombinaçao dos individuos*

se (aleatorio(0<=rand<=1) > 0.65) entao *//se factor de ponderaçao for maior,*

filhos[i] := pais[i] *// entao os individuos transitam como estao*

filhos[i+1] := pais[i+1] *//*

senao *//se factor de ponderaçao não for maior,*

corte := aleatorio(1<=rand<=dimGene0-1) *//calculo do ponto de corte do primeiro gene*

para j := 0 até j < corte *//até ao corte do primeiro gene*

filhos[i].gene[0].alelo[j] := pais[i].gene[0].alelo[j] *//copia dos pais para os filhos*

filhos[i+1].gene[0].alelo[j] := pais[i+1].gene[0].alelo[j] *//*

j := j+1 *//nova iteraçao*

para k := corte até k < dim *//depois do corte do primeiro gene*

filhos[i].gene[0].alelo[k] := pais[i+1].gene[0].alelo[k] *//copia dos pais para os filhos,*

filhos[i+1].gene[0].alelo[k] := pais[i].gene[0].alelo[k] *//com os progenitores trocados*

k := k+1 *//nova iteraçao*

corte := aleatorio(1<=rand<=dimGene1-1) *//calculo do ponto de corte do segundo gene*

para m := 0 até m < corte *//até ao corte do segundo gene*

filhos[i].gene[1].alelo[m] := pais[i].gene[1].alelo[m] *//copia dos pais para os filhos*

filhos[i+1].gene[1].alelo[m] := pais[i+1].gene[1].alelo[m]*//*

m := m+1 *//nova iteraçao*

para n := corte até n < dim *//depois do corte do segundo gene*

filhos[i].gene[1].alelo[n] := pais[i+1].gene[1].alelo[n] *//copia dos pais para os filhos*

filhos[i+1].gene[1].alelo[n] := pais[i].gene[1].alelo[n] *//com os progintores trocados*

n := n+1 *//nova iteraçao*

i := i+2 *//nova iteraçao com um incremento de 2*

sai(Populacao filhos[])