Instituto Politécnico de Tomar

Projecto de Sistemas de Informação

Realizado por :

Paulo Chorinca

Carlos Neto

Diogo António

Bruno Mendes

Gonçalo Dias

Inovative

Índice

Resumo

Desenvolveu-se neste trabalho a implementação computacional de um algoritmo genético.
Este se constituiu de uma população inicial sobre a qual agem, operadores fundamentais uma nova população.
Sobre a qual agem novamente os operadores genéticos, e assim sucessivamente produzindo uma sequência de populações. O operador *seleção* foi implementado em dois algoritmos básicos: *roleta* e *torneio*. A *substituição* de indivíduos da população pelos filhos ocorre de três maneiras básicas: *dos pais*,*dos menos aptos,* e *dos indivíduos sorteados aleatoriamente*.

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

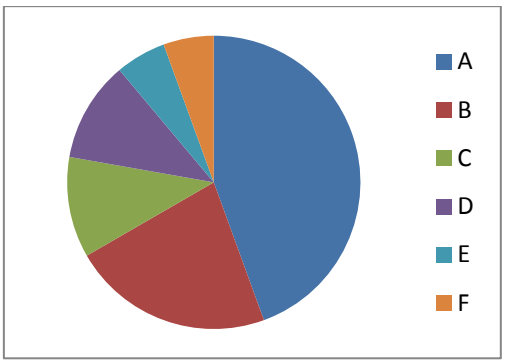
Características gerais

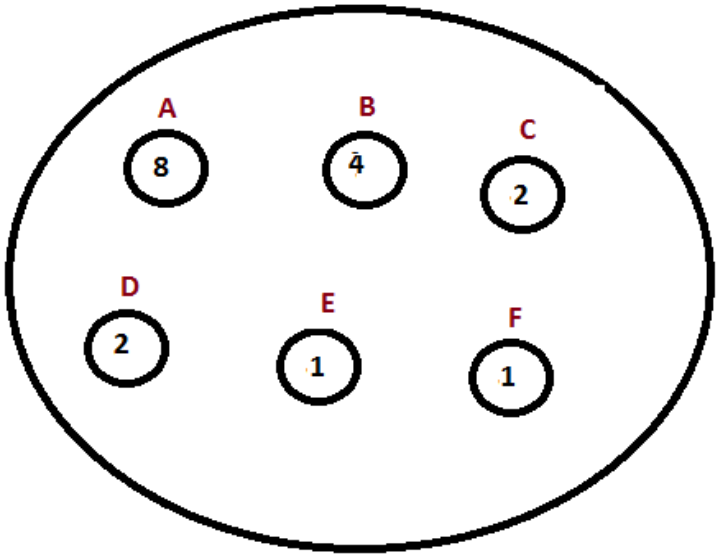
- Utilizam uma codificação do conjunto de
- Parâmetros (indivíduos)e não comos próprios parâmetros(estados);
- Vasculham várias regiões do espaço de busca de cada vez;
- Utilizam informações diretas de qualidade ,em contraste com as derivadas utilizadas nos métodos tradicionais de otimização;
- Utilizam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

2. Fazer a soma do fitness do individuo anterior ao ser, com base na ordem em que se encontram na linha;
3. Obter o total do somatório de todos os fitness, o fitness do ultimo indivíduo, pois é o acumulado de todos os indivíduos;
4. 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;
5. Definir qual vai ser o offset que se vai acrescentar ao ponto de partido. Calcula através

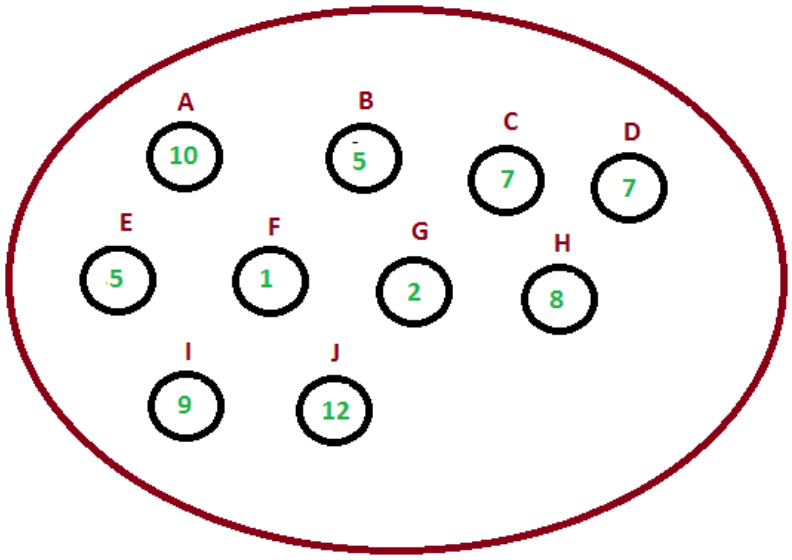
Total fitness

_____

Total nº indivíduos

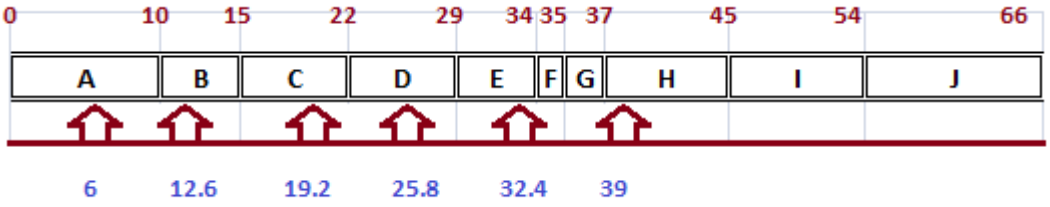
6. Selecionar o individuo para onde o ponto aponta;
7. Somar ao ponto o offset;
8. Repetir os pontos 6º e 7º ate obter o numero de indivíduos selecionados pretendidos;
9. Caso se atinja o fim da linha de indivíduos então volta se ao inicio da linha

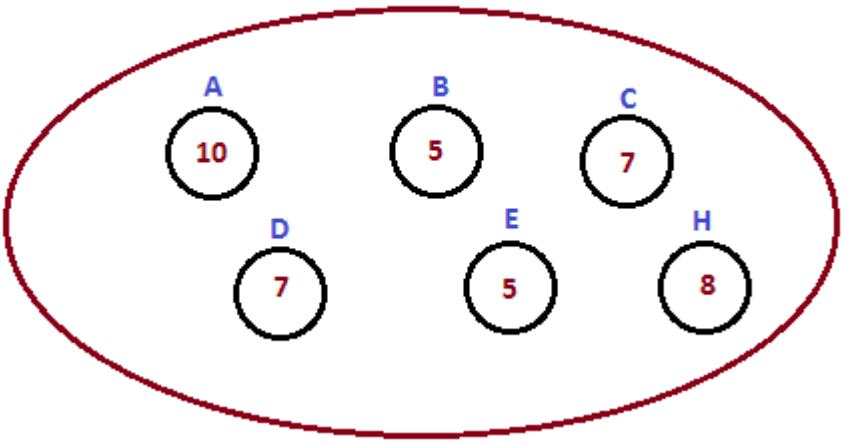
Exemplo: população 10 indivíduos e devemos escolher 6 indivíduos para reprodução



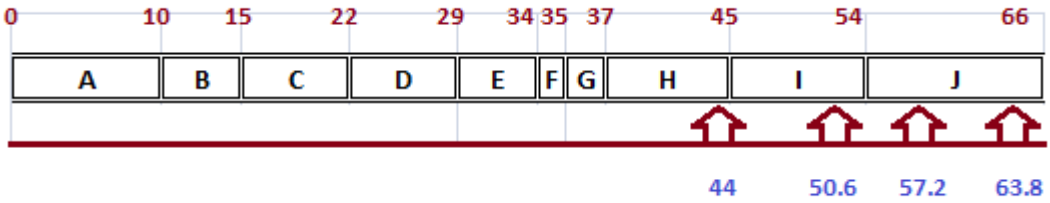
População

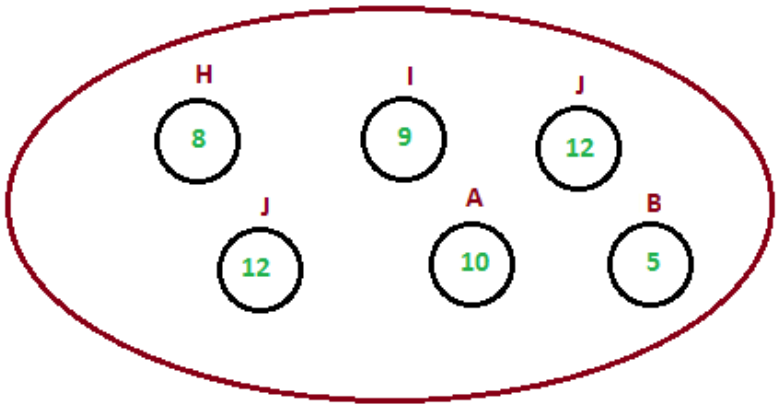
$$offset = \frac{\text{total fitness}}{\text{total nº individuos}} = \frac{66}{10} = 6.6$$





Indivíduos selecionados





Indivíduos Selecionados

Uniforme –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

3. 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.

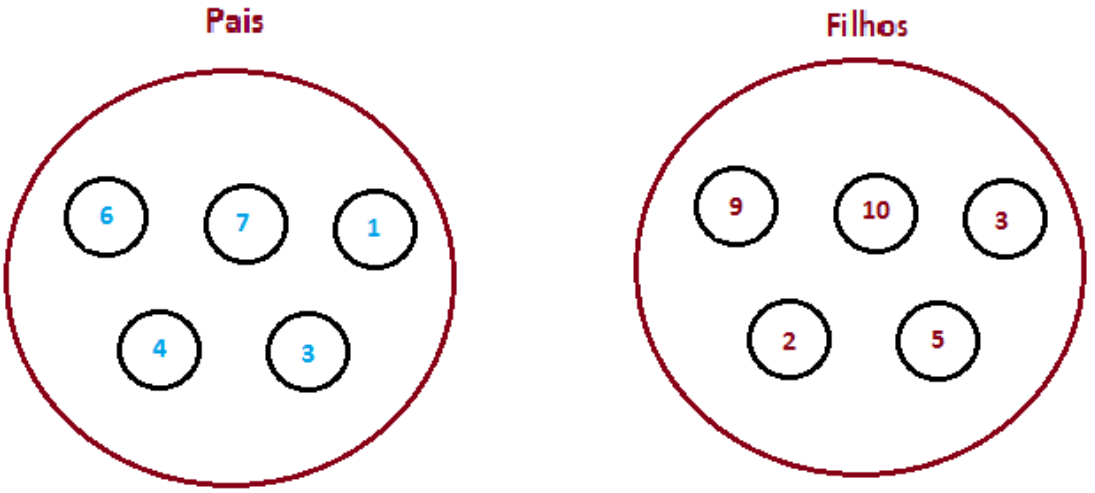
Solução

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

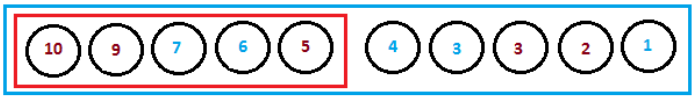
Operados treincation –substituição

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

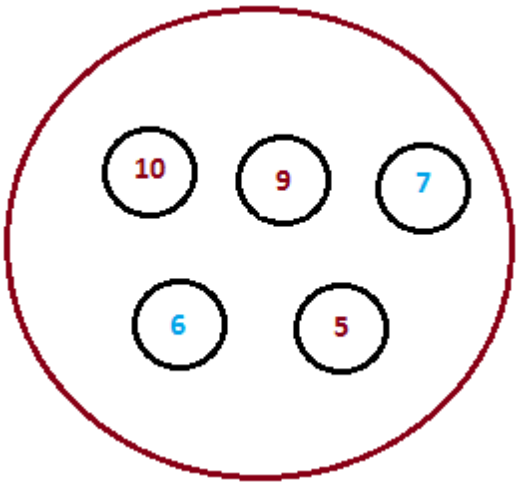


Indivíduos ordenados



Os indivíduos das duas populações juntos e ordenados

Os 5 primeiros indivíduos



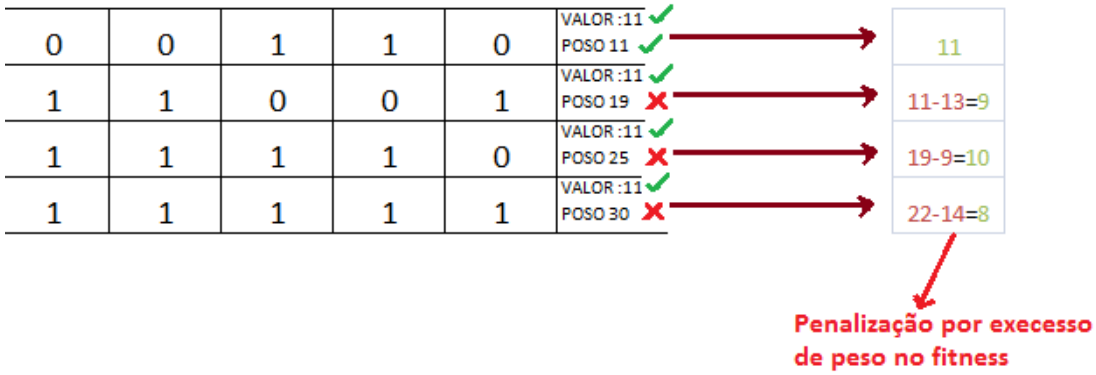
Nova População os 5 indivíduos com melhor fitness selecionado

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

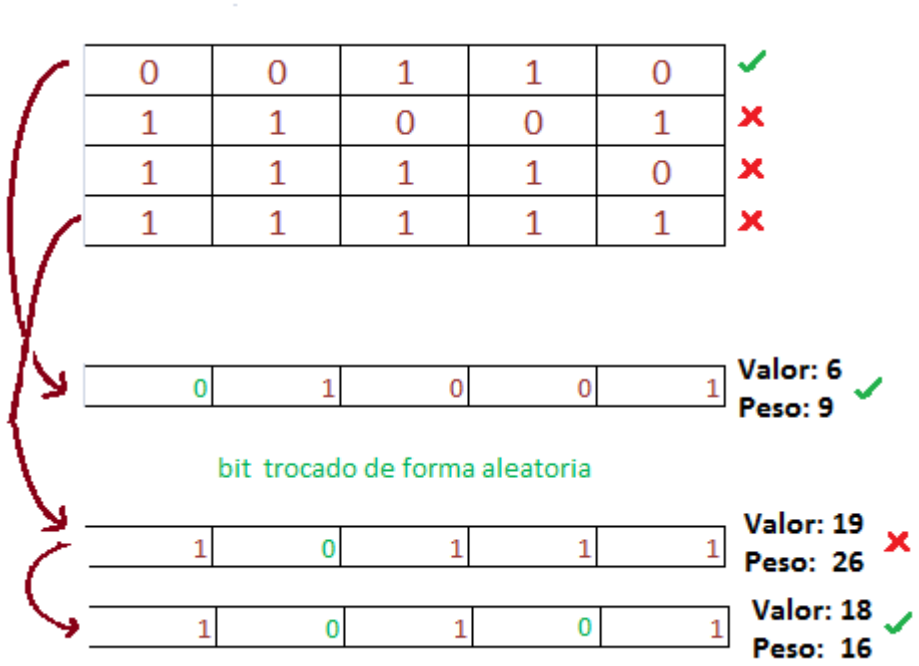


Tipos de penalizações

Linear: $-3^1=3$ (usada no exemplo anterior)

Quadrática: $-3^2=9$

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

Mochila Fitness Penalização
Mochila Reparacao Aleatoria
Mochila Reparacao Pseudo Aleatoria
Mochila-part1
Mochila-part2
Mochila-part3
Mochila-part4
Operador Roleta
Operador SUS
Operador Truncation
Operator Truncation
pseudo-codigo_mochila fitnessPenalizacao
pseudo-codigo_mochila reparacao aleatoria
pseudo-codigo_mochila reparacao pseudo-aleatoria
pseudo-codigo_Operador crossover
pseudo-codigo_Operador Roleta
pseudo-codigo_Operador SUS
pseudo-codigo_Operador Truncation
Relatorio_Final
Roleta
SUS-part1
SUS-part2
UniformCrossover-part1
UniformeCrossover-part2
Esquemas 1
Esquemas 2
Esquemas 3
Esquemas 4
Esquemas 5
Esquemas 6
Versao2

Reparação:Psenda - Aleatória

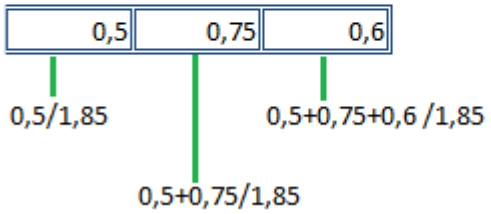
Peso	10	4	1	10	5
Valor	5	3	10	1	3
Valor/Peso	0,5	0,75	10	0,1	0,6

Calculo de Relação

$$\text{Relação} = \frac{\text{Valor}}{\text{Peso}} = \frac{5}{10} = 0{,}5$$

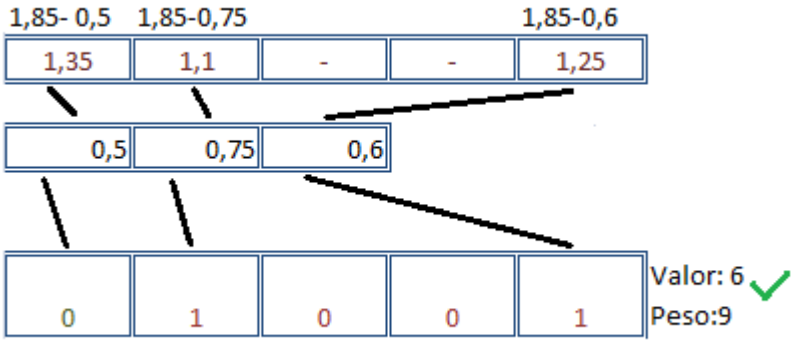
Exemplo:

1	1	0	0	1
0,5	0,75	-	-	0,6



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

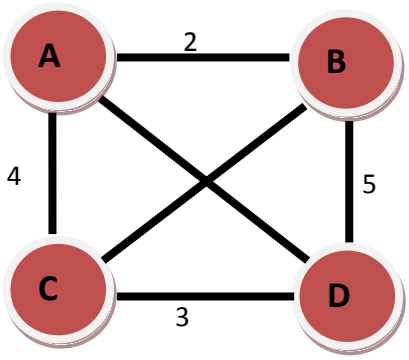
Descrição:

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 n^2 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	-

Representação Gene

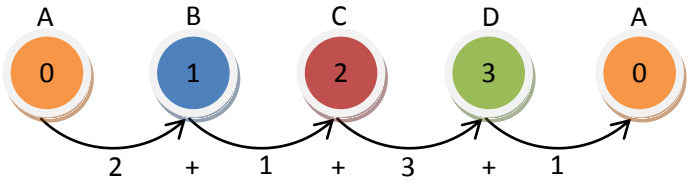
0	1	2	3

Cidade	A	B	C	D
Índex	0	1	2	3

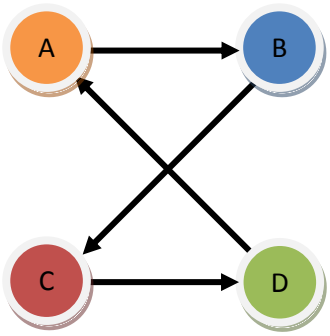
Ordem:

Cálculo do Fitness:

- Exemplo 1

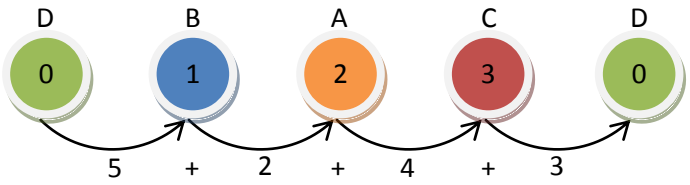


= 7 Fitness

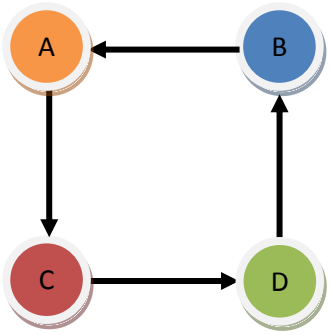


Representação do caminho

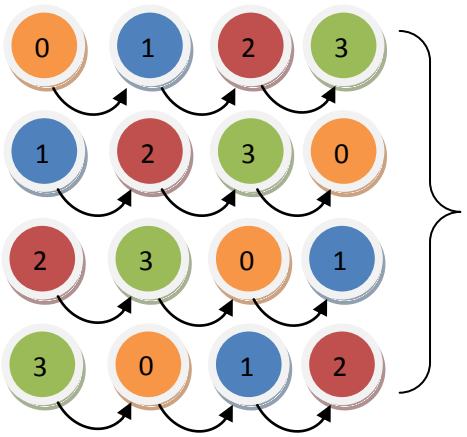
- Exemplo 2



= 14 Fitness



Representação do caminho

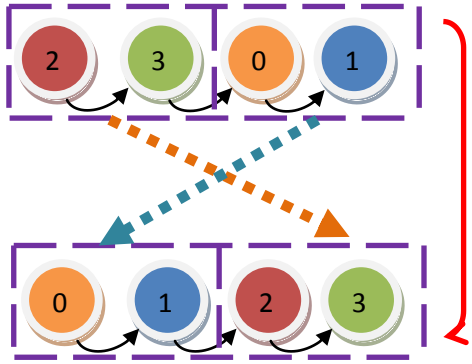


São sempre o mesmo caminho,

começam é em pontos diferentes

Reparar /Mitigar Programa

Pegar na cidade 0 e por como primeira cidade através de uma rotação, para o mesmo caminho só ter uma representação.



Transforma-se numa única

representação

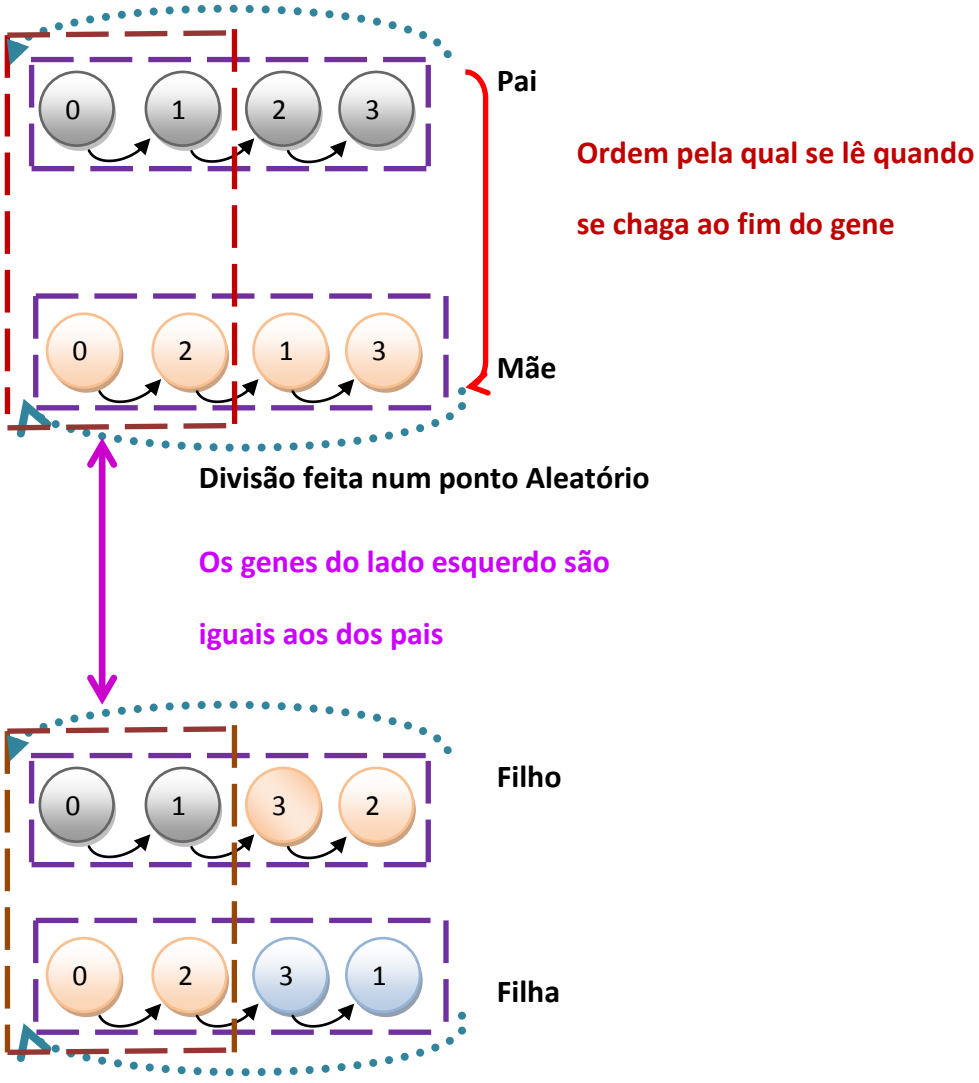
fazemos isso, só partir o gene em dois no ponto onde aparece o 0, depois criamos um novo gene com a(2º parte) em primeiro e a (1º parte) em segundo, e assim fica o 0 sempre como cidade de partida.

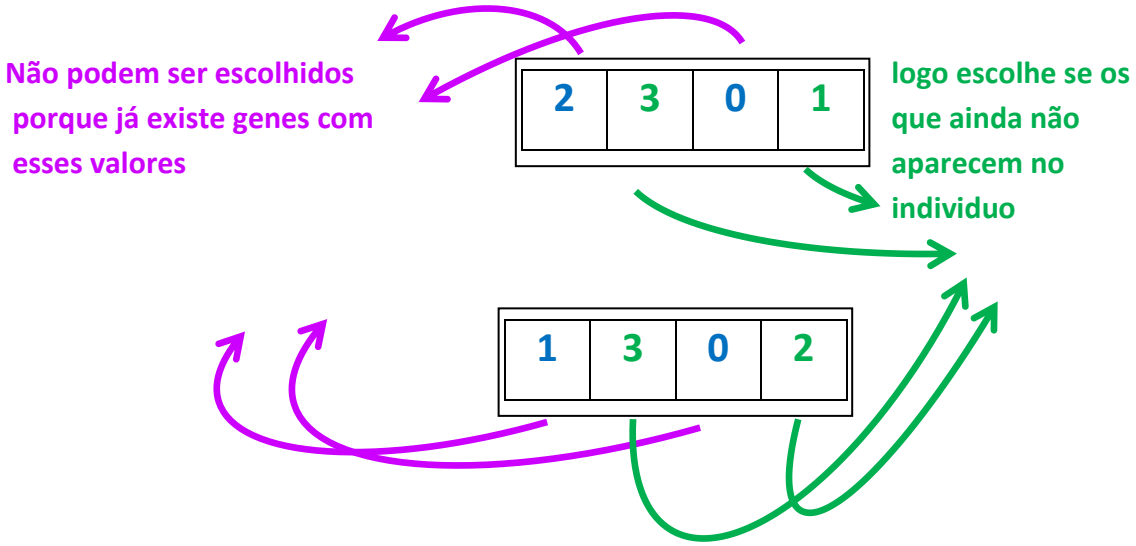
Order Crossover

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





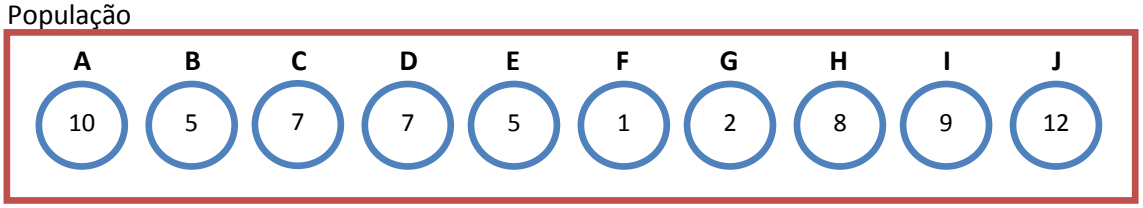
Código

```
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



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

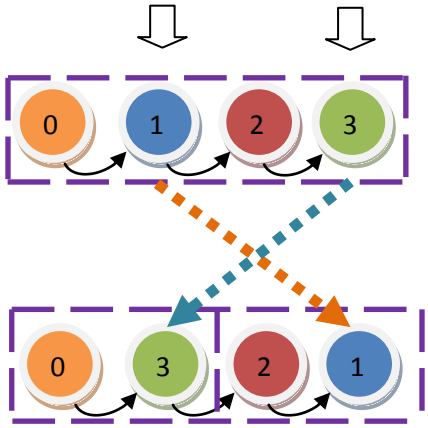
Nota: Não esquecer que neste SUS procuramos o valor mais pequeno de fitness, mas o SUS "não devolve os valores mais pequenos", logo tivemos que transformar os valores grandes em valores pequenos e fazer o contrário com os valores pequenos.

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

DemoTSP
TSP
Apresentação Caixeiro Viajante
CaxeiroViajante-Part1
CaxeiroViajante-Part2
CaxeiroViajante-Part3
cycleCrossover(fluxograma)
cycleCrossover
pseudo-codigo cycleCrossover
InversionMutation
InvertionMutation
pseudo-codigo InvertionMutation
Mutation
pseudo-codigo Mutation
PMX
PMX
PMX_1
PMX_2
Relatorio_Final
SUS Minimizacao
pseudo-codigo SUS_Minimizacao
SUS_Minimizacao-Part1
SUS_Minimizacao-Part2
SwapGenes fluxograma
Swapgenes
pseudo-codigo Swapgenes
Versao3

Função de avaliação

$f(x_1,x_2)=21.5 + x_1\ m(4\pi*x_1)+x_2\ m(20\ \pi*x_2)$

com, $-3,0 \le x_1 \le 12,1$ e $4,1 \le x_2 \le 5,8$

Representação binaria para 4 casas decimais :

$Dx_1=\underline{15,1}=12,1 -(-3,0)$

$15,1 *10\ 000 = 15100 \Rightarrow 2^{17} \le 151000 \le 2^{18}$

$Dx_2=\underline{1,7}=5,8 - 4,1$

$1,7 *10\ 000 =17000 \Rightarrow 2^{14} \le 17000 \le 2^{15}$

Indivíduo i é composto de 2 genes (x_1 e x_2)

X_1 =18 alelos

X_2 =15 alelos

Exemplo:

1	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**

X_1= =250552d

0	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**		

X_2= =4800d

$X'_1=-3+250552+\frac{15.1}{2^{18}-1}=11,4323$

$X'_2=4.1+4800+\frac{1.7}{2^{15}-1}=4,3490$

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

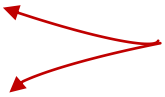
1. Escolhem-se 2 indivíduos do topo da lista obtida (que ainda não tivessem sido escolhidos)
2. Calcula-se um numero aleatório entre [0;1[

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 x_1=a+b



Pontos de corte aleatória

gene x_2=c+d

Indivíduos:i_1 e i_2

i_1= x_1 & x_2 ⇔ i_1= a+b & c+d

i_2= x_1 & x_2 ⇔ i_1= 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 .

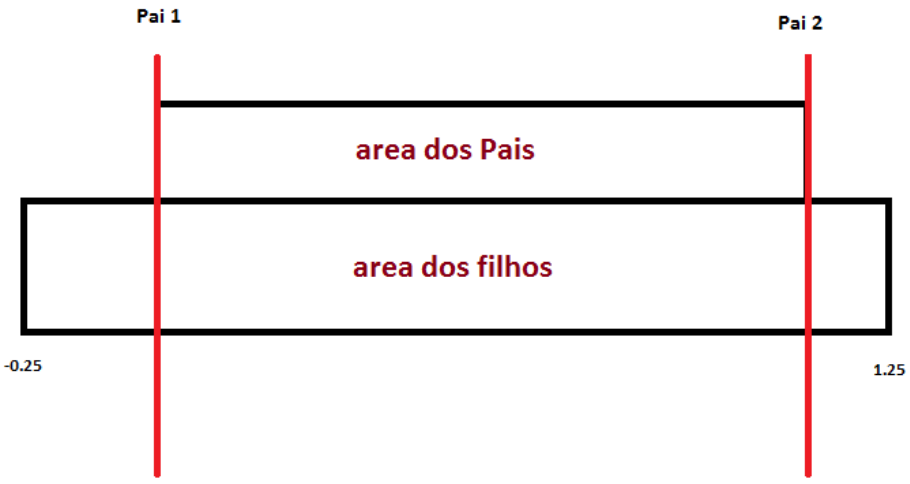
4. Volta-se ao ponto 2 até terminar a população.

Exemplo:

i'_3= 1011011010000111010 & 1110010001011110

Intermediate recombination

1. Selecionam-se 2 indivíduos, P_1 e P_2, assumindo o papel de "~~Pais~~" (P_1 "Pai_1")
2. Selecionam-se os genes $Genes_i^{P1}$ e $Genes_i^{P2}$, dos indivíduos P_1 e P_2 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 $Genes_i^F$ do novo indivíduo F⟶"filho ", através da seguinte formula.
   $Genes_i^{F} = Genes_i^{P1} * a_i + Genes_i^{P2} * a_i$
5. Volta-se ao ponto 2 até que todos os genes dos indivíduos " Pais " P_1 e P_2 sejam percorridos (i=numero genes).



Exemplo

Considerar os seguintes indivíduos, com 3 genes cada

Pai 1	12	25	5
Pai 2	123	4	34
	$Genes_1^{P2}$	$Genes_2^{P2}$	$Genes_3^{P2}$

Considerar os seguintes valores de a para esta exemplo

a_1	a_2	a_3
0.5	1.1	-0.1

O novo filho calculado

	$Genes_1^F$	$Genes_2^F$	$Genes_3^F$
Pai 2	67.5	1.9	2.1

$$Genes_i^F = Genes_i^{P1} * a_i + Genes_i^{P2} * (1- a_i)$$

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

= 6 + 61.5

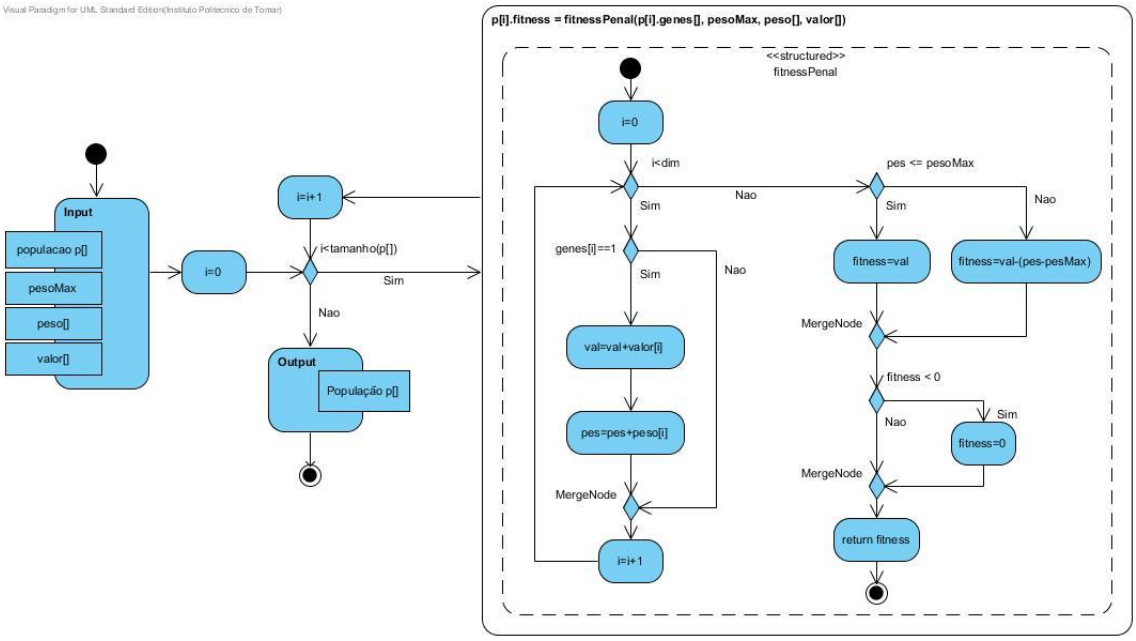
=67.5

Anexos

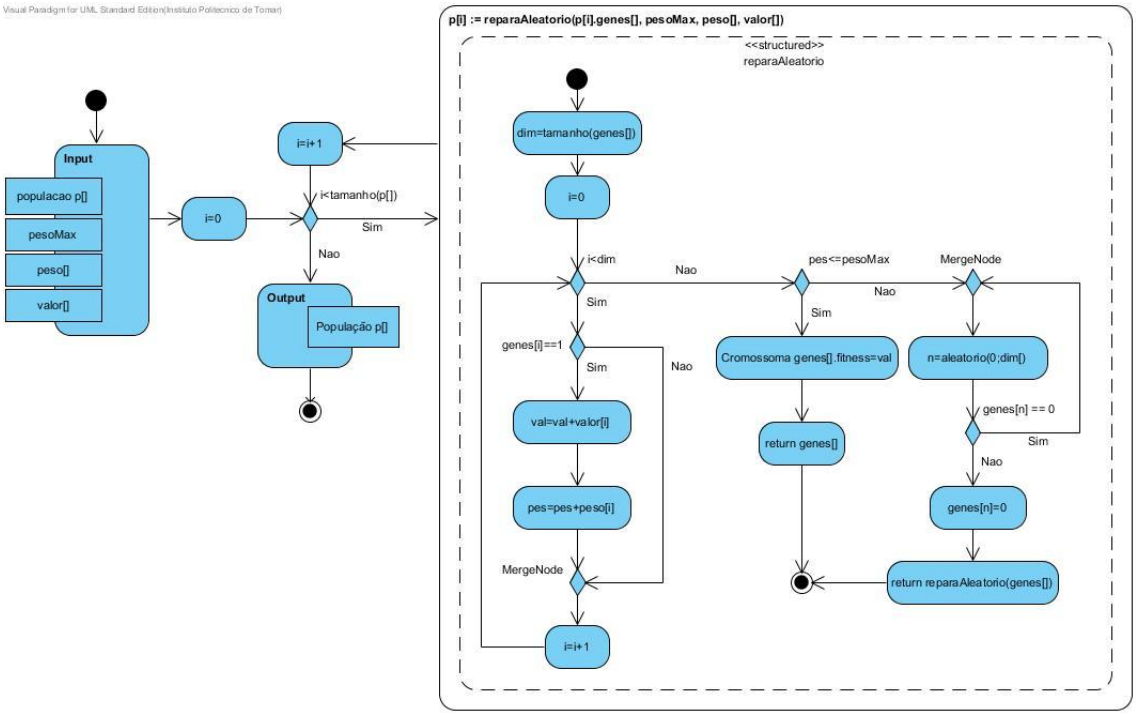
Demo
GeneticAlgorithm_real_coded
PSO_DE_GA
Differential Evolution
intermediate_crossover
intermediate_crossover_parte1
intermediate_crossover_parte2
IntermediateCrossover
Mutation Binaria
optimizacao_funcaoMat_PDF
_pseudo-codigo Mutation_binaria
P.S.O
PSO
pseudo-codigo PSO
Recombinacao Binaria
pseudo-codigo Recombinacao_binaria
Relatorio_Final4
Esquemas 1
Esquemas 2
Esquemas 3
solucaoDemo
Versao4

Fluxogramas

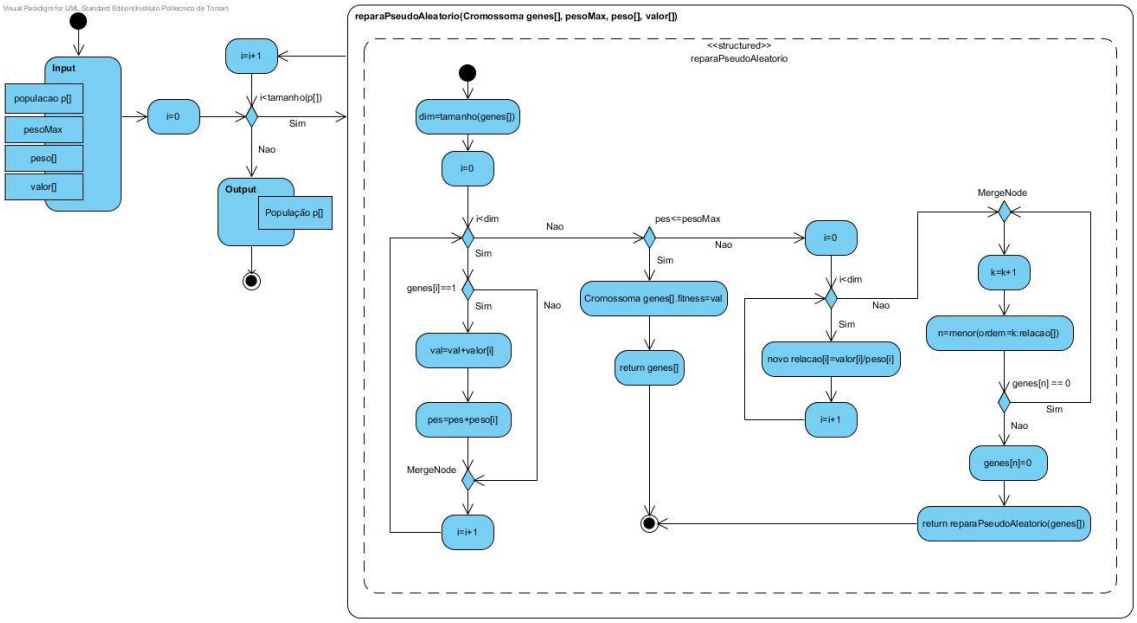
Fitness Penalização



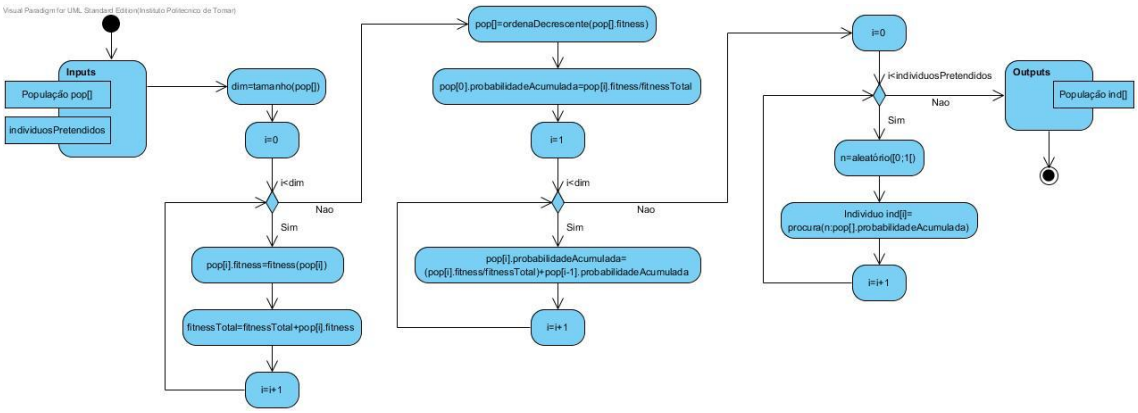
Reparação Aleatória



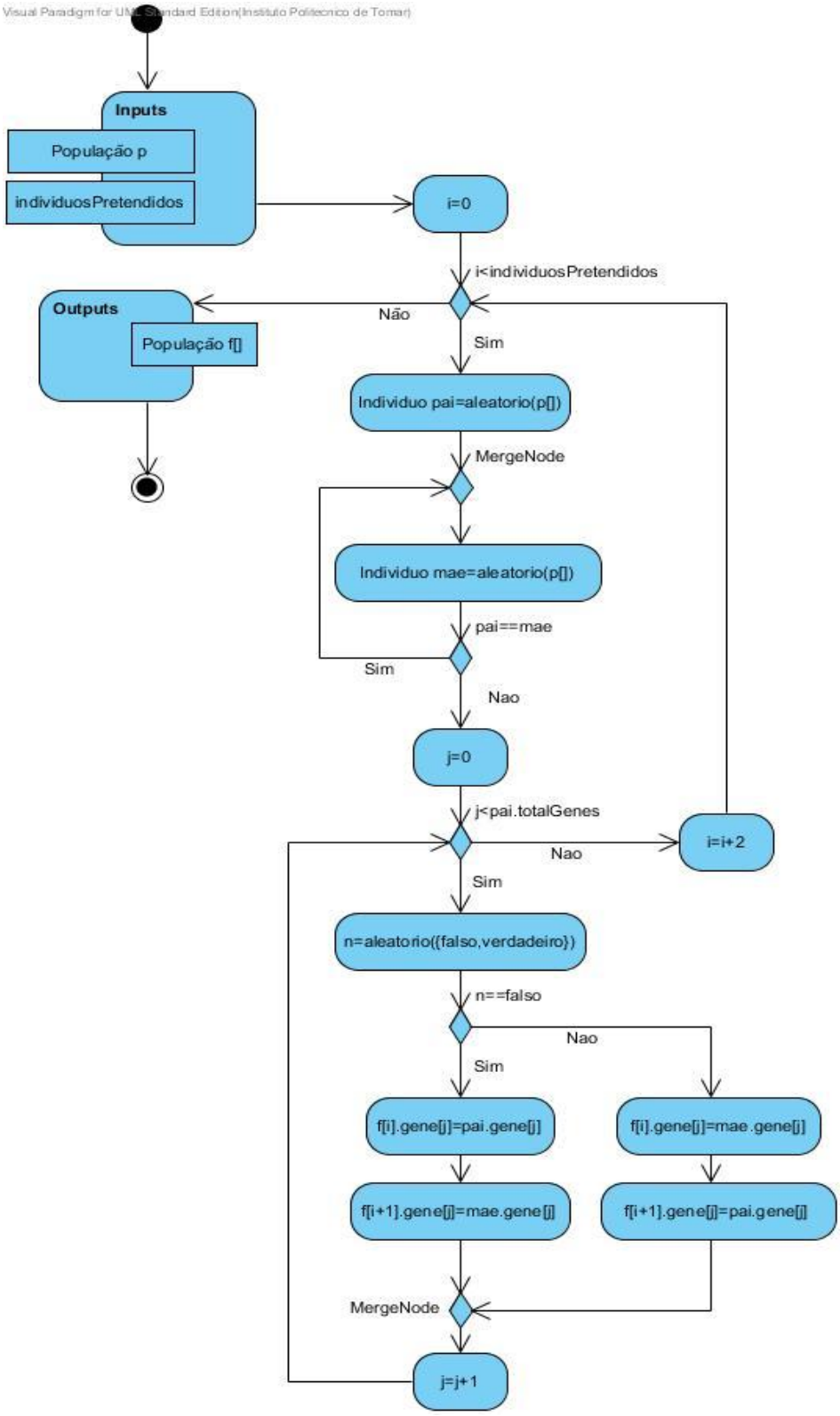
Reparação Pseudo Aleatória



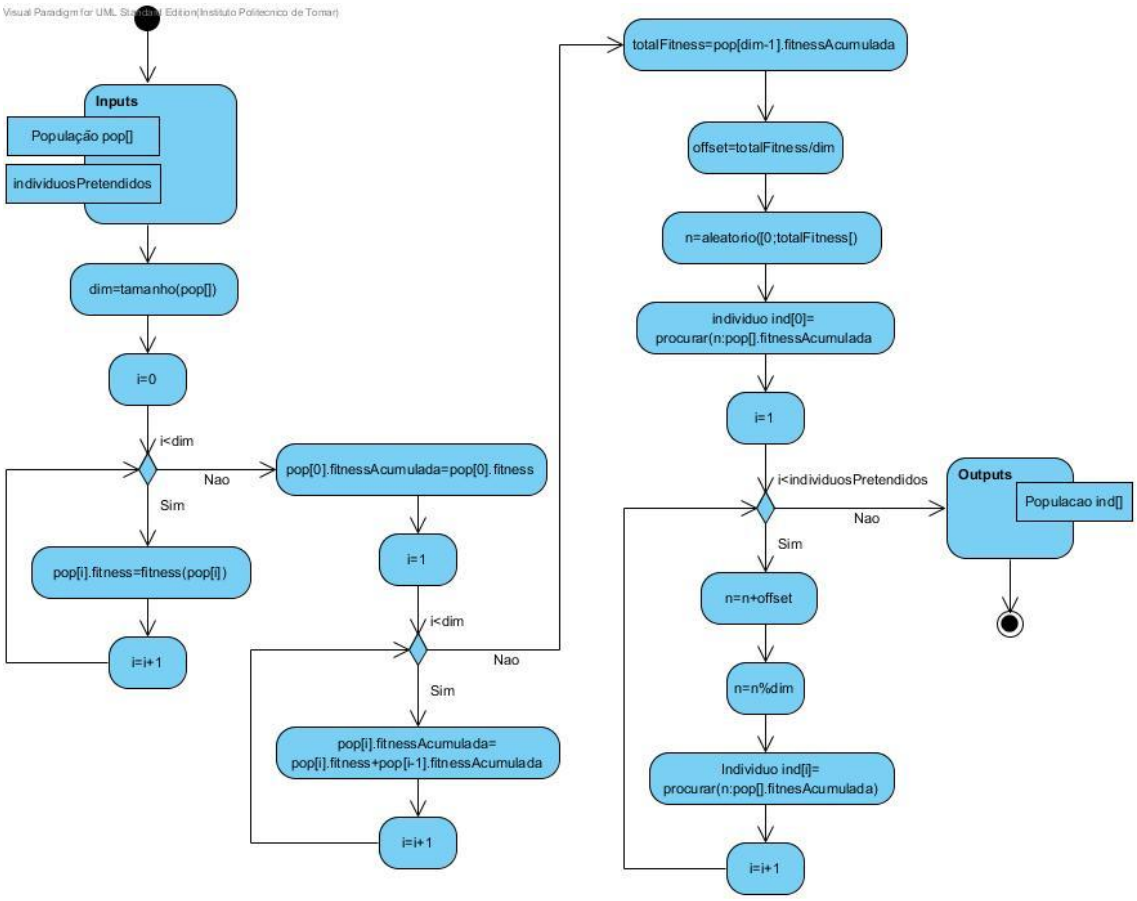
Operador Roleta



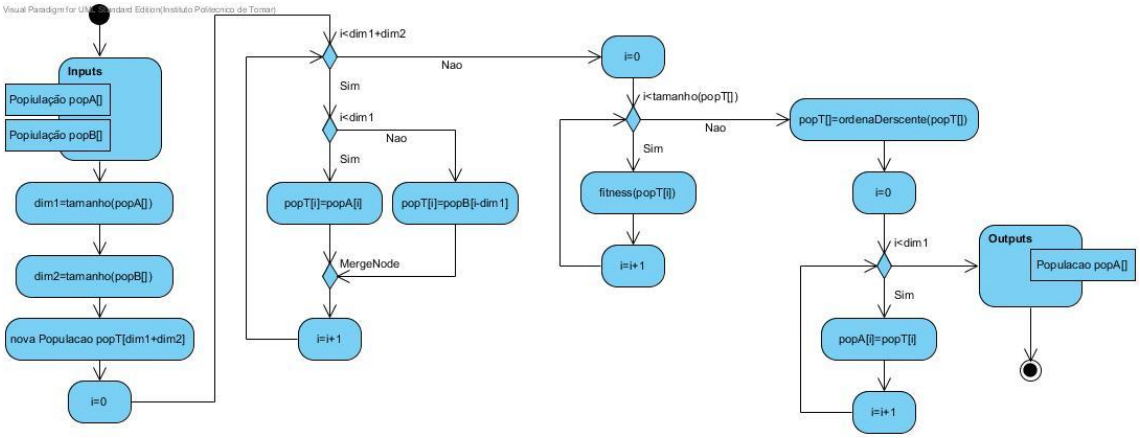
Operador Crossover



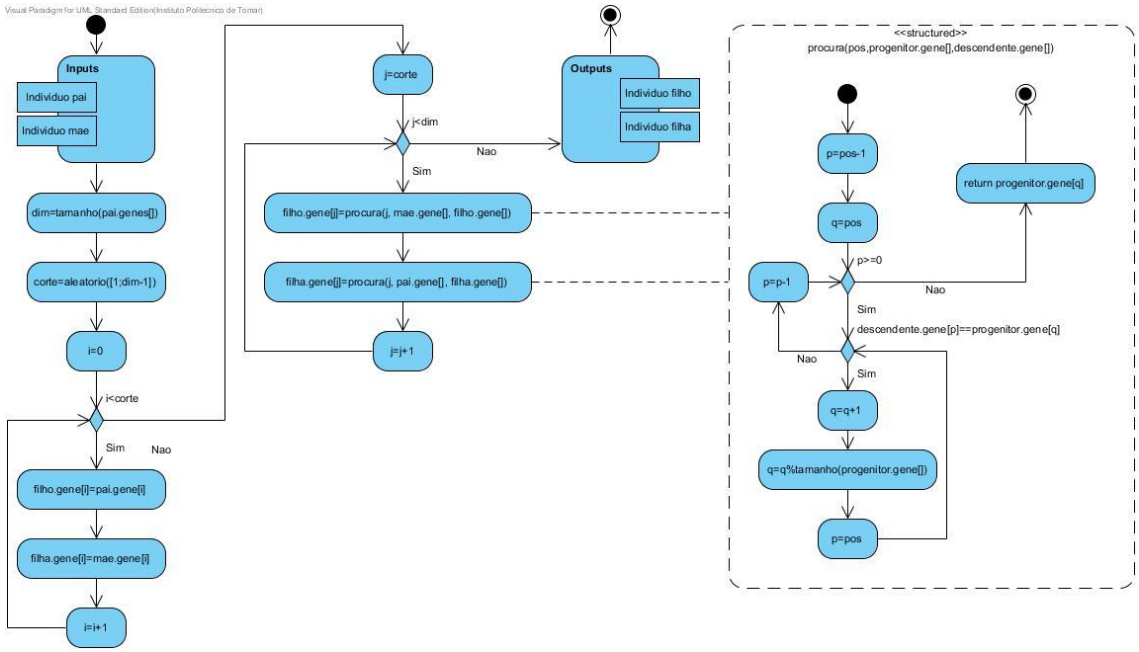
Operador SUS



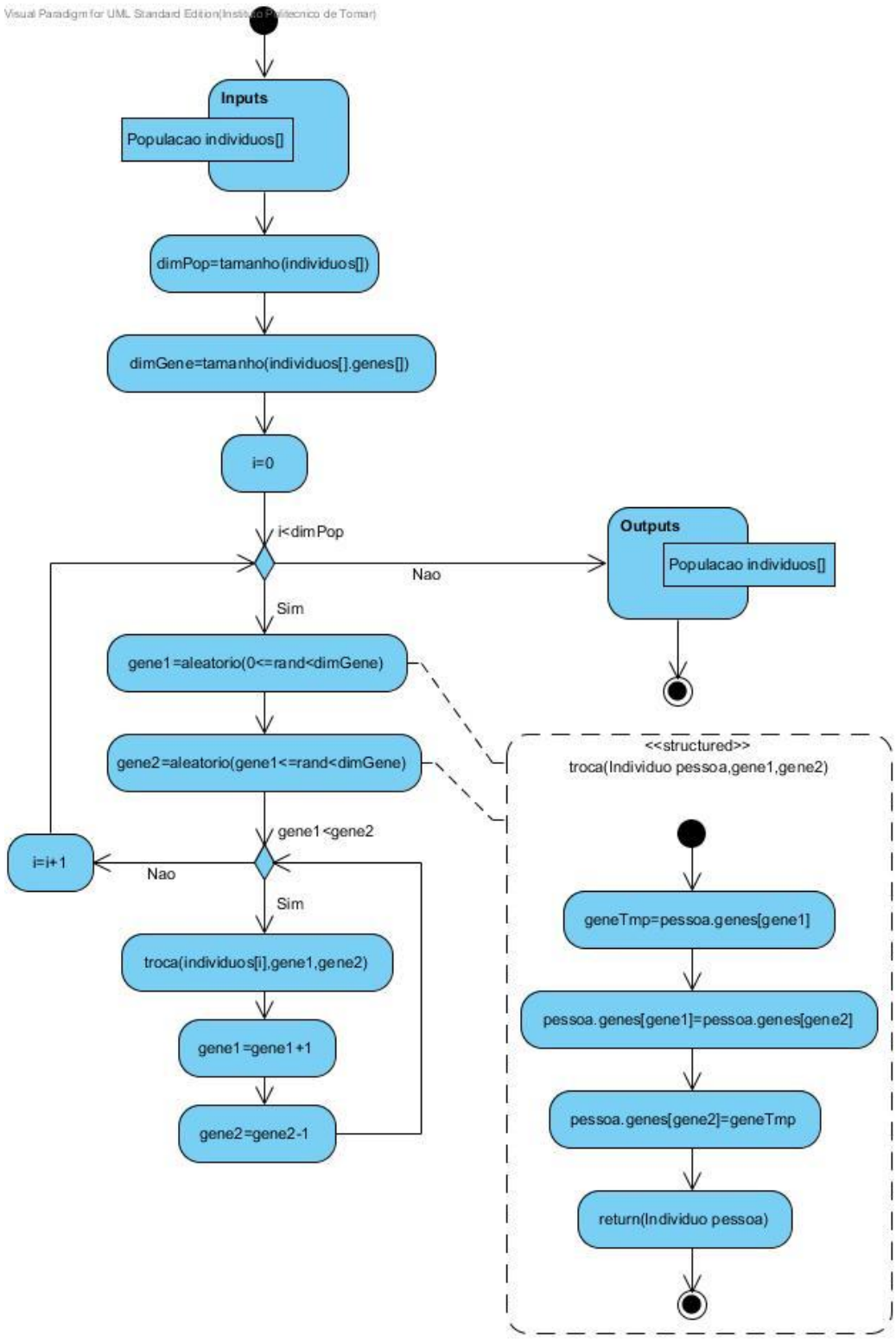
Operador Truncation



Crossover

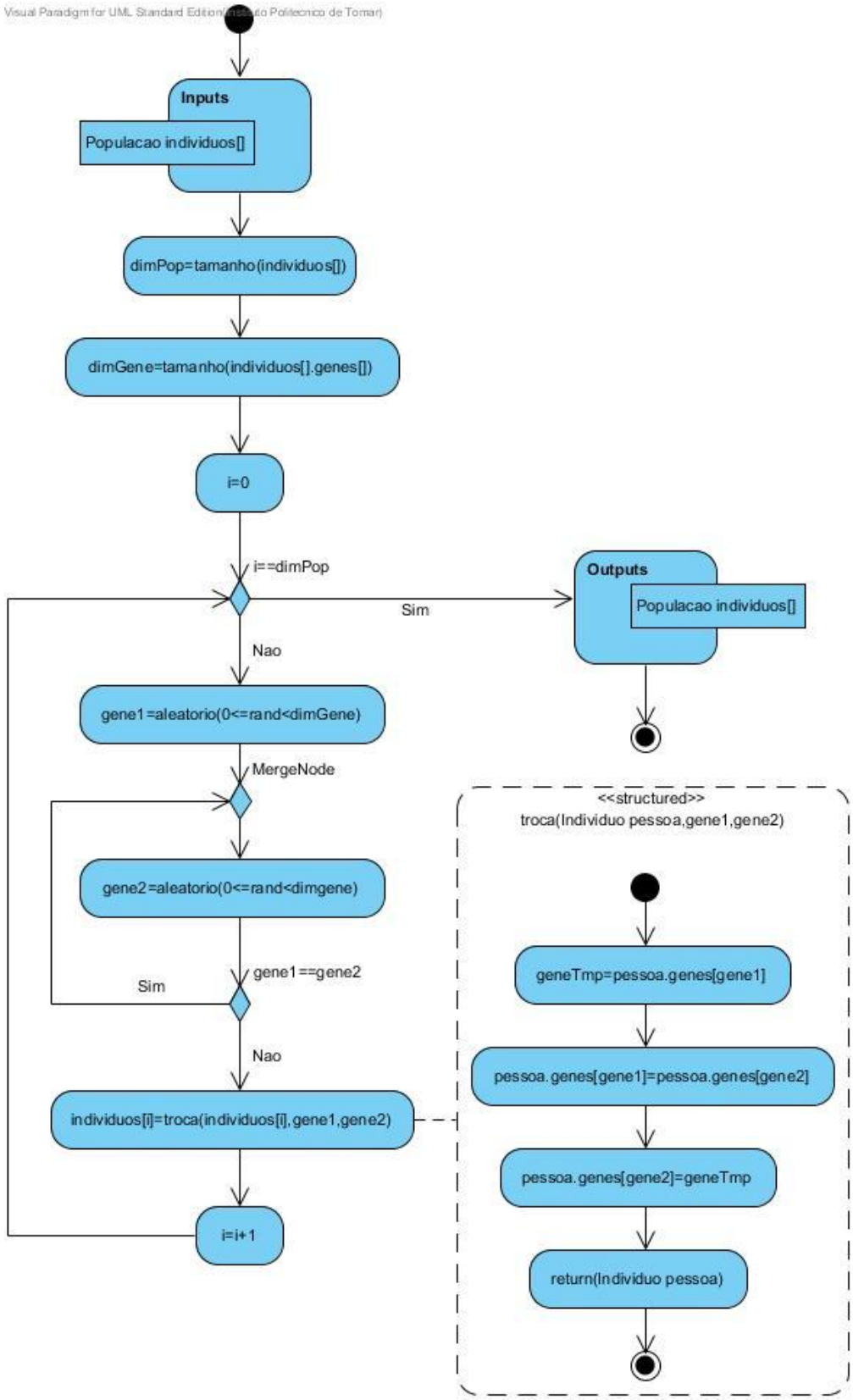

Visual Paradigm for UML Standard Edition(Instituto Politecnico de Tomar)
Inputs
Individuo pai
Individuo mae
dim=tamanho(pai.genes[])
corte=aleatorio([1;dim-1])
i=0
i<corte
Sim
Nao
filho.gene[i]=pai.gene[i]
filha.gene[i]=mae.gene[i]
i=i+1
j=corte
j<dim
Nao
Sim
filho.gene[j]=procura(j, mae.gene[], filho.gene[])
filha.gene[j]=procura(j, pai.gene[], filha.gene[])
j=j+1
Outputs
Individuo filho
Individuo filha
<<structured>>
procura(pos,progenitor.gene[],descendente.gene[])
p=pos-1
q=pos
p>=0
Nao
Sim
return progenitor.gene[q]
p=p-1
descendente.gene[p]==progenitor.gene[q]
Nao
Sim
q=q+1
q=q%tamanho(progenitor.gene[])
p=pos

Invertion Mutation

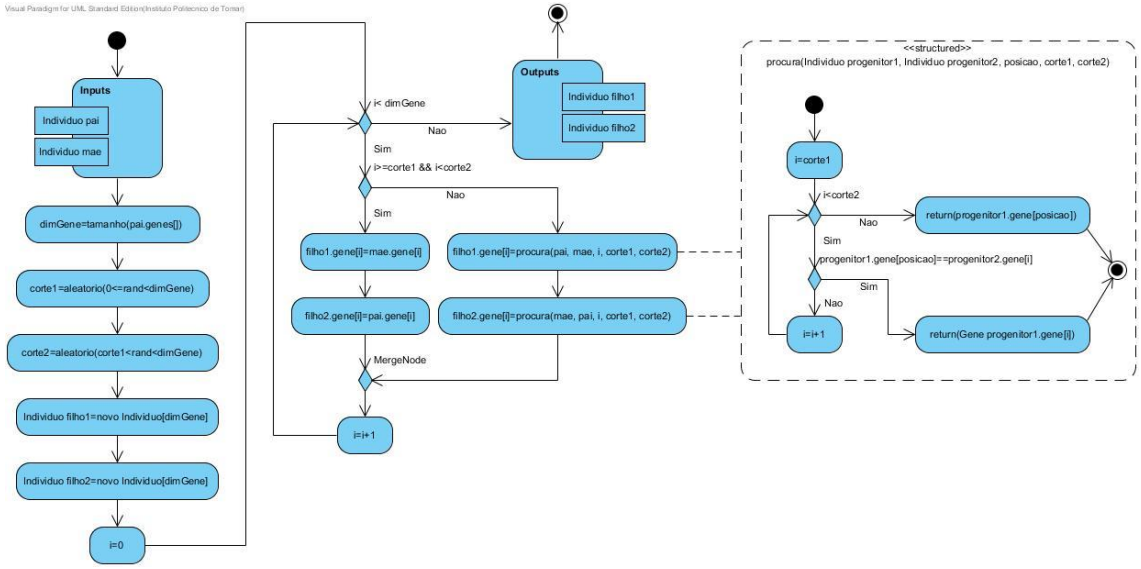


.

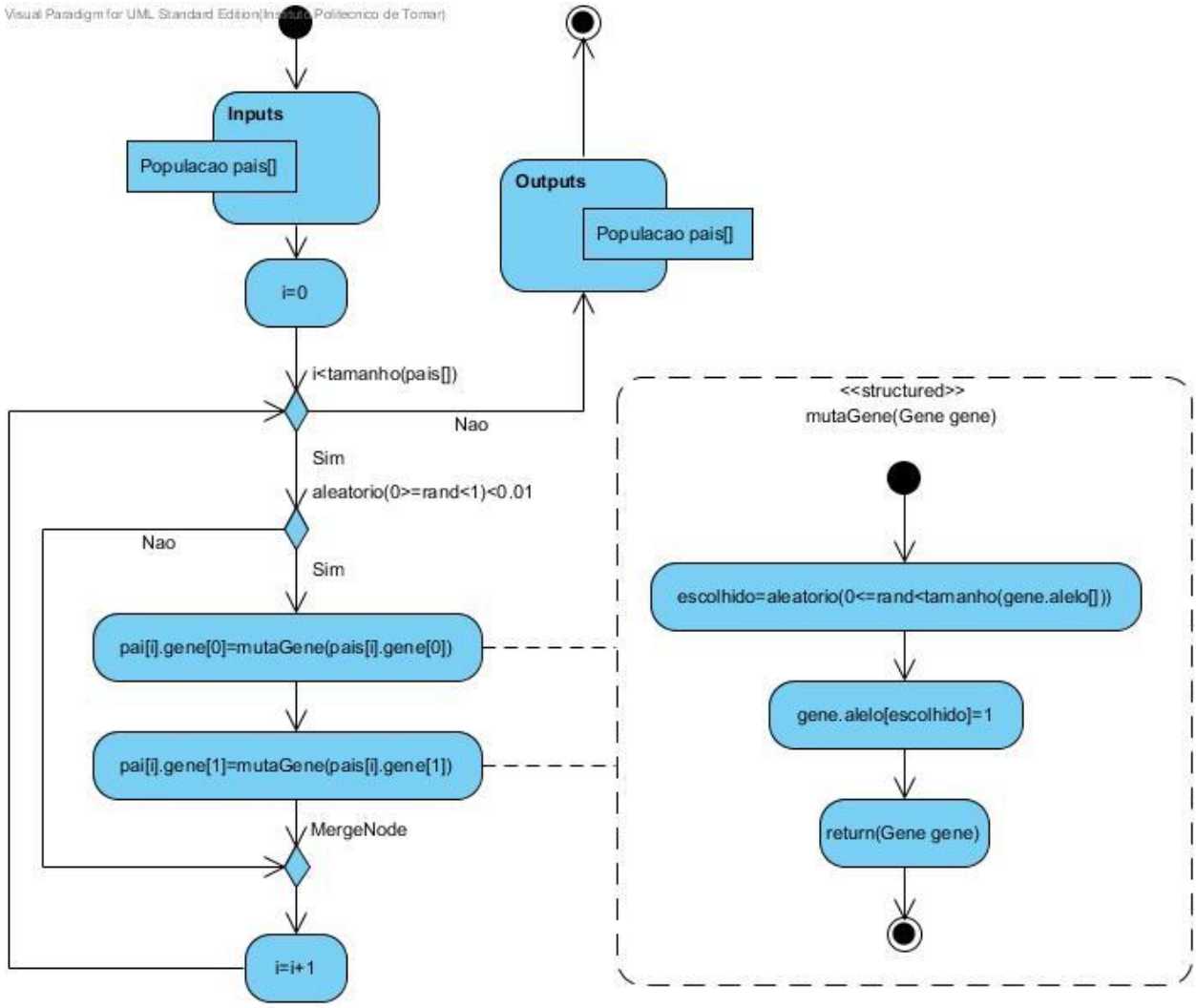
Mutation


Inputs
Populacao individuos[]
dimPop=tamanho(individuos[])
dimGene=tamanho(individuos[].genes[])
i=0
i==dimPop
Sim
Outputs
Populacao individuos[]
Nao
gene1=aleatorio(0<=rand<dimGene)
MergeNode
gene2=aleatorio(0<=rand<dimgene)
gene1==gene2
Sim
Nao
individuos[i]=troca(individuos[i],gene1,gene2)
i=i+1
<<structured>>
troca(Individuo pessoa,gene1,gene2)
geneTmp=pessoa.genes[gene1]
pessoa.genes[gene1]=pessoa.genes[gene2]
pessoa.genes[gene2]=geneTmp
return(Individuo pessoa)

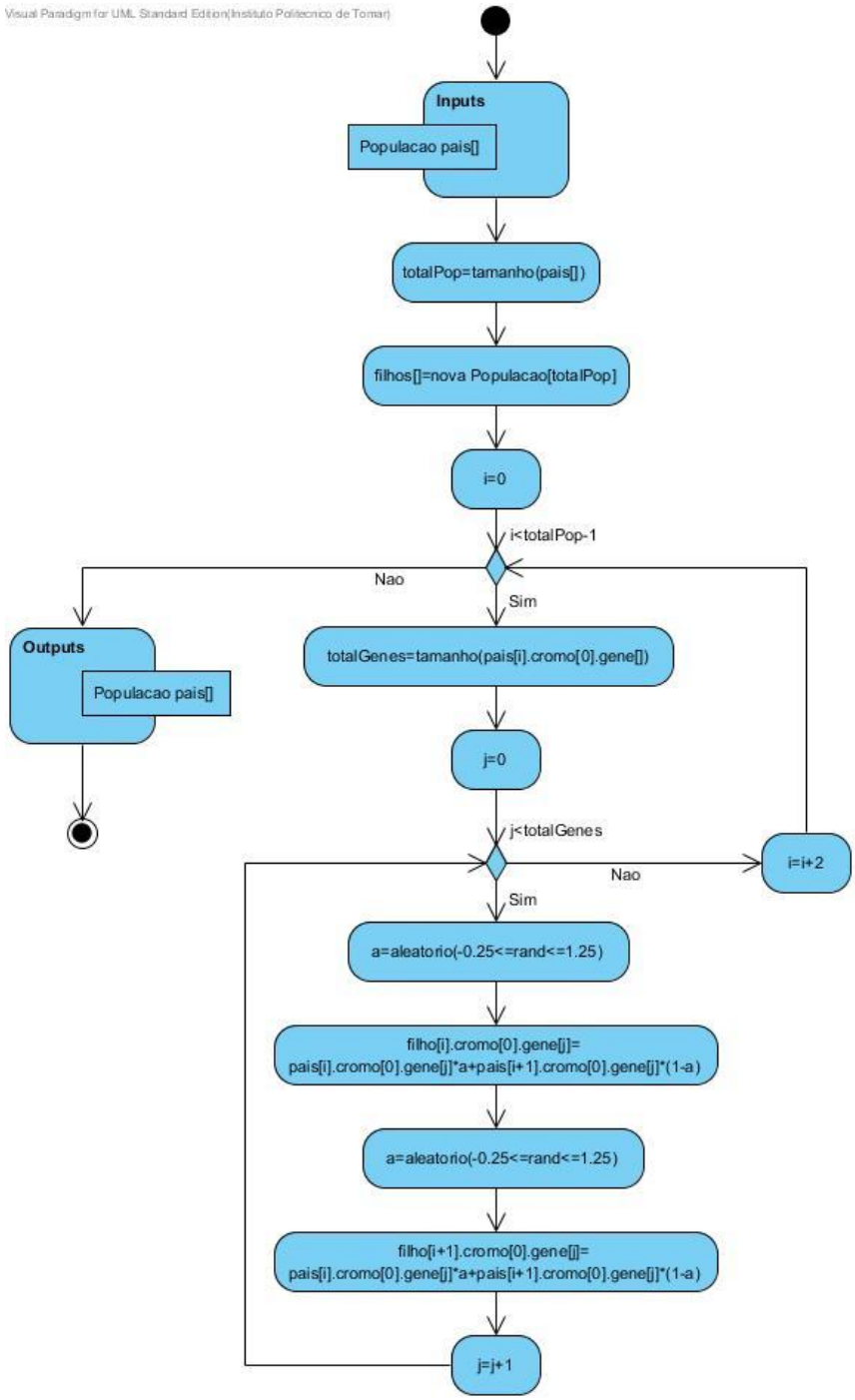
PMX


Inputs
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]
i=0
i< dimGene
Nao
Sim
Outputs
Individuo filho1
Individuo filho2
i>=corte1 && i<corte2
Nao
Sim
filho1.gene[i]=mae.gene[i]
filho1.gene[i]=procura(pai, mae, i, corte1, corte2)
filho2.gene[i]=pai.gene[i]
filho2.gene[i]=procura(mae, pai, i, corte1, corte2)
MergeNode
i=i+1
<<structured>>
procura(Individuo progenitor1, Individuo progenitor2, posicao, corte1, corte2)
i=corte1
i<corte2
Nao
return(progenitor1.gene[posicao])
Sim
progenitor1.gene[posicao]==progenitor2.gene[i]
Sim
Nao
i=i+1
return(Gene progenitor1.gene[i])

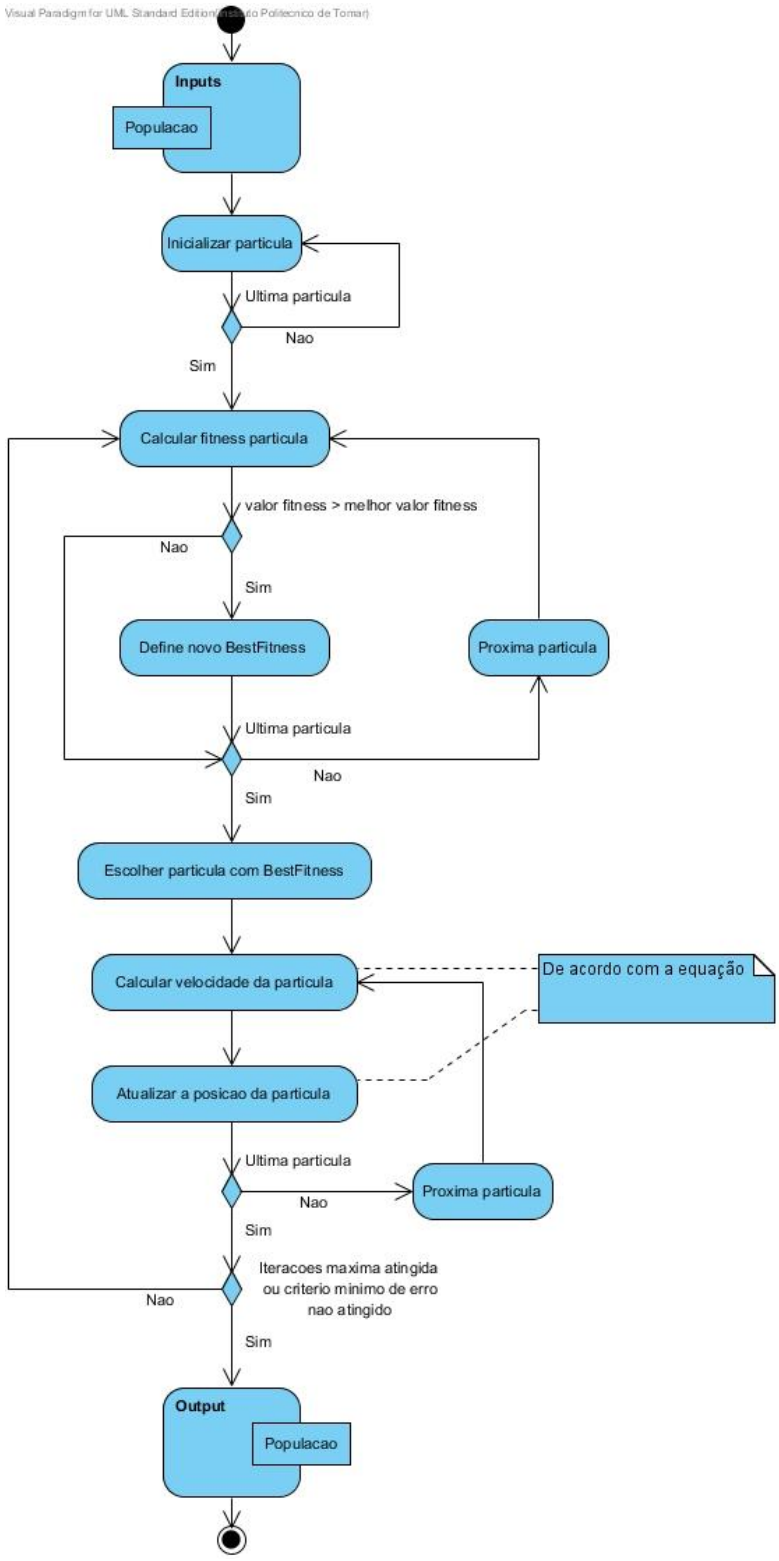
Crossover


Visual Paradigm for UML Standard Edition(Instituto Politecnico de Tomar)
Inputs
Populacao pais[]
i=0
i<tamanho(pais[])
Nao
Outputs
Populacao pais[]
Sim
aleatorio(0>=rand<1)<0.01
Nao
Sim
pai[i].gene[0]=mutaGene(pais[i].gene[0])
pai[i].gene[1]=mutaGene(pais[i].gene[1])
MergeNode
i=i+1
<<structured>>
mutaGene(Gene gene)
escolhido=aleatorio(0<=rand<tamanho(gene.alelo[]))
gene.alelo[escolhido]=1
return(Gene gene)

Mutação Binaria



P.S.O


Inputs
Populacao
Inicializar particula
Ultima particula
Nao
Sim
Calcular fitness particula
valor fitness > melhor valor fitness
Nao
Sim
Define novo BestFitness
Proxima particula
Ultima particula
Nao
Sim
Escolher particula com BestFitness
Calcular velocidade da particula
De acordo com a equação
Atualizar a posicao da particula
Ultima particula
Nao
Proxima particula
Sim
Iteracoes maxima atingida
ou criterio minimo de erro
nao atingido
Nao
Sim
Output
Populacao

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[])
```