**Instituto Politécnico de Tomar**



Índice

[Problema do caixeiro viajante 3](#_Toc326404299)

[Descrição 3](#_Toc326404300)

[Esquemas: 3](#_Toc326404301)

[Cálculo do Fitness: 4](#_Toc326404302)

[Reparar /Mitigar Programa 5](#_Toc326404303)

[Order Crossover 6](#_Toc326404304)

[Descrição 6](#_Toc326404305)

[Esquema 6](#_Toc326404306)

[SUS – Problema de Minimização 8](#_Toc326404307)

[Esquema 8](#_Toc326404308)

[Comparação de valores: 8](#_Toc326404309)

[SwapGenes: Mutação 9](#_Toc326404310)

[Descrição 9](#_Toc326404311)

[Esquemas 9](#_Toc326404312)

[Fluxogramas 10](#_Toc326404313)

[Crossover 10](#_Toc326404314)

[Invertion Mutation 11](#_Toc326404315)

[Mutation 13](#_Toc326404316)

[PMX 14](#_Toc326404317)

[Pseudo-codigo 15](#_Toc326404318)

[Crossover 15](#_Toc326404319)

[Invertion Mutation 16](#_Toc326404320)

[Mutation 16](#_Toc326404321)

[PMX 17](#_Toc326404322)

[SUS Minimização 17](#_Toc326404323)

# 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 delos**

**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]

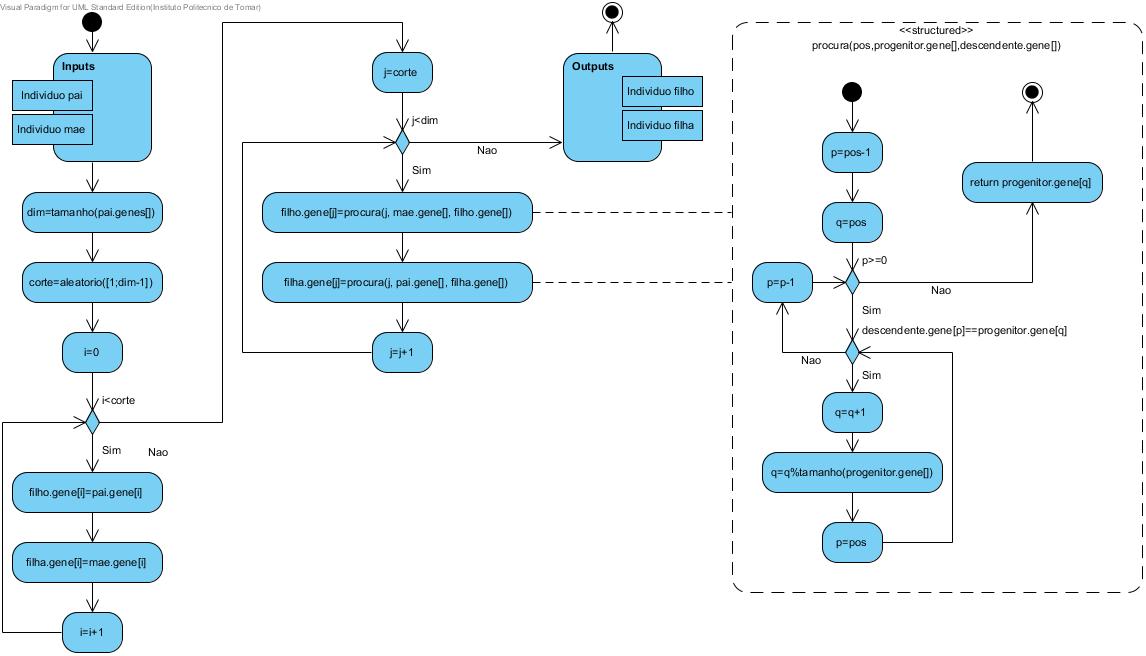
individuos [i] genes [alelo2]:. = aleloTmp

i: = i +1

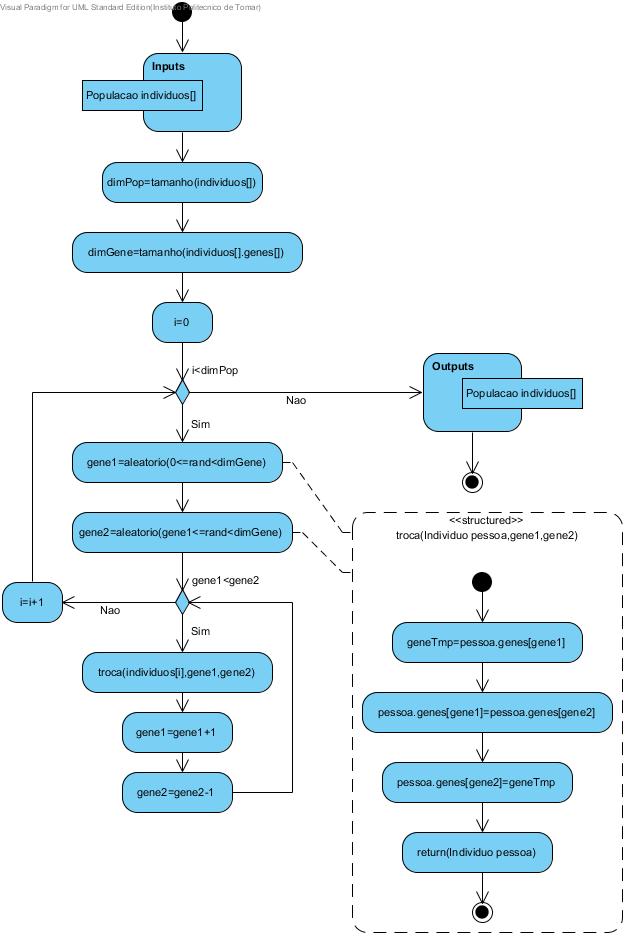
sai (População individuos [])

# Fluxogramas

## Crossover

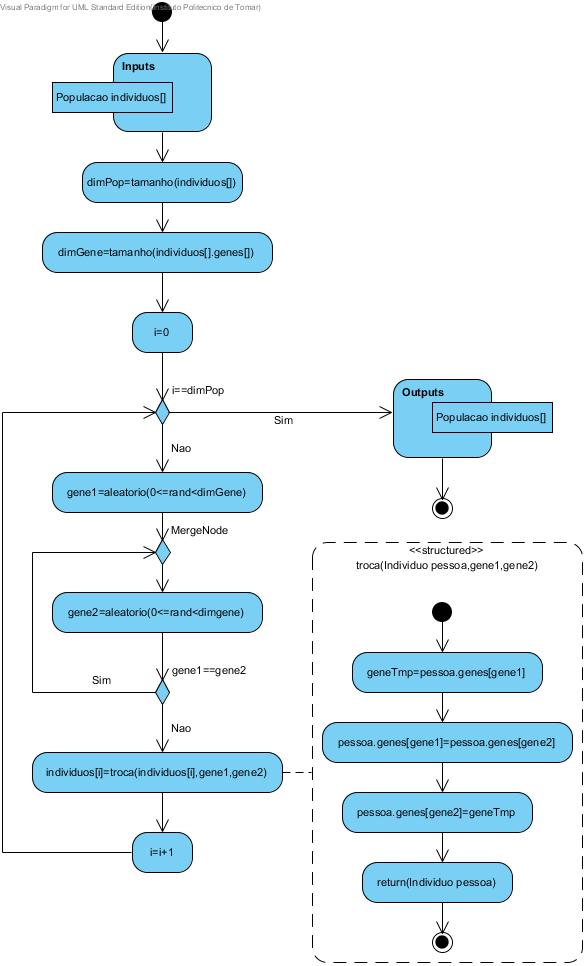
****

## Invertion Mutation

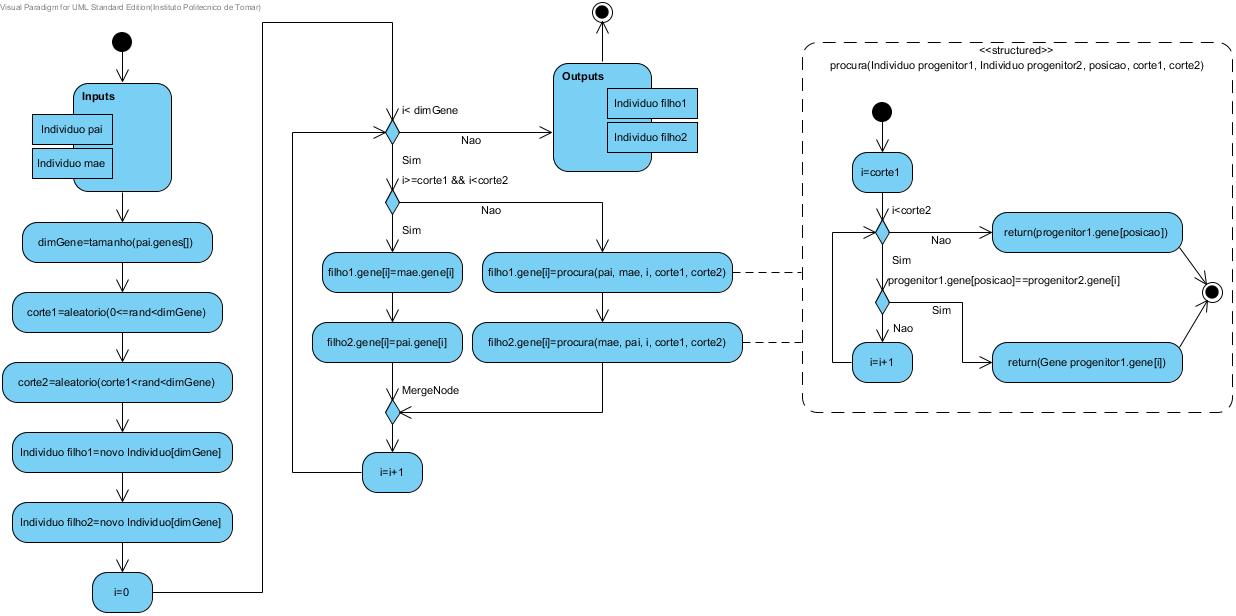
****

**.**

## Mutation

****

## PMX

****

# Pseudo-codigo

## 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**