# Introdução à Inteligência Artificial

16 JANEIRO

**Instituto Superior de Engenharia de Coimbra**
**Docente: Carlos Pereira**
**Realizado por: Francisco Almeida – 2020138795**
**Realizado por: Diogo Pinto - 2020133653**

isec Engenharia

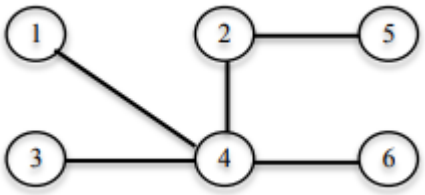
# Contents


Introdução ..... 3
Exemplo: ..... 3
Representação ..... 4
Algoritmo Pesquisa local ..... 5
Trepa colinas ..... 5
Testes ..... 6
Resultados ..... 6
Método 1 sem aceitar soluções de custo igual ..... 6
Método 1 a aceitar soluções de custo igual ..... 7
Método 2 sem aceitar soluções de custo igual ..... 7
Método 2 a aceitar soluções de custo igual ..... 8
Algoritmo de Pesquisa evolutiva ..... 9
Testes ..... 9
Resultados ..... 10
Recombinação 1 ponto + mutação binária – File1.txt ..... 10
Recombinação 1 ponto + mutação binária – File4.txt ..... 11
Recombinação 1 ponto + mutação customizada – File1.txt ..... 12
Recombinação 2 ponto + mutação binária – File1.txt ..... 13
Recombinação 2 ponto + mutação binária – File4.txt ..... 14
Hibrido 1 ..... 15
Testes ..... 15
Resultados ..... 15
Recombinação 1 ponto + mutação binária – File1.txt ..... 15
Recombinação 1 ponto + mutação binária – File4.txt ..... 16
Hibrido 2 ..... 17
Testes ..... 17
Resultados ..... 17
Recombinação 1 ponto + mutação binária – File1.txt ..... 17
Recombinação 1 ponto + mutação binária – File4.txt ..... 18
Conclusão ..... 19

# Introdução

Este trabalho consiste na implementação e análise de vários algoritmos de forma a obter soluções ao problema de Maximização do conjunto estável proposto no enunciado.
Este problema diz que tendo um grafo composto por um conjunto de vértices "**V**" e arestas ligadas entre si **"A"** podemos obter um conjunto estável "**S**". Denomina-se conjunto estável quando o conjunto de vértices de "**S**" não possuem uma ligação entre si. O objetivo do problema é conseguir obter um conjunto estável com o maior número de vértices.

## Exemplo:

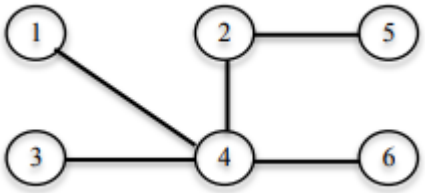



Para este grafo, um conjunto estável poderia ser o conjunto S={1,2,3,6}. Um exemplo de um conjunto estável inválido seria o conjunto S={1,2,3,4,6} pois o vértice 4 está ligado a todos os outros vértices.

# Representação

Existem duas possíveis representações deste problema. Uma lista de adjacências onde cada vértice tem associado a ele um array com todos os vértices a quem ele está ligado e uma matriz de adjacências onde cada linha e coluna da tabela representa os vértices de 1-n e onde a linha represente o vértice em questão e a coluna representa os vértices a quem ele pode estar ligado.

Esta matriz é representada por um conjunto de 0 e 1 onde 0 significa que o vértice não está ligado e 1 o contrário.




Voltando a esta imagem, uma matriz de adjacências dela seria:

| | **V1** | **V2** | **V3** | **V4** | **V5** | **V6** |
|---|---|---|---|---|---|---|
| **V1** | 0 | 0 | 0 | **1** | 0 | 0 |
| **V2** | 0 | 0 | 0 | **1** | **1** | 0 |
| **V3** | 0 | 0 | 0 | **1** | 0 | 0 |
| **V4** | **1** | **1** | **1** | 0 | 0 | **1** |
| **V5** | 0 | **1** | 0 | 0 | 0 | 0 |
| **V6** | 0 | 0 | 0 | **1** | 0 | 0 |

E uma lista de adjacências seria:

V1-[**V4**]
V2-[**V4,V5**]
V3-[**V4**]
V4-[**V1,V2,V3,V6**]
V5-[**V2**]
V6-[**V4**]

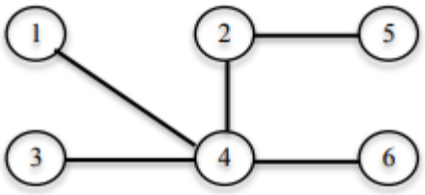
No nosso trabalho optamos por uma utilização de ambas as representações convertendo de uma para uma consoante necessário, normalmente utilizando a lista de adjacências para algoritmos de pesquisa local e a matriz para pesquisa evolutiva.

# Algoritmo Pesquisa local

## Trepa colinas

Optamos pelo uso do trepa colinas como algoritmo de pesquisa local. Este irá escolher um ponto(vértice) ao acaso e irá procurar outros pontos, também escolhidos ao acaso, para formar uma solução aceitando apenas soluções válidas, ou seja, se um ponto escolhido já estiver ligado com algum dos pontos na solução ele irá descartá-lo e procurar outro.

Em conjunto com este algoritmo utilizámos a representação de lista de adjacências o que lhe facilitou o trabalho pois poderia verificar logo se o vértice que escolheu estava ligado a algum dos escolhidos anteriormente.




Por exemplo, nesta imagem o algoritmo iria escolher um primeiro ponto. Vamos assumir que escolhia o vértice **1** (**S={1}**) de seguida ele ia repetir esse processo e escolher o **2**. Ele iria verificar se o **2** estava ligado ao **1** ou se ja tinha escolhido esse ponto, caso ambas as afirmações fossem falsas ele adicionaria **2** ao conjunto, S={1,2}.

Ele repete este processo n-vezes(depende do método escolhido) e no averigua a quantidade de vértices em "**S"** e vê se essa quantidade é superior à da solução anterior, neste caso 2 > 0. Logo a nova solução seria S={1,2} com cardinalidade(número de vértices no conjunto) 2.

## Testes

Testamos este algoritmo nos 6 ficheiros disponibilizados aceitando e rejeitando soluções de custo igual,variando o numero de iterações e mudando entre o método que corre **n_vezes = n_vertices * 2** e quando descarta um ponto, procura outro em vez de avançar no loop. E o método que corre **n_vezes = n_vertices * 20**(ele tinha resultados insatisfatórios por isso decidimos que faze-lo correr durante mais tempo era uma boa ideia), este método quando encontra um ponto que não corresponde simplesmente avança no loop perdendo uma iteração desnecessária.

## Resultados

### Tabela de refêrencia de soluções optimais

| Ficheiro | Vertices | Arestas | Best solution |
|---|---|---|---|
| teste.txt | 6 | 5 | 4 |
| file1.txt | 11 | 20 | 5 |
| file2.txt | 28 | 210 | 7 |
| file3.txt | 64 | 704 | 12 |
| file4.txt | 79 | 156 | 39 |
| file5.txt | 200 | 1534 | 18 |
| file6.txt | 300 | 10933 | 38 |

### Método 1 sem aceitar soluções de custo igual

| FICHEIRO | | 100 it | 1000 it | 5000 it | 10000 it |
|---|---|---|---|---|---|
| file1.txt | Melhor | 5 | 5 | 5 | 5 |
| | MBF | 5 | 5 | 5 | 5 |
| file2.txt | Melhor | 7 | 7 | 7 | 7 |
| | MBF | 7 | 7 | 7 | 7 |
| file3.txt | Melhor | 12 | 12 | 12 | 12 |
| | MBF | 12 | 12 | 12 | 12 |
| file4.txt | Melhor | 37 | 39 | 39 | 39 |
| | MBF | 36.6 | 38.2 | 38.5 | 38.9 |
| file5.txt | Melhor | 18 | 18 | 18 | 18 |
| | MBF | 18 | 18 | 18 | 18 |
| file6.txt | Melhor | 33 | 35 | 36 | 37 |
| | MBF | 31.3 | 33.2 | 34.7 | 35.3 |

Tendo a tabela de soluções optimais como referência conseguimos verificar que os quanto mais iterações utilizando este método mais próximos ficamos das soluções optimais.

### Método 1 a aceitar soluções de custo igual

| FICHEIRO | | 100 it | 1000 it | 5000 it | 10000 it |
|---|---|---|---|---|---|
| file1.txt | Melhor | 5 | 5 | 5 | 5 |
| | MBF | 5 | 5 | 5 | 5 |
| file2.txt | Melhor | 7 | 7 | 7 | 7 |
| | MBF | 7 | 7 | 7 | 7 |
| file3.txt | Melhor | 12 | 12 | 12 | 12 |
| | MBF | 12 | 12 | 12 | 12 |
| fila4.txt | Melhor | 38 | 39 | 39 | 39 |
| | MBF | 36.8 | 38.1 | 38.3 | 38.8 |
| file5.txt | Melhor | 18 | 18 | 18 | 18 |
| | MBF | 18 | 18 | 18 | 18 |
| file6.txt | Melhor | 31 | 35 | 36 | 36 |
| | MBF | 32.1 | 33.6 | 34.8 | 35.4 |

Aceitando soluções iguais conseguimos ver um pequeno aumento de na média das soluções no ficheiro 4 e 6 ,isto pode ter sido apenas sorte e a subida não é suficiente para declarar que aceitar soluções de custo igual melhorou o algoritmo, o que faria sentido tendo em conta que ele baseia-se na escolha de pontos ao acaso.

### Método 2 sem aceitar soluções de custo igual

| FICHEIRO | | 100 it | 1000 it | 5000 it | 10000 it |
|---|---|---|---|---|---|
| file1.txt | Melhor | 5 | 5 | 5 | 5 |
| | MBF | 5 | 5 | 5 | 5 |
| file2.txt | Melhor | 7 | 7 | 7 | 7 |
| | MBF | 7 | 7 | 7 | 7 |
| file3.txt | Melhor | 12 | 12 | 12 | 12 |
| | MBF | 12 | 12 | 12 | 12 |
| fila4.txt | Melhor | 38 | 38 | 39 | 39 |
| | MBF | 36.9 | 37.6 | 38.2 | 38.6 |
| file5.txt | Melhor | 18 | 18 | 18 | 18 |
| | MBF | 17.9 | 18 | 18 | 18 |
| file6.txt | Melhor | 38 | 34 | 37 | 37 |
| | MBF | 32.5 | 33.4 | 35.3 | 35.3 |

O segundo método aparenta atingir melhores resultados que o primeiro porém tem o problema de demorar mais a obter estes resultados.

### Método 2 a aceitar soluções de custo igual

| FICHEIRO | | 100 it | 1000 it | 5000 it | 10000 it |
|---|---|---|---|---|---|
| file1.txt | Melhor | 5 | 5 | 5 | 5 |
| | MBF | 5 | 5 | 5 | 5 |
| file2.txt | Melhor | 7 | 7 | 7 | 7 |
| | MBF | 7 | 7 | 7 | 7 |
| file3.txt | Melhor | 12 | 12 | 12 | 12 |
| | MBF | 12 | 12 | 12 | 12 |
| fila4.txt | Melhor | 39 | 39 | 39 | 39 |
| | MBF | 38.1 | 38.1 | 38.2 | 38.7 |
| file5.txt | Melhor | 18 | 18 | 18 | 18 |
| | MBF | 18 | 18 | 18 | 18 |
| file6.txt | Melhor | 33 | 36 | 36 | 36 |
| | MBF | 31.3 | 33.7 | 35.1 | 35.5 |

Aqui verificamos outra vez que aceitando soluções de igual valor pouco muda o resultado do algoritmo.

# Algoritmo de Pesquisa evolutiva

Para este algoritmo optamos pela utilização de uma matriz de adjacências. Ele gera uma população com tamanho "**popsize**"(no nosso caso a população é um conjunto de arrays todos a 0 pois verificamos que se utilizassemos arrays preenchidos por 0 e 1 ao acaso o algoritmo demorava muito para completar e obtia resultados piores). Após gerar a população ele entra num loop durante **n_gerações** onde irá utilizar diversos tipos de recombinação(1 ponto e 2 pontos) e de mutações(binária e customizada) para criar um conjunto de filhos que iram substituir a população que os criou. Ele repete isto até acabar dando como resultado a melhor solução de todas as gerações.

## Testes

Para este algoritmo utilizamos como base e primeiro teste o **algoritmo génetico com mutação binária e recombinação com 1 ponte de corte.** Variamos diversos valores como a probabilidade de recombinação e mutação e o número de gerações.
Realizamos mais 1 teste para recombinação 2 pontos de corte e mutação binária e outro para recombinação de 1 ponto de corte e mutação customizada.
A nossa "**mutação customizada**" é apenas preencher o filho criado com metade 1's e metade 0's e em seguida dar shuffle de tudo.

# Resultados

## Tabela de refêrencia de soluções optimais

| Ficheiro | Vertices | Arestas | Best solution |
|---|---|---|---|
| teste.txt | 6 | 5 | 4 |
| file1.txt | 11 | 20 | 5 |
| file2.txt | 28 | 210 | 7 |
| file3.txt | 64 | 704 | 12 |
| file4.txt | 79 | 156 | 39 |
| file5.txt | 200 | 1534 | 18 |
| file6.txt | 300 | 10933 | 38 |

## Recombinação  1 ponto + mutação binária – File1.txt

| | | tmo base+ mutação binária+ recombinação de um | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 5 | 4.6 |
| pop = 100 | pr = 0.5 | 5 | 4.7 |
| pm = 0.01 | pr = 0.7 | 5 | 4.6 |
| ger = 2500 | pm = 0.0 | 0 | 0 |
| pop = 100 | pm = 0.001 | 5 | 4 |
| | pm = 0.01 | 5 | 4.2 |
| pr = 0.7 | pm = 0.05 | 5 | 5 |
| pr = 0.7 | pop = 10 (ger = 25K) | 5 | 4.9 |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 5 | 5 |
| | pop = 100 (ger = 2.5K) | 0 | 0 |

Para o ficheiro 1 este algoritmo consegue obter o resultado optimal para a maioria das variações.

**Recombinação  1 ponto + mutação binária – File4.txt**

| | | tmo base+ mutação binária+ recombinação de um | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 38 | 37 |
| pop = 100 | pr = 0.5 | 38 | 36.4 |
| pm = 0.01 | pr = 0.7 | 38 | 36.6 |
| ger = 2500 | pm = 0.0 | 0 | 0 |
| pop = 100 | pm = 0.001 | 35 | 31 |
| | pm = 0.01 | 14 | 11.8 |
| pr = 0.7 | pm = 0.05 | 19 | 15.5 |
| pr = 0.7 | pop = 10 (ger = 25K) | 13 | 10.3 |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 17 | 14.4 |
| | pop = 100 (ger = 2.5K) | 0 | 0 |

Para o ficheiro 4 conseguimos observar que a variação de valores já afeta mais o algoritmo obtendo valores perto de optimais das primeiras linhas mas diminuindo ao diminuir o tamanho da população e a percentagem de mutação.

**Recombinação  1 ponto + mutação customizada – File1.txt**

| | | com mutação customizada | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 5 | |
| pop = 100 | pr = 0.5 | 5 | |
| pm = 0.01 | pr = 0.7 | 5 | |
| ger = 2500 | pm = 0.0 | 0 | |
| pop = 100 | pm = 0.001 | 1 | |
| | pm = 0.01 | 5 | |
| pr = 0.7 | pm = 0.05 | 5 | |
| pr = 0.7 | pop = 10 (ger = 25K) | 5 | |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 5 | |
| | pop = 100 (ger = 2.5K) | 5 | |

É possivel observar que utilizando a mutação customizada obtemos valores mais constantes do que a mutação binária para o ficheiro 1.

### Recombinação 1 ponto + mutação customizada – File4.txt

| Parâmetros Fixos | Parâmetros a variar | com mutação binária | |
|---|---|---|---|
| | | Best | MBF |
| ger = 2500 | pr = 0.3 | 8 | |
| pop = 100 | pr = 0.5 | 6 | |
| pm = 0.01 | pr = 0.7 | 7 | |
| ger = 2500 | pm = 0.0 | 0 | |
| pop = 100 | pm = 0.001 | 6 | |
| | pm = 0.01 | 9 | |
| pr = 0.7 | pm = 0.05 | 3 | |
| pr = 0.7 | pop = 10 (ger = 25K) | 0 | |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 7 | |
| | pop = 100 (ger = 2.5K) | 16 | |

Esta mutação falha em ficheiros que possuem uma maior quantidade de vértices como podemos observar pelos seus maus resultados no ficheiro 4.

### Recombinação 2 ponto + mutação binária – File1.txt

| Parâmetros Fixos | Parâmetros a variar | com recombinação 2 pontos | |
|---|---|---|---|
| | | Best | MBF |
| ger = 2500 | pr = 0.3 | 4 | |
| pop = 100 | pr = 0.5 | 4 | |
| pm = 0.01 | pr = 0.7 | 3 | |
| ger = 2500 | pm = 0.0 | 0 | |
| pop = 100 | pm = 0.001 | 4 | |
| | pm = 0.01 | 4 | |
| pr = 0.7 | pm = 0.05 | 5 | |
| pr = 0.7 | pop = 10 (ger = 25K) | 5 | |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 5 | |
| | pop = 100 (ger = 2.5K) | 5 | |

A utilização de uma recombinação de 2 pontos dá resultados semelhantes aos outros, podendo tirarem-se poucas conclusões.

**Recombinação  2 ponto + mutação binária – File4.txt**

| | | com recombinação | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 25 | |
| pop = 100 | pr = 0.5 | 30 | |
| pm = 0.01 | pr = 0.7 | 30 | |
| ger = 2500 | pm = 0.0 | 0 | |
| pop = 100 | pm = 0.001 | 32 | |
| | pm = 0.01 | 13 | |
| pr = 0.7 | pm = 0.05 | 14 | |
| pr = 0.7 | pop = 10 (ger = 25K) | 8 | |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 8 | |
| | pop = 100 (ger = 2.5K) | 15 | |

No ficheiro 4 já podemos verificar que a recombinação de 2 pontos aparenta dar resultados inferiores ao algoritmo incial porém mais constantes.

# Hibrido 1

Para o nosso modelo hibrido 1 utilizamos o trepa colinas para gerar um conjunto de soluções(população) que o algoritmo génetico depois irá utilizar e melhor(se possível).

## Testes

Os testes deste algoritmo vão ser baseados nos mesmos testes que o algoritmo génetico sofreu por fazer uso deste algoritmo, porém vamos utilizar o **método de recombinação de 1 ponto + mutação binária** apenas pois este foi o melhor de todos testes que fizemos ao algoritmo de pesquisa evolutiva

## Resultados

**Recombinação  1 ponto + mutação binária – File1.txt**

| | | Hibrido 1 | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 5 | 5 |
| pop = 100 | pr = 0.5 | 5 | 5 |
| pm = 0.01 | pr = 0.7 | 4 | 3.7 |
| ger = 2500 | pm = 0.0 | 3 | 2.2 |
| pop = 100 | pm = 0.001 | 5 | 5 |
| | pm = 0.01 | 4 | 3.8 |
| pr = 0.7 | pm = 0.05 | 5 | 4.8 |
| pr = 0.7 | pop = 10 (ger = 25K) | 5 | 5 |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 5 | 5 |
| | pop = 100 (ger = 2.5K) | 5 | 5 |

O algoritmo hibrido apresenta boas soluções para o ficheiro 1 mas dado o baixo número de vértices so no ficheiro 4 é que podemos tirar conclusões melhores.

**Recombinação  1 ponto + mutação binária – File4.txt**

| | | Hibrido 1 | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 17 | 16.6 |
| pop = 100 | pr = 0.5 | 18 | 18 |
| pm = 0.01 | pr = 0.7 | 31 | 31 |
| ger = 2500 | pm = 0.0 | 32 | 32 |
| pop = 100 | pm = 0.001 | 31 | 31 |
| | pm = 0.01 | 34 | 34 |
| pr = 0.7 | pm = 0.05 | 31 | 30.8 |
| pr = 0.7 | pop = 10 (ger = 25K) | 27 | 27 |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 31 | 30.8 |
| | pop = 100 (ger = 2.5K) | 30 | 31 |

Aqui já podemos ver que este algoritmo consegue obter boas soluções mas ainda não se aproxima do ótimo com consistência.

# Hibrido 2

Para este algoritmo utilizamos o algortimo de pesquisa evolutiva para gera uma solução e o de pesquisa local para utilizar essa solução como solução inicial e só aceitar futuras soluções que sejam melhores que essa.

## Testes

Os testes deste algoritmo vão ser baseados nos mesmos testes que o algoritmo génetico sofreu por fazer uso deste algoritmo, porém vamos utilizar o **método de recombinação de 1 ponto + mutação binária** apenas pois este foi o melhor de todos testes que fizemos ao algoritmo de pesquisa evolutiva

## Resultados

**Recombinação  1 ponto + mutação binária – File1.txt**

| | | Hibrido 2 | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 5 | 5 |
| pop = 100 | pr = 0.5 | 5 | 5 |
| pm = 0.01 | pr = 0.7 | 5 | 5 |
| ger = 2500 | pm = 0.0 | 0 | 0 |
| pop = 100 | pm = 0.001 | 5 | 5 |
| | pm = 0.01 | 5 | 5 |
| pr = 0.7 | pm = 0.05 | 5 | 5 |
| pr = 0.7 | pop = 10 (ger = 25K) | 5 | 5 |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 5 | 5 |
| | pop = 100 (ger = 2.5K) | 5 | 5 |

O algoritmo hibrido 2 aparenta ter soluções optimais e constantes porém vamos verificar o ficheiro 4.

**Recombinação  1 ponto + mutação binária – File4.txt**

| | | Hibrido 2 | |
|---|---|---|---|
| Parâmetros Fixos | Parâmetros a variar | Best | MBF |
| ger = 2500 | pr = 0.3 | 38 | 38 |
| pop = 100 | pr = 0.5 | 38 | 38 |
| pm = 0.01 | pr = 0.7 | 38 | 38 |
| ger = 2500 | pm = 0.0 | 0 | 0 |
| pop = 100 | pm = 0.001 | 38 | 38 |
| | pm = 0.01 | 38 | 38 |
| pr = 0.7 | pm = 0.05 | 38 | 38 |
| pr = 0.7 | pop = 10 (ger = 25K) | 39 | 38 |
| pm = melhor valor obtido | pop = 50 (ger = 5K) | 38 | 38 |
| | pop = 100 (ger = 2.5K) | 38 | 38 |

Podemos observar que o modelo hibrido 2 possui uma grande consistência e elevados resultados, sendo o melhor algoritmo de todos eles.

# Conclusão

É possivel concluir que de entre todos os algoritmos os algoritmos testes os melhores e por vezes até mais rápidos para resolver problemas deste género são os algoritmos de pesquisa local(trepa colinas) e o algoritmo hibrido 2(junção de algoritmo genético com o trepa colinas). O genético produz resultados muito inferiores, e talvez possa ser melhorado ao gerar uma melhor população e não apenas uma população de 0's, esta ideia é apoiada pelo algoritmo hibrido 1 que gera melhores resultados e a sua população advém de o algoritmo de pesquisa local.