Heurísticas
e
Meta-heurísticas

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior
Novembro de 2021

Definição

- Heurística:
  - 1. Arte de inventar ou descobrir.
  - 2. Método que pretende levar a inventar, descobrir ou a resolver problemas.

("heurística", in Dicionário Priberam da Língua Portuguesa, 2008-2021. Disponível em: <https://dicionario.priberam.org/heur%C3%Adstica>. Acesso em: 09 Nov. 2021)

- Meta-heurística:
  - Exprime a noção de transcendência;
  - Exprime a noção de reflexão sobre si;

("meta", in Dicionário Priberam da Língua Portuguesa, 2008-2021. Disponível em: <https://dicionario.priberam.org/meta>. Acesso em: 09 Nov. 2021)

  - Heurística sobre heurística.

Direitos Autorais

- Os *slides* a seguir foram desenvolvidos fundamentando-se na apostila disponibilizada na Internet por:
  - SOUZA, Marcone Jamilson Freitas. **Inteligência Computacional para Otimização**. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm>. Acesso em: 20 Out. 2021.

Introdução

- De acordo com Souza (2021), muitos problemas práticos são modelados da seguinte forma:
  - dado um conjunto S de variáveis discretas s (chamadas soluções) e uma função objetivo $f : S \leftarrow \mathbb{R}$, que associa cada solução $s \in S$ a um valor real $f(s)$. Encontre a solução $s^* \in S$, dita ótima. Para qual $f(s)$ tem o valor mais favorável - valor mínimo no caso do problema ter como objetivo a minimização de f ou o valor máximo no caso do problema ter como objetivo a maximização de f.

Introdução

- Grande parte desses problemas são combinatórios, sendo classificados na literatura como NP-difíceis e

- assim sendo, ainda não existem algoritmos que os resolvam em tempo polinomial.

Problema do Caixeiro Viajante

- O Problema do Caixeiro Viajante (PCV) é descrito por um conjunto de *n* cidades e uma matriz de distância entre elas, tendo o seguinte objetivo:
  - o caixeiro viajante deve sair de uma cidade, dita cidade origem e visitar cada uma das $n - 1$ cidades restantes apenas uma única vez. E retornar à cidade origem percorrendo a menor distância possível. Em outras palavras, deve ser encontrada uma rota fechada (ciclo hamiltoniano) de comprimento mínimo que passe exatamente uma única vez em cada cidade.

Problema do Caixeiro Viajante

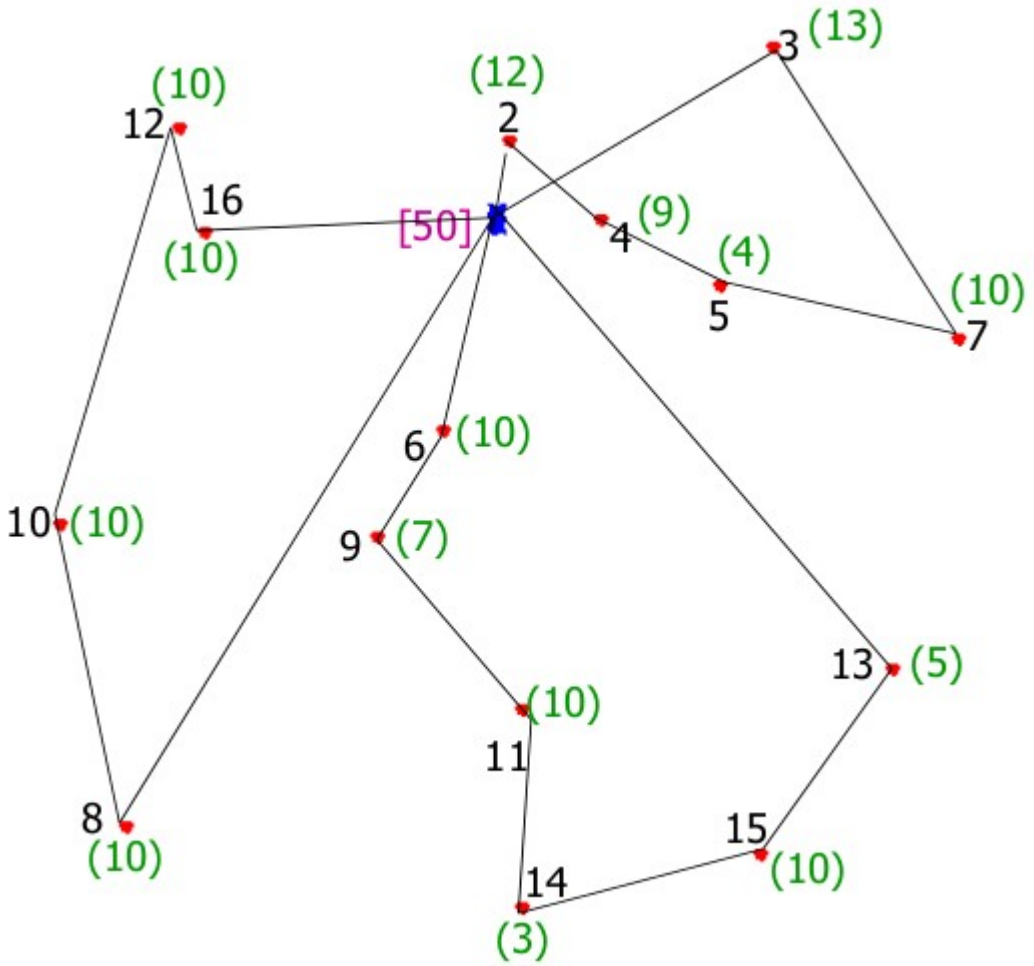


Imagem extraída de: Souza, Marcone J. F. e Penna, Puca H. V. Introdução. **Notas de aula de Técnicas Meta-heurísticas para Otimização Combinatória**. Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, 2021. Disponível em <www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/Introducao.pptx>. Acesso em: 09 Nov. 2021.

PCV – Dificuldade de Resolução

- Assuma que a distância de uma cidade *i* à outra cidade *j* seja simétrica, isto é, que $d_{ij}=d_{ji}$
- o número total de rotas possíveis é o equivalente a $(n-1)!/2$
- para se ter uma ideia da magnitude dos tempos envolvidos na resolução do PCV por enumeração completa de todas as possíveis soluções, para $n = 20$ tem-se $6*10^{16}$ rotas;
- um computador que avalia uma rota em 10^{-8} segundos gastaria cerca de 19 anos para encontrar a melhor rota e
- por fim: nos problemas da classe NP-difícil, não é possível garantir que a rota de custo mínimo seja encontrada em tempo polinomial.

Heurísticas

- De acordo com Souza(2021), define-se heurística como sendo uma técnica inspirada em processos intuitivos que procura uma boa solução a um custo computacional aceitável. Sem, no entanto, estar capacitada a garantir sua otimalidade e tampouco o quão próximo da solução ótima ela está;
- o desafio é produzir, em tempo reduzido, soluções tão próximas quanto possível da solução ótima.

Heurísticas

- Muitos esforços têm sido feitos no sentido de se produzir heurísticas que sejam eficientes para diversos problemas;
- entretanto, a maioria delas é muito específica para um problema particular.
- Em resumo: não são eficientes ou, em alguns casos, aplicáveis na resolução de uma classe mais ampla de problemas.

Heurística Construtiva (HC)

- Ainda conforme Souza (2021), uma HC tem por objetivo construir uma **solução**, elemento por elemento;
- a escolha de cada elemento a ser inserido, a cada passo, varia de acordo com a **função de avaliação** adotada; a qual variará de acordo com o problema abordado;
- nas heurísticas clássicas os elementos candidatos são geralmente ordenados segundo uma função gulosa, que estima o benefício da inserção de cada elemento e somente o “melhor” é inserido a cada passo.

Pseudocódigo Construção Gulosa

```
procedimento ConstrucaoGulosa (g(.),s)
  s ← ∅;
  inicialize conjunto C de elementos candidatos;
  enquanto(C ≠ ∅) faça
    g(t_melhor) = melhor{g(t)|t ∈ C};
    s ← s U {t_melhor};
    atualize conjunto C de elementos candidatos;
  fim enquanto
  retorne s;
fim
```

Legenda:
∅ → vazio

g(.) → função gulosa

Fonte: SOUZA, Marcone Jamilson Freitas. Inteligência Computacional para Otimização. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm>. Acesso em: 20 Out. 2021, p. 3.

Construindo uma Solução Inicial

- Variará de acordo com o tipo de problema abordado.
- Exemplo o **problema da mochila**:
  - existe uma mochila com capacidade b e um conjunto de n objetos que podem ser colocados na mochila. A cada objeto j está associado um peso w_j e um valor de retorno p_j (benefício). Considerando a existência de uma unidade de cada objeto; o objetivo é determinar o conjunto de objetos que devem ser colocados na mochila, de forma a maximizar o valor de retorno respeitando a capacidade da mochila.

Construindo uma Solução Inicial

- Seja uma mochila de capacidade $b = 23$ e os 5 objetos (j) da tabela a seguir, com os respectivos pesos (w) e benefícios (p):

Objeto (j)	1	2	3	4	5
Peso (w_j)	4	5	7	9	6
Benefício (p_j)	2	2	3	4	4

- para construirmos uma solução adicionamos à mochila, a cada passo, o objeto com maior benefício e que não ultrapasse a capacidade da mochila. Em caso de empate escolhe-se o objeto com menor peso. E reordena-se os objetos de acordo com este critério.

Construindo uma Solução Inicial

- Representando uma solução s por um valor binário de n posições:

Objeto (j)	5	4	3	1	2
Peso (w_j)	6	9	7	4	5
Benefício (p_j)	4	4	3	2	2

- passos:

1) adicionamos o objeto 5 que tem o maior benefício e, assim, $s=(00001)^t$ e $f(s)=4$. Peso da mochila=6
2) adicionamos o objeto 4 que tem o maior benefício entre os demais, assim, $s=(00011)^t$ e $f(s)=8$. Peso da mochila=15
3) adicionamos o objeto 3 que tem o maior benefício entre os demais, assim, $s=(00111)^t$ e $f(s)=11$. Peso da mochila=22
4) o objeto a ser alocado agora seria o 1. Porém, se o colocarmos na mochila o seu peso, somado aos dos objetos já colocados, ultrapassará a capacidade da mochila. Então deve-se tentar colocar o próximo objeto com o maior valor p_j ainda não analisado, que é o 2. Mas, este também ultrapassa a capacidade da mochila e como não há outros objetos, então temos como solução:

   $s^*=(00111)^t$ com $f(s^*)=11$

Construindo uma Solução Inicial

- Outra forma de se gerar uma solução inicial é escolhendo aleatoriamente os elementos candidatos;
- a cada passo na construção da solução inicial, o elemento a ser inserido é sorteado entre os demais.
- A vantagem desta metodologia está na simplicidade de implementação;
- a desvantagem é a baixa qualidade, em média, da solução produzida. O que exigirá maior esforço computacional na fase de refinamento.

Pseudocódigo Construção Aleatória

```
procedimento ConstrucaoAleatoria(s)
    s ← ∅;
    inicialize conjunto C de elementos candidatos;
    enquanto(C ≠ ∅) faça
    • escolha aleatoriamente {t_escolhido ∈ C};
      s ← s U {t_escolhido};
      atualize conjunto C de elementos candidatos;
    fim enquanto
    retorne s;
fim
```

Legenda:
∅ → vazio

Fonte: SOUZA, Marcone Jamilson Freitas. Inteligência Computacional para Otimização. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm>. Acesso em: 20 Out. 2021, p. 3.

HC's Utilizadas para o PCV

- Serão apresentadas três heurísticas construtivas utilizadas para o PCV:
  a) Heurística do Vizinho Mais Próximo,
  b) Heurística de Nemhauser e Bellmore e
  c) Heurística da Inserção Mais Barata.
- Para tanto tomaremos como verdade a seguinte tabela de cidades e suas distâncias:

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Heurística do Vizinho Mais Próximo

- Funcionamento:
  - de acordo com Souza(2021), parte-se da cidade origem e adiciona-se, a cada passo, a cidade `k` ainda não visitada cuja a distância é a menor possível;
  - a construção termina quando todas as cidades forem visitadas, situação na qual é feita a ligação entre a última cidade visitada e a origem.

Passo 1

Heurística do Vizinho Mais Próximo

- Considerando a tabela ao lado e a cidade `1` como origem, temos:

```
s = (1)
```

Acrescenta-se a cidade origem à sub-rota.

Cidade origem.

Cidade origem.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Passo 2

Heurística do Vizinho Mais Próximo

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1,3)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 1.

Como as cidades 3 e 6 possuem distâncias idênticas, então, qualquer uma das duas poderá ser escolhida.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidade já visitada e
- última cidade visitada.

Passo 3

Heurística do Vizinho Mais Próximo

- Considerando a tabela ao lado e a cidade `1` como origem, temos:

```
s = (1)
s = (1,3)
s = (1,3,4)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 3.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidades já visitadas e
- última cidade visitada.

Passo 4

Heurística do Vizinho Mais Próximo

- Considerando a tabela ao lado e a cidade `1` como origem, temos:

```
s = (1)
s = (1,3)
s = (1,3,4)
s = (1,3,4,5)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 4.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidades já visitadas e
- última cidade visitada.

Passo 5

Heurística do Vizinho Mais Próximo

- Considerando a tabela ao lado e a cidade `1` como origem, temos:

```
s = (1)
s = (1,3)
s = (1,3,4)
s = (1,3,4,5)
s = (1,3,4,5,6)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 5.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidades já visitadas e
- última cidade visitada.

Passo 6

Heurística do Vizinho Mais Próximo

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1,3)
s = (1,3,4)
s = (1,3,4,5)
s = (1,3,4,5,6)
s = (1,3,4,5,6,2)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 6.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

- ao final dos 6 passos temos a seguinte solução **s=(1 3 4 5 6 2)**.
- Distância percorrida:

$$dist = d_{13}+d_{34}+d_{45}+d_{56}+d_{62}+d_{21}$$

dist = 1+3+2+2+2+2 = 12

Legenda:
- cidades já visitadas e
- última cidade visitada.

Heurística de Bellmore e Nemhauser

- Funcionamento:
  - de acordo com Souza(2021), adiciona-se à rota corrente a cidade `k`, ainda não visitada, que esteja mais próxima dos extremos da sub-rota, ou seja, a cidade `k` se liga a uma cidade que esteja em uma das extremidades da sub-rota.
  - Exemplo:

```
cidades = (1 2 3 4 5 6)
rota ou solução = (5 1 4 2 6 3)
sub-rota = (5 1 4 2)
```

Cidade origem.

Contempla todas as cidades.

Rota em construção.

Extremo da sub-rota.

Outro extremo da sub-rota.

A próxima cidade a ser adicionada à sub-rota poderá ser colocada antes da 5 ou depois da 2.

Passo 1

Heurística de Bellmore e Nemhauser

- Considerando a tabela ao lado e a cidade `1` como origem, temos:

Acrescenta-se a cidade origem à sub-rota.

```
s = (1)
```

Cidade origem.

Cidade origem.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Passo 2

Heurística de Bellmore e Nemhauser

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
```

```
s = (1→3)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 1. Neste caso pode ser a 3 ou a 6..

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Como as cidades 3 e 6 possuem distâncias idênticas da única cidade que está nos extremos da sub-rota, então pode-se escolher uma ou outra.

Legenda:
- - cidade já visitada e
- - cidade nos extremos da sub-rota.

Passo 3

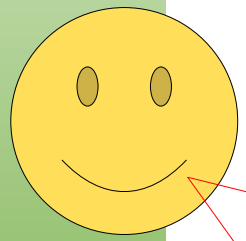
Heurística de Bellmore e Nemhauser

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (6→1→3)
```

Procura-se pela cidade que estiver mais próxima de um dos dois extremos da sub-rota.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0



A cidade mais próxima da cidade 1, extremo esquerdo da sub-rota, é a cidade 6, cuja distância é $\mathbf{d_{16}=1}$.

A cidade mais próxima da cidade 3, extremo direito da sub-rota, é a cidade 4, cuja distância é $\mathbf{d_{34}=3}$.

Tem-se então que $d_{16}<d_{34}$ e assim sendo a cidade **6** foi escolhida.

Legenda:
- - cidades já visitadas e
- - cidades nos extremos da sub-rota.

Passo 4

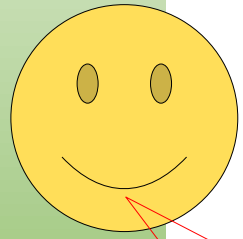
Heurística de Bellmore e Nemhauser

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (6→1→3)
s = (2→6→1→3)
```

Procura-se pela cidade que estiver mais próxima de um dos dois extremos da sub-rota, ou seja, 6 ou 3.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0



As cidades mais próximas da cidade 6, extremo esquerdo da sub-rota, são as cidades: 2 e 5, cuja distância de ambas, em relação a cidade 6, é $\mathbf{d_{6x}=2}$. Assim, pode-se escolher qualquer uma das duas. Então optou-se pela cidade 2.

A cidade mais próxima da cidade 3, extremo direito da sub-rota, é a cidade 4, cuja distância é $\mathbf{d_{34}=3}$.

Tem-se então que $d_{62}<d_{34}$ e assim sendo a cidade **2** foi escolhida.

Legenda:
- cidades já visitadas e
- cidades nos extremos da sub-rota.

Passo 5

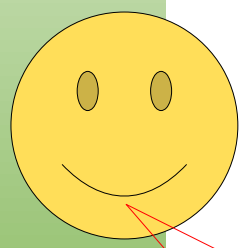
Heurística de Bellmore e Nemhauser

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (6→1→3)
s = (2→6→1→3)
s = (2→6→1→3→4)
```

Procura-se pela cidade que estiver mais próxima de um dos dois extremos da sub-rota, ou seja, 2 ou 3.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0



A cidade mais próxima da cidade 2, extremo esquerdo da sub-rota, é a cidade 5, cuja distância é $\mathbf{d_{25}=7}$.

A cidade mais próxima da cidade 3, extremo direito da sub-rota, é a cidade 4, cuja distância é $\mathbf{d_{34}=3}$.

Tem-se então que $d_{34}<d_{25}$ e assim sendo a cidade **4** foi escolhida.

Legenda:
- - cidades já visitadas e
- - cidades nos extremos da sub-rota.

Passo 6

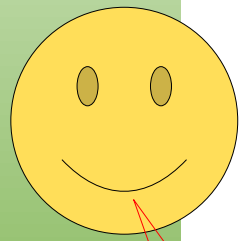
Heurística de Bellmore e Nemhauser

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (6→1→3)
s = (2→6→1→3)
s = (2→6→1→3→4)
s = (2→6→1→3→4→5)
```

Procura-se pela cidade que estiver mais próxima de um dos dois extremos da sub-rota, ou seja, 2 ou 4.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0



A cidade mais próxima da cidade 2, extremo esquerdo da sub-rota, é a cidade 5, cuja distância é $\mathbf{d_{25}=7}$.

A cidade mais próxima da cidade 4, extremo direito da sub-rota, é a cidade 5, cuja distância é $\mathbf{d_{45}=2}$.

Tem-se então que $d_{45}<d_{25}$ e assim sendo a cidade **5** foi escolhida, porém partindo-se da cidade 4.

Legenda:
- - cidades já visitadas e
- - cidades nos extremos da sub-rota.

Heurística de Bellmore e Nemhauser

- E assim temos como rota ou solução:

  s = (2→6→1→3→4→5)

- Distância percorrida:

  $dist = d_{26}+d_{61}+d_{13}+d_{34}+d_{45}+d_{52}$

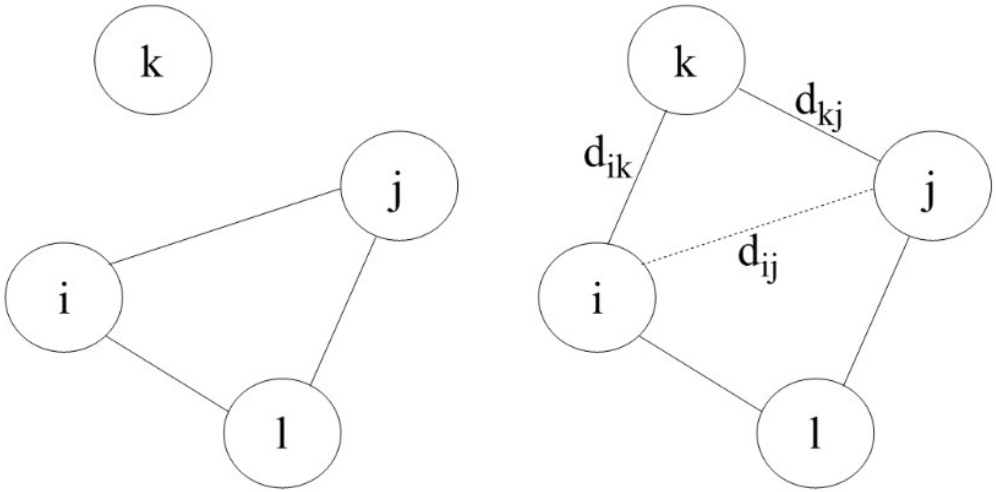
  dist = $2+1+1+3+2+7$ **= 16**

Heurística da Inserção Mais Barata

- Funcionamento:
  - segundo Souza(2021), parte-se de uma sub-rotina inicial envolvendo três cidades e a cada passo adiciona-se uma cidade `k`, ainda não visitada, entre as cidades `i` e `j` da sub-rota cujo custo de inserção s^k_{ij}, dado pela fórmula abaixo, seja o menor possível.
  - Fórmula:

$$s^k_{ij} = d_{ik} + d_{kj} - d_{ij}$$

As figuras a seguir ilustram a inserção da cidade k entre as cidades i e j.



(a) Antes da inserção (b) Depois da inserção

Figura extraída de SOUZA, Marcone Jamilson Freitas. Inteligência Computacional para Otimização. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm>. Acesso em: 20 Out. 2021, p. 7.

Heurística da Inserção Mais Barata

- Observação:
  - a sub-rota inicial pode ser formada por um procedimento construtivo qualquer.
- Exemplo:
  - parta da cidade origem e adicione à sub-rota a cidade mais próxima. Então, considerando as duas extremidades (cidade origem e última inserida à sub-rota), adicione a cidade ainda não visitada cuja soma das distâncias às duas extremidades seja a menor.

Passo 1

Heurística da Inserção Mais Barata

- Considerando a tabela ao lado e a cidade `1` como origem, temos:

```
s = (1)
```

Acrescenta-se a cidade origem à sub-rota.

Cidade origem.

Cidade origem.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Passo 2

Heurística da Inserção Mais Barata

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
```

Procura-se pela cidade, ainda não visitada, mais próxima da cidade 1.

Como as cidades 3 e 6 possuem distâncias idênticas, então, qualquer uma das duas poderá ser escolhida.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- - cidade já visitada e
- - extremos da sub-rota.

Passo 3

Heurística da Inserção Mais Barata

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (1→3→2)
```

Aplica-se a fórmula $s^k_{ij}=d_{ik}+d_{kj}-d_{ij}$ para descobrir qual das cidades, ainda não visitadas, possui a menor distância.

Como pode-se ver nas contas abaixo, as cidades 2, 4 e 6 empataram nas distâncias. Assim sendo, qualquer uma poderá ser utilizada. Neste caso escolheu-se a cidade 2.

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

$s^2_{13}=d_{12}+d_{23}-d_{13}$ $s^2_{13}=2+5-1=6$	$s^4_{13}=d_{14}+d_{43}-d_{13}$ $s^4_{13}=4+3-1=6$
$s^5_{13}=d_{15}+d_{53}-d_{13}$ $s^5_{13}=9+8-1=16$	$s^6_{13}=d_{16}+d_{63}-d_{13}$ $s^6_{13}=1+6-1=6$

Legenda:
- - cidades já visitadas e
- - extremos da sub-rota.

Passo 4

Heurística da Inserção Mais Barata

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (1→3→2)
s = (1→3→2→6)
```

Na tabela à esquerda (roxa), mostrou-se o resultado dos cálculos das distâncias entre as cidades existentes (ainda não visitadas) e as demais que já se encontram na sub-rota. Nota-se que quem tem o menor custo de inserção é s^6_{21}. Portanto a cidade 6 foi inserida entre as cidades 2 e 1.

i	k	j	s^k_{ij}	$d_{ik}+d_{kj}-d_{ij}$	=
1	4	3	s^4_{13}	4+3-1	6
1	5	3	s^5_{13}	9+8-1	16
1	6	3	s^6_{13}	1+6-1	6
3	4	2	s^4_{32}	3+9-5	7
3	5	2	s^5_{32}	8+7-5	10
3	6	2	s^6_{32}	6+2-5	3
2	4	1	s^4_{21}	9+4-2	11
2	5	1	s^5_{21}	7+9-2	14
2	6	1	s^6_{21}	2+1-2	1

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidades já visitadas.

Passo 5

Heurística da Inserção Mais Barata

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

  s = (1)
  s = (1→3)
  s = (1→3→2)
  s = (1→3→2→6)
  s = (1→4→3→2→6)

Na tabela à esquerda (roxa), mostrou-se o resultado dos cálculos das distâncias entre as cidades existentes (ainda não visitadas) e as demais que já se encontram na sub-rota. Nota-se que quem tem o menor custo de inserção é s^4_{13}. Portanto a cidade 4 foi inserida entre as cidades 1 e 3.

i	k	j	s^k_{ij}	$d_{ik}+d_{kj}-d_{ij}$	=
1	4	3	s^4_{13}	4+3-1	6
1	5	3	s^5_{13}	9+8-1	16
3	4	2	s^4_{32}	3+9-5	7
3	5	2	s^5_{32}	8+7-5	10
2	4	6	s^4_{26}	9+5-2	12
2	5	6	s^5_{26}	7+2-2	7
6	4	1	s^4_{61}	5+4-1	8
6	5	1	s^5_{61}	2+9-1	10

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidades já visitadas.

Passo 6

Heurística da Inserção Mais Barata

- Considerando a tabela ao lado e a cidade 1 como origem, temos:

```
s = (1)
s = (1→3)
s = (1→3→2)
s = (1→3→2→6)
s = (1→4→3→2→6)
s = (1→5→4→3→2→6)
```

Após os cálculos com a última cidade, 5, nota-se que quem tem o menor custo de inserção são: s^5_{14}, s^5_{43} e s^5_{26}. Como houve o empate então escolhe-se um. E para esta demonstração foi escolhido s^5_{14}. Assim sendo, a cidade 5 foi inserida entre as cidades 1 e 4 e a rota ficou completa.

i	k	j	s^k_{ij}	$d_{ik}+d_{kj}-d_{ij}$	=
1	5	4	s^5_{14}	9+2-4	7
4	5	3	s^5_{43}	2+8-3	7
3	5	2	s^5_{32}	8+7-5	10
2	5	6	s^5_{26}	7+2-2	7
6	5	1	s^5_{61}	2+9-1	10

Cidade	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	5
5	9	7	8	2	0	2
6	1	2	6	5	2	0

Legenda:
- cidades já visitadas.

Heurísticas de Refinamento

- De acordo com Souza(2021), também são conhecidas como **técnicas de busca local**;
- são constituídas de técnicas de pesquisa fundamentadas na noção de vizinhança.
- Seja S o espaço de busca de um problema de otimização e f a função objetivo a minimizar;
- a função N, a qual depende da estrutura do problema tratado, associa a cada solução $s \in S$, sua vizinhança $N(S) \subseteq S$.
- Cada solução $s' \in N(s)$ é chamada de vizinho de s.
- Denomina-se movimento, a modificação m que transforma uma solução s em outra s' que esteja em sua vizinhança.

Heurísticas de Refinamento

- Ainda conforme Souza(2021), esta classe de heurísticas parte de uma **solução inicial** e caminha, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança estabelecida.
- Lembrando que uma solução inicial pode ser obtida por uma **heurística construtiva** ou então gerada aleatoriamente.
- A definição de vizinhança é fundamental em uma heurística de refinamento. Ou seja, de uma solução *s* do espaço de soluções deve ser sempre possível atingir outra solução; isto, com um número finito de passos utilizando um ou mais tipos de movimentos.

Heurísticas de Refinamento

- Exemplo:
  - considere no problema da mochila as soluções $s^{(1)}=(01001)^t$ e $s^{(2)}=(11010)^t$ do espaço de soluções;
  - com o movimento $m=\{$*trocar o valor de um bit*$\}$ é possível navegar no espaço de soluções do problema de $s^{(1)}$ a $s^{(2)}$;
  - observe! Com esse movimento *m* podemos percorrer o seguinte caminho: $s^{(1)}=(01001)^t \rightarrow (11001)^t \rightarrow (11011)^t \rightarrow (11010)^t=s^{(2)}$.
- No entanto, se definíssemos *m* como sendo a troca de 2 bits simultaneamente, então não conseguiríamos chegar a $s^{(2)}$ a partir de $s^{(1)}$.

Heurísticas de Refinamento

- Souza(2021) afirma que em muitos problemas de combinatórios é difícil, até mesmo, encontrar uma solução viável;
- nesta situação é má ideia caminhar pelo espaço de busca de soluções viáveis do problema considerado.
- Para tais problemas, o espaço de busca pode incluir soluções inviáveis, obtidas a partir do relaxamento de algumas restrições do problema original. Para tanto, basta acrescentar à função de avaliação, componentes que penalizem violações de restrições.

Continua...

Heurísticas de Refinamento

- Um exemplo típico é o problema de Programação de Horários de Cursos Universitários (*Course Timetabling Problem*);
- onde a restrição principal deste problema requer que as aulas lecionadas pelo mesmo Professor para turmas distintas, não se sobreponham. Isto é, que não sejam realizadas no mesmo horário.
- Considerando como movimento *m* a mudança das aulas de um curso de um horário para outro, dificilmente geraríamos quadros de horários sem situações de sobreposição.
- Relaxando-se as restrições e penalizando-as na função de avaliação, torna-se muito mais eficiente a exploração do espaço de busca, isto de acordo com Hertz(1992) *apud* Souza(2021).

Heurísticas de Refinamento

- Exemplo:
  - como gerar diferentes estruturas de vizinhanças:
    - vamos considerar o PCV, para o qual representamos uma solução s por um vetor de n posições, sendo que em cada posição i tem-se a cidade s_i
    - com o movimento m de troca de posição entre duas cidades definimos a estrutura de vizinhança $N^{(T)}$. Assim $s=(s_1\ s_2\ s_3\ \ldots\ s_n)^t$ tem como vizinhos em $N^{(T)}(s)$ as seguintes soluções:
      - $s^{(1)}=(\mathbf{s_2\ s_1}\ s_3\ \ldots\ s_n)^t$,
      - $s^{(2)}=(\mathbf{s_3}\ s_2\ \mathbf{s_1}\ \ldots\ s_n)^t$,
      - $s^{(n-1)}=(\mathbf{s_n}\ s_1\ s_3\ \ldots\ \mathbf{s_1})^t$,
      - $s^{(n)}=(s_1\ \mathbf{s_3\ s_2}\ \ldots\ s_n)^t$, ... $s^{(n*(n-1)/2)}=(s_1\ s_2\ s_3\ \ldots\ \mathbf{s_n\ s_{n-1}})^t$.
    - por outro lado, considerando como movimento m a realocação de uma cidade de uma posição na sequência de visita para outra, definimos a estrutura de vizinhança $N^{(R)}$.

continua...

Heurísticas de Refinamento

Nesta estrutura, $N^{(R)}$, são de $s=(s_1\ s_2\ s_3\ \ldots\ s_{n-1}\ s_n)^t$ as seguintes soluções:

- $s^{(1)}=(s_2\ \mathbf{s_1}\ s_3\ \ldots\ s_{n-1}\ s_n)^t$,
- $s^{(2)}=(s_2\ s_3\ \mathbf{s_1}\ \ldots\ s_{n-1}\ s_n)^t$,
- ...
- $s^{(n-2)}=(s_2\ s_3\ \ldots\ s_{n-1}\ \mathbf{s_1}\ s_n)^t$,
- $s^{(n-1)}=(s_2\ s_3\ \ldots\ s_{n-1}\ s_n\ \mathbf{s_1})^t$,
- $s^{(n)}=(s_1\ s_3\ \mathbf{s_2}\ \ldots\ s_{n-1}\ s_n)^t$,
- $s^{(n+1)}=(s_1\ s_3\ s_4\ \mathbf{s_2}\ldots\ s_{n-1}\ s_n)^t$,
- ...
- $s^{(2n-4)}=(s_1\ s_3\ s_4\ \ldots\ s_{n-1}\ \mathbf{s_2}\ s_n)^t$,
- $s^{(2n-3)}=(s_1\ s_3\ s_4\ \ldots\ s_{n-1}\ s_n\ \mathbf{s_2})^t$,
- ...
- $s^{(2n-5)}=(s_1\ \mathbf{s_3}\ s_2\ s_4\ \ldots\ s_{n-1}\ s_n)^t$,
- ...
- $s^{(n-2)\hat{}2*(n-1)}=(s_1\ s_2\ s_3\ s_4\ \ldots\ \mathbf{s_n}\ s_{n-1})^t$.

continua...

Heurísticas de Refinamento

Podemos também, definir como vizinhança de uma solução *s* o conjunto de vizinhos gerados tanto por movimentos de troca quanto por movimentos de realocação. Isto é,

$N(s)=N^{(T)}(s) \cup N^{(R)}(s)$

- existem outros movimentos mais elaborados, por exemplo o *Or*, que consiste em realocar um bloco contíguo de cidades em outra posição da sequência.
- Exemplo:
  - considerando a solução $s = (4\ 3\ 1\ 2)^t$ e blocos de tamanho 2, teríamos os seguintes vizinhos para *s*:
    - $s'_1=(1\ \mathbf{4}\ \mathbf{3}\ 2)^t$,
    - $s'_2=(4\ 2\ \mathbf{3}\ \mathbf{1})^t$ e
    - $s'_3=(4\ \mathbf{1}\ \mathbf{2}\ 3)^t$.

Heurísticas de Refinamento

- **Método da Descida/Subida** (*Uphill Method*)
  - De acordo com Souza(2021), a ideia desta técnica é partir de uma solução inicial qualquer e a cada passo analisar todos os seus possíveis vizinhos, movendo somente para aquele que representar uma melhora no valor atual da função de avaliação;
  - pelo fato de analisar todos os vizinhos e escolher o melhor, esta técnica é comumente referenciada na literatura inglesa por *Best Improvement Method (BI)*.
  - Este método para quando um ótimo local é encontrado.

Heurísticas de Refinamento

- O pseudocódigo a seguir é o do **Método de Descida** aplicado à minimização de uma função de avaliação *f* a partir de uma solução inicial *s* e considerando a busca em uma dada vizinhança *N(.)*.

```
procedimento Descida(f(.), N(.), s)
  V = {s' ∈ N(s) | f(s') < f(s)};
  enquanto (|V| > 0) faça
    Selecione s' ∈ V onde s'=arg min{f(s') | s' ∈ V };
    s ← s';
    V = {s' ∈ N(s) | f(s') < f(s)};
  fim enquanto;
  retorne s;
fim procedimento;
```

Heurísticas de Refinamento

- **Método da Primeira Melhora**
  - O método de descida/subida requer a exploração de toda a vizinhança;
  - já o Método da Primeira Melhora (*First Improvement Method* – FI), de acordo com Souza(2021), evita esta pesquisa exaustiva e para a busca na vizinhança quando um vizinho melhor for encontrado.
- **Método de Descida/Subida Randômica**
  - Como já mostrado, o Método de Descida/Subida requer a exploração de toda a vizinhança;
  - outro método alternativo, que também evita esta pesquisa exaustiva, é o Método da Descida/Subida Randômica (*Random Descent/Uphill Method*).
  - Este método consiste em analisar um vizinho qualquer e o aceitar somente se ele for estritamente melhor que a solução corrente. Se não for, então a solução corrente permanece inalterada e outro vizinho é gerado.

continua...

Heurísticas de Refinamento

- O procedimento é interrompido após um número fixo de iterações sem melhora no valor da melhor solução obtida.
- A seguir é mostrado o pseudocódigo do Método de Descida Randômica aplicado ao refinamento de uma solução *s* em um problema de minimização de uma função *f(.)*, utilizando uma estrutura de vizinhança *N(.)*. Neste pseudocódigo `IterMax` representa o número máximo de iterações sem melhora no valor da função de avaliação.

```
procedimento DescidaRandomica(f(.), N(.), IterMax, s)
  Iter ← 0;  { Contador de iterações sem melhora }
  enquanto (Iter < IterMax) faça
    Iter ← Iter + 1 ;
    Selecione aleatoriamente s' ∈ N(s);
    se (f(s') < f(s)) então
      Iter ← 0;
      s ← s';
    fim se
  fim enquanto
  retorne s;
fim procedimento
```

Heurísticas de Refinamento

- **Métodos: Não Ascendente Randômico e Não Descendente Randômico**
  - Os Métodos: Não Ascendente Randômico (RNA) e Não Descendente Randômico (RND) são variantes dos Métodos de Descida/Subida Randômica. Eles diferem destes últimos por aceitarem vizinhos gerados aleatoriamente, se este for melhor ou igual a solução corrente;
  - os métodos param após um número fixo de iterações sem melhora no valor da melhor solução.

Heurísticas de Refinamento

- **Método de Descida em Vizinhança Variável**
  - Segundo Souza (2021) o Método de Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND) é um método de refinamento que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança, aceitando somente soluções de melhora da solução corrente e retornando à primeira estrutura quando uma solução melhor for encontrada;
  - o pseudocódigo deste método, em que se considera o refinamento de uma solução *s* utilizando uma função de avaliação *f*, a ser minimizada e um conjunto de *r* diferentes vizinhanças $N=\{N^{(1)}, N^{(2)}, \ldots, N^{(r)}\}$ é mostrado no próximo *slide*.
  - Dependendo do problema abordado, a busca pelo melhor vizinho (linha 5 do pseudocódigo) pode ser cara computacionalmente. Nesta situação é comum fazer a busca pela primeira solução de melhora. Outra alternativa é considerar a exploração apenas em um certo percentual da vizinhança. E uma terceira alternativa, muito utilizada, é aplicar o Método de Descida Randômica para cada vizinhança explorada.

continua...

Heurísticas de Refinamento

- o pseudocódigo:

```
procedimento VND (f(.), N(.), r, s)
  Seja r o número de estruturas diferentes de vizinhança;
  k ← 1; {Tipo de estrutura de vizinhança corrente}
  enquanto (k <= r) faça
    Encontre o melhor vizinho s' ∈ N(k)(s);
    se (f(s') < f(s)) então
      s ← s';
      k ← 1;
    senão
      k ← k + 1;
    fim se
  fim enquanto
  retorne s;
fim procedimento
```

Meta-heurísticas

- De acordo com Ribeiro (1996) *apud* Souza (2021), as meta-heurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima;
- consiste na aplicação, a cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico.

Meta-heurísticas

- Diferentemente das heurísticas convencionais, as meta-heurísticas são de caráter geral e providas de mecanismos para evitar que fiquem presas em ótimos locais;
- as meta-heurísticas diferenciam-se entre si, basicamente pelo mecanismo usado para sair das armadilhas dos ótimos locais.
- **Elas se dividem em duas categorias:**
  - as de busca local e
  - as de busca populacional.

Meta-heurísticas de Busca Local

- Neste tipo de meta-heurística a exploração do espaço de busca é feita por meio de movimentos, os quais são aplicados a cada pessoa sobre a solução corrente, gerando outra solução promissora em sua vizinhança.
- Exemplos:
  - Busca Tabu,
  - Simulated Annealing,
  - Variable Neighborhood Search e
  - Iterated Local Search.

Meta-heurísticas de Busca Populacional

- Os métodos baseados em busca populacional, por sua vez, consistem em manter um conjunto de boas soluções e combiná-las de forma a tentar produzir soluções ainda melhores.
- Exemplos:
  - Algoritmo Genético,
  - Algoritmo Memético e o
  - Algoritmo Colônia de Formigas.

Algoritmo Genético

- Segundo Souza (2021), trata-se de uma meta-heurística que se fundamenta em uma analogia com processos naturais de evolução, onde dada uma população:
  - os indivíduos com características genéticas melhores, têm maiores chances de sobrevivência e de produzirem filhos cada vez mais aptos e
  - os indivíduos menos aptos tendem a desaparecer.
- Foi proposto por John Holland na década de 1970 (Hansen (1986) *apud* Souza (2021)).

Algoritmo Genético

- Num Algoritmo Genético (AG), cada cromossomo (indivíduo da população) está associado a uma solução do problema e cada gene está associado a um componente da solução;
- um alelo está associado a um valor que cada componente da solução pode assumir;
- o mecanismo de reprodução, baseado em processos evolutivos, é aplicado sobre a população com o objetivo de explorar o espaço de busca e encontrar melhores soluções para o problema;
- cada indivíduo é avaliado por uma função de aptidão que mensura seu grau de adaptação com o meio;
- quanto maior for o valor da função de aptidão, mais o indivíduo estará adaptado ao meio.

Algoritmo Genético (AG)

- Um AG inicia sua busca com uma população

  $\{s^0_1\ s^0_2\ s^0_3\ \ldots\ s^0_n\}$

  normalmente escolhida aleatoriamente, a qual é chamada de população no tempo 0 (zero).
- O procedimento principal é um loop que cria uma população $\{ s^{t+1}_1\ s^{t+1}_2 \ldots s^{t+1}_n \}$ no tempo *t+1* a partir de uma população no tempo *t*;
- para criar a nova população, os indivíduos da população anterior (tempo *t*) passam por uma fase de reprodução, a qual consiste em selecionar indivíduos para operações de recombinação e/ou mutação.
- Existem várias formas de selecionar indivíduos para o processo de reprodução, uma delas é conhecida por *Binary Tournament Selection*.

Binary Tournament Selection

- Neste processo de seleção dois indivíduos são escolhidos aleatoriamente e aquele que tiver o maior valor para a função de aptidão é selecionado para ser o primeiro pai;
- o segundo pai é escolhido de forma análoga.
- Outra forma de selecionar os pais para o processo de seleção, é escolhê-los aleatoriamente.

Binary Tournament Selection

- Na operação de recombinação, os genes de dois cromossomos pais são combinados de forma a gerar cromossomos filhos (normalmente dois);
- para cada cromossomo filho há um conjunto de genes de cada um dos cromossomos pais.
- A operação de mutação consiste em alterar aleatoriamente uma parte dos genes de cada cromossomo.

Binary Tournament Selection

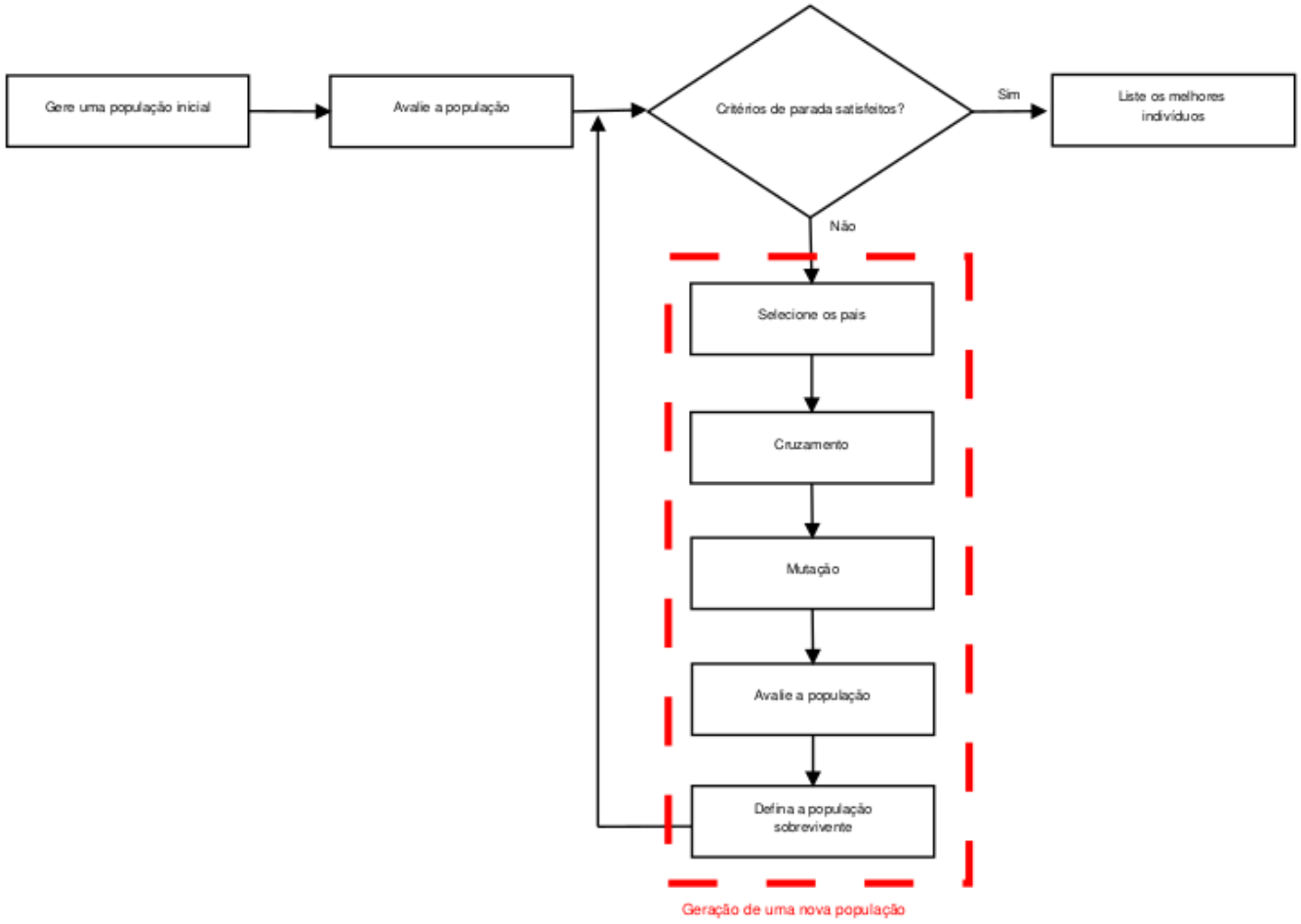
- Ambas as operações são realizadas com uma certa probabilidade.
- A operação de recombinação é realizada com uma probabilidade mais elevada (por exemplo 80%);
- já a operação de mutação com uma probabilidade mais baixa, algo entre 1 e 2% geralmente!

Algoritmo Genético (AG)

- Gerada uma nova população do tempo *t+1*, define-se a população sobrevivente, ou seja, as *n* soluções que integrarão a nova população.
- A população sobrevivente é definida pela aptidão dos indivíduos.
- Os critérios comumente usados para escolher os cromossomos sobreviventes são os seguintes:
  - aleatório,
  - roleta (onde a chance de sobrevivência de cada cromossomo é proporcional ao seu nível de aptidão) e
  - misto (uma combinação dos dois critérios anteriores).
- Em qualquer um destes critérios admite-se a sobrevivência de indivíduos menos aptos.

Algoritmo Genético (AG)

- Estrutura de um Algoritmo Genético básico:



Algoritmo Genético (AG)

- Os critérios de parada mais comuns são:
  - quando um certo número de gerações é atingido,
  - quando não há melhora após um certo número de iterações ou
  - quando o desvio padrão da população é inferior a um certo valor (situação que evidencia uma homogeneidade da população).

Algorimo Genético (AG)

- Os parâmetros principais de controle do método são:
  - o tamanho *n* da população,
  - a probabilidade da operação *crossover*,
  - a probabilidade da operação de mutação,
  - o número de gerações e
  - o número de iterações sem melhora.

Pseudocódigo do Algoritmo Genético

```
procedimento AG
  t←0;
  Gere a população inicial P(t);
  Avalie P(t);
  enquanto (os critérios de parada não estiverem
            satisfeitos) faça
    t←t+1;
    Gere P(t) a partir de P(t-1);
    Avalie P(t);
    Defina a populção sobrevivente;
  fim enquanto
fim procedimento
```

Representação Genética de Soluções

- Normalmente uma solução de um problema está associada a um cromossomo *p* representado na forma de um vetor (ou lista) com m posições: $p = (x_1, x_2, \ldots x_n)$, onde cada componente x_i representa um gene (ou uma variável da solução).
- Dentre os tipos de representação de um cromossomo, os mais conhecidos são:
  - representação binária e
  - representação por inteiros.

Representação Binária

- Uma solução do problema é representada por um vetor (ou lista) de 0's (zeros) e 1's (uns).
- Para ilustrar esta representação, considere o seguinte exemplo extraído de Michalewicz (1982) *apud* Souza (2021), no qual deseja-se maximizar a função:

$$f(x) = |11 * num(x) - 150|$$

- em que *num(x)* fornece o número de 1's do vetor de cromossomos.
- Exemplo: se *x=(101001)* então *num(x)=3*.

Representação Binária

- Considere agora a representação binária de dois cromossomos compostos de 30 genes:

$p_1 = (10101010101010101010101010101010)$

$p_2 = (11000000110000111000000000000000)$

- Então:

$f(p_1) = |11 * num(p_1) - 150|$

$f(p_1) = |11 * 15 - 150| = \mathbf{15}$

$f(p_2) = |11 * num(p_2) - 150|$

$f(p_2) = |11 * 7 - 150| = \mathbf{73}$

Representação Binária

- Observe que o cromossomo que fornece um ponto de *máximo local* é dado por:

$$p_0=(00000000000000000000000000000000)$$

- com:

$$f(p_0)=150$$

- Já o *máximo global* de *f* é fornecido pelo cromossomo:

$$p^*=(11111111111111111111111111111111)$$

- com:

$$f(p^*)=180$$

Representação por Inteiros

- Uma solução do problema é representada por um vetor (ou lista) de números inteiros.
- Considerando o PCV, o cromossomo *p=(1 3 6 5 4 2 7 9 8)* pode representar exatamente a ordem de visita do PCV, ou seja, a rota (*tour*) *t* do PCV é exatamente *p*:

  t=(1 3 6 5 4 2 7 9 8)

Representação por Inteiros

- Em um problema de programação de horários, um cromossomo pode ser uma matriz de números inteiros em que cada componente x_{ij} representa o número da turma para a qual o Professor *i* leciona a aula no horário *j*.

Operador *Crossover* Clássico

- A ideia é a de efetuar o cruzamento entre dois ou mais cromossomos pais e fomar cromossomos filhos (*offsprings*) a partir da união de segmentos de genes de cada pai.
- Inicialmente são feitos cortes aleatórios nos pais. Exemplo: considere dois pais e um ponto de corte realizado na parte central dos cromossomos.

$p_1 = (1\ 0\ 1 \mid 0\ 1\ 0) = (p^1_1 \mid p^2_1)$

$p_2 = (1\ 1\ 1 \mid 0\ 0\ 0) = (p^1_2 \mid p^2_2)$

↑ ponto de corte

  - a partir dos cortes são gerados dois filhos, cada qual formado da união de partes de cada um dos pais:

$O_1 = (p^1_1 \mid p^2_2) = (1\ 0\ 1 \mid 0\ 0\ 0)$

$O_2 = (p^1_2 \mid p^2_1) = (1\ 1\ 1 \mid 0\ 1\ 0)$

Operador de Mutação Clássico

- Consiste em alterar um ou mais genes de um cromossomo.
- Para ilustrar, seja o cromossomo *p* de 7 genes, dado por:

$$p=(x_1\ x_2\ x_3\ \ldots\ x_7) = (1\ 0\ 1\ 0\ 1\ 0\ 1)$$

- considerando como mutação a alteração de um gene de um valor 1 para 0 ou vice-versa, então o cromossomo $p'=(1\ \mathbf{1}\ 1\ 0\ 1\ 0\ 1)$ representa uma mutação de p, no qual o gene x_2 foi alterado do valor 0 para 1.

Operador *Crossover* para o PCV

- A representação por inteiros é considerada mais adequada que a representação binária:
  - primeiramente por melhor se relacionar com uma solução (rota) do PCV que consiste de uma lista de números inteiros (cidades) e
  - segundo porque a representação por inteiros favorece a construção de cromossomos que representam soluções viáveis no PCV.

Operador *Crossover* para o PCV

- O uso de operadores clássicos, neste caso, costumam gerar soluções inviáveis. Sendo necessário incorporar regras adicionais para viabilizar tais soluções.
- Para ilustrar, consideremos 2 cromossomos pais:

  p_1=(1 2 3 | 4 5 6) associado a rota t_1={1 2 3 4 5 6}

  p_2=(3 5 6 | 2 1 4) associado à rota t_2={3 5 6 2 1 4}

  neste caso são gerados dois cromossomos filhos:

  O_1=(p^1_1 | p^2_2)=(1 2 3 | 2 1 4)

  O_2=(p^1_2 | p^2_1)=(3 5 6 | 4 5 6)

- as rotas associadas a O_1 e O_2 são inviáveis. Em O_1 foi gerada uma rota contendo apenas as cidades *1*, *2*, *3* e *4*, repetindo as visitas às cidades *1* e *2*. Em O_2 foram repetidas as cidades *5* e *6* e não foram visitadas as cidades *1* e *2*.

Operador *Crossover* para o PCV

- Como corrigir o problema?
  - foram criados operadores de *crossover* específicos para tratar os casos do PCV;
  - a seguir...

Operador PMX

- **Operador PMX (*Partial Mapped Crossover*):**
  - considere dois cromossomos pais, p_1 e p_2. Selecione dois cortes em ambos aleatoriamente. No caso de serem gerados dois cromossomos filhos O_1 e O_2, os genes localizaos entre os dois cortes de p_1 e p_2 são herdados integralmente por O_2 e O_1, preservando a **ordem** e a **posição** de cada cidade.

Operador PMX

- Exemplo:

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

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

então:

O_1=(X X X | 1 8 5 9 | X X)

O_2=(X X X | 4 5 6 7 | X X)

a seguir tenta-se preencher cada componente *X* de O_1 pelo componente de p_1 e os de O_2 com p_2. Caso não formem uma rota ilegal para o PCV

↓

O_1=(X 2 3 | 1 8 5 9 | X X)

O_2=(X 2 X | 4 5 6 7 | 3 X)

considere agora o primeiro *X* de O_1 (que deveria ser $X=1$), mas forneceria uma rota ilegal, já que esta cidade já se encontra presente na rota).

Operador PMX

- Substituir o *X* por *1* não é possível. Então, toma-se a cidade correspondente à posição em que está a cidade *1* em p_1 no segundo pai, p_2. A cidade 1 está na posição 1 do cromossomo.

```
    X
    ⇓
p1=(1 ...
p2=(4 ...
```

```
p1=(1 2 3 | 4 5 6 7 | 8 9)
p2=(4 2 6 | 1 8 5 9 | 3 7)
```

obtem-se com isto:

```
O1=(4 2 3 | 1 8 5 9 | X X)
```

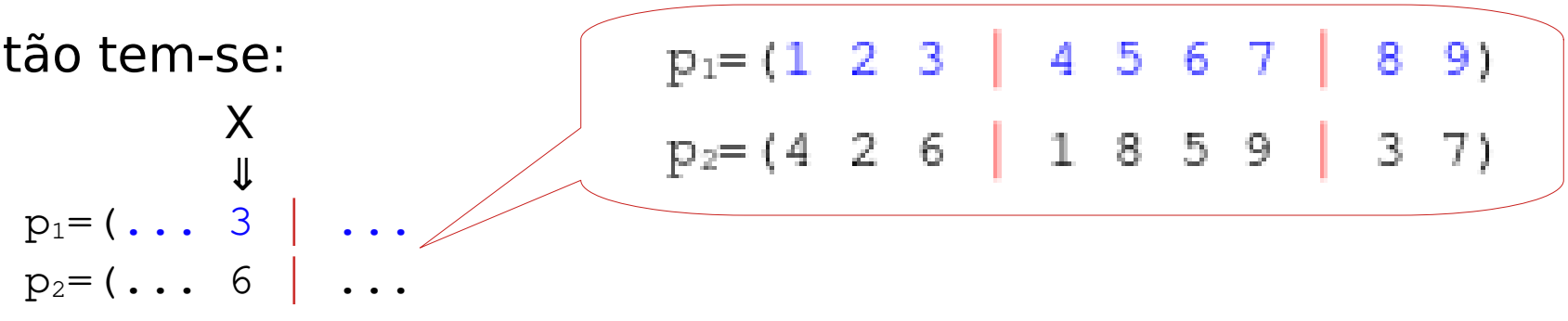
repete-se o procedimento para o próximo *X* de O_1 (onde X = 8)

```
                X
                ⇓
p1=(... | ... | 8 ...)
p2=(... | ... | 3 ...)
```

mas *X=3* já está presente em O_1, então toma-se o número de p_2 associado à posição do número *3* do cromossomo de p_1.

Operador PMX

então tem-se:




```
          X
          ⇓
p1=(... 3 | ...
p2=(... 6 | ...
```

e obtem-se:

```
O1=(4 2 3 | 1 8 5 9 | 6 X)
```

repete-se o procedimento para o próximo *X* de O_1 (onde X = 9)

```
                    X
                    ⇓
p1=(... | ... | ... 9)
p2=(... | ... | ... 7)
```

como o 7 ainda não está presente em O_1, substitui-se o X por 7 e obtem-se:

```
O1=(4 2 3 | 1 8 5 9 | 6 7)
```

Operador PMX

Por fim, de forma análoga aos passos anteriores, preenche-se os *X's* de O_2, obtendo-se ao final:

$O_2 = (1\ \ 2\ \ 8 \mid 4\ \ 5\ \ 6\ \ 7 \mid 3\ \ 9)$

- O operador PMX explora a conveniência ou não de se reproduzir cromossomos localmente idênticos a um cromossomo pai. Esta propriedade é útil, por exemplo, quando se têm rotas já localmente otimizadas.

Operador OX

- Proposto por Davis (1985) *apud* Souza (2021), o ***Ordenated Crossover*** (OX) constrói um *offspring* (cromossomo filho) escolhendo uma subsequência de uma rota associada a um cromossomo pai p_1 e preservando a **ordem** relativa das cidades do outro cromossomo pai p_2.

Operador OX

- Exemplo:

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

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

  os filhos O_1 e O_2 herdam a faixa entre os dois cortes de p_1 e p_2, respectivamente:

  O_1=(X X X | 4 5 6 7 | X X)

  O_2=(X X X | 1 8 7 6 | X X)

  agora, partindo-se do segundo corte de um pai p_2, copia-se em uma lista as n cidades de p_2. Remove-se desta lista as cidades contidas entre os dois cortes do outro pai (p_1). A subsequência resultante é enxertada no fillho O_1 associado a p_1 a partir do segundo corte, seguindo a ordem da subsequência.

- Assim temos a sequência:

  9 3 4 5 2 1 8 7 6

  remove-se 4, 5, 6 e 7 desta lista; uma vez que estas cidades já foram visitadas. Resultando na subsequência: 9, 3, 2, 1 e 8.

Operador OX

- Exemplo:

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

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

  os filhos O_1 e O_2 herdam a faixa entre os dois cortes de p_1 e p_2, respectivamente:

  O_1=(X X X | 4 5 6 7 | X X)

  O_2=(X X X | 1 8 7 6 | X X)

  agora, partindo-se do segundo corte de um pai p_2, copia-se em uma lista as n cidades de p_2. Remove-se desta lista as cidades contidas entre os dois cortes do outro pai (p_1). A subsequência resultante é enxertada no fillho O_1 associado a p_1 a partir do segundo corte, seguindo a ordem da subsequência.

- Assim temos a sequência:

  9 3 4 5 2 1 8 7 6

  remove-se 4, 5, 6 e 7 desta lista; uma vez que estas cidades já foram visitadas. Resultando na subsequência: 9, 3, 2, 1 e 8.

Operador OX

- A subsequência resultante, 9 3 2 1 8 deve ser inserida em O_1 a partir do segundo corte em ordem:

  O_1=(X X X | 4 5 6 7 | X X)

  torna-se:

  O_1=(2 1 8 | 4 5 6 7 | 9 3)

- Analogamente partindo-se de p_1, tem-se a sequência:

  8 9 1 2 3 4 5 6 7

  remove-se 1, 8, 7 e 6 desta sequência e tem-se a subsequência: 9, 2, 3, 4 e 5. Que deve ser enxertada em O_2 a partir do segundo corte,

  O_2=(X X X | 1 8 7 6 | X X)

  obtendo-se:

  O_2=(3 4 5 | 4 5 6 7 | 9 2)

Operador OX

- O operador OX prioriza a **ordem** das cidades e não suas posições na rota. A posição das cidades na rota não é importante no PCV e sim, a ordem de visitas, ou seja, no PCV é importante saber quem são as cidades vizinhas de uma dada cidade *i*.
- A irrelevância das posições na rota pode ser vista no seguinte exemplo:

  T_1= 5 - 3 - 1 - 2 - 4

  T_2= 3 - 1 - 2 - 4 - 5

- em T_1 e T_2 as cidades aparecem em posições diferentes, mas tanto T_1 quanto T_2 representam a mesma solução.

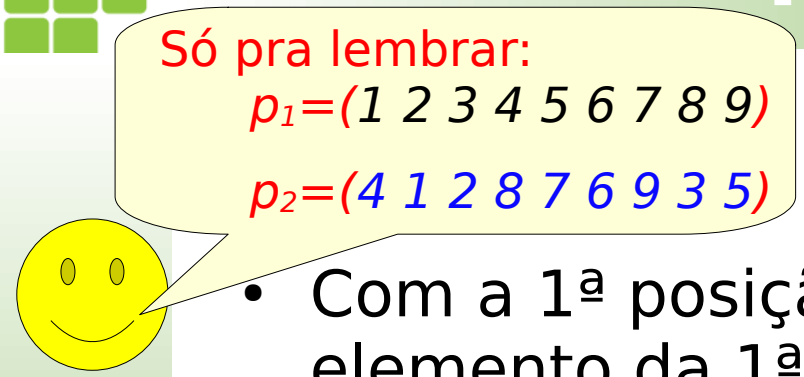
Operador CX

- Proposto por Oliver *et. al.* (1987) *apud* Souza (2021), o operador *Cycle Crossover* (CX) preserva a posição absoluta das cidades nos cromossomos pais.
- Exemplo:

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

  $p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$
- o primeiro filho O_1 é obtido tomando-se inicialmente a primeira cidade de p_1:

  $O_1 = (1\ X\ X\ X\ X\ X\ X\ X\ X)$



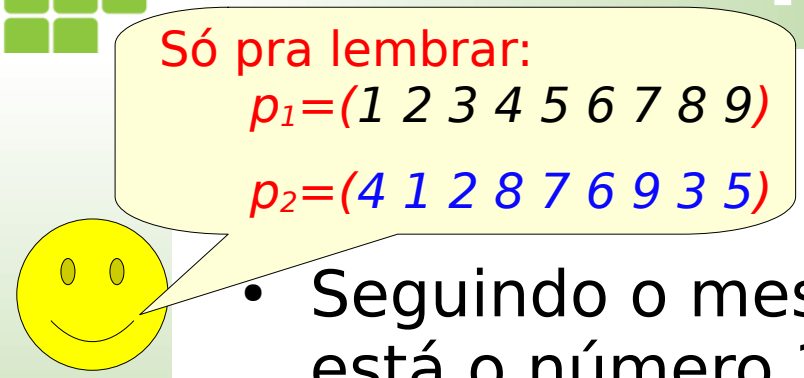
Operador CX

- Com a 1ª posição de O_1 já preenchida, então o elemento da 1ª posição de p_2 igual a 4 não pode ser alocado na 1ª posição de O_1. Portanto, o elemento da 1ª posição de p_2, o número 4, é herdado pelo filho O_1 na posição que ele ocupa em p_1. Ou seja, o número 4 está na 4ª posição de p_1, logo estará também na 4ª posição de O_1

  O_1=(1 X X 4 X X X X X)

- a seguir, sendo o elemento da 4ª posição do outro pai p_2 igual 8. E o número 8 em p_1 se encontra na 8ª posição, então aloca-se o 8 na 8ª posição de O_1

  O_1=(1 X X 4 X X X 8 X)



Operador CX

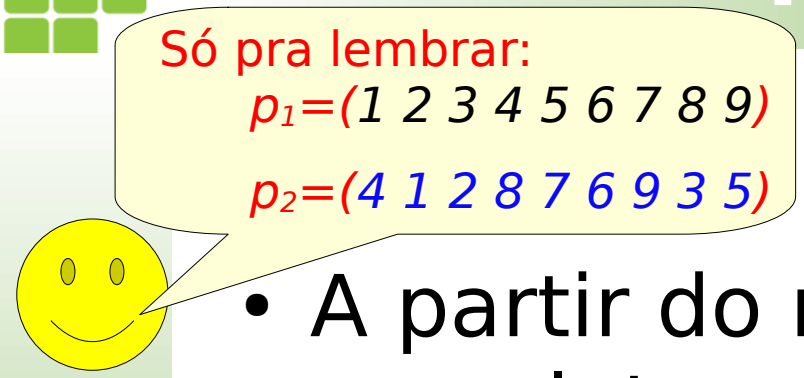
- Seguindo o mesmo procedimento, na 8ª posição de p_2 está o número 3, que se encontra na 3ª posição de p_1 e, assim sendo, O_1 recebe o número 3 na 3ª posição.

  O_1=(1 X 3 4 X X X 8 X)

- a seguir, verificamos que o número que está na 3ª posição de p_2 é o 2. Mas, ele se encontra na 2ª posição de p_1. Então O_1 recebe o 2 na 2ª posição.

  O_1=(1 2 3 4 X X X 8 X)

- agora observa-se que o número que está na 2ª posição de p_2 é o 1, que já aparece em O_1. E, neste caso, diz-se que foi completado o ciclo.



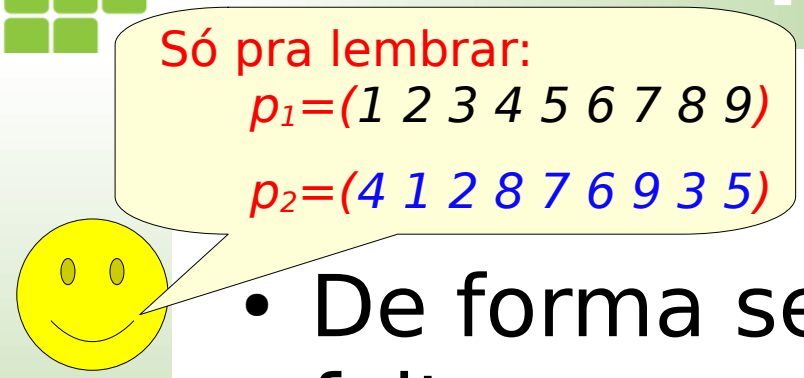
Operador CX

- A partir do momento que o ciclo está completo, obtem-se as cidades restantes para completar O_1 do outro pai (p_2)

  O_1=(1 2 3 4 X X X 8 X)

- observe que faltam ser preenchidas as posições: 5ª, 6ª, 7ª e 9ª de O_1.
- Então:

  O_1=(1 2 3 4 7 6 9 8 5)



Operador CX

- De forma semelhante ao processo já feito para a constituição de O_1, toma-se inicialmente a 1ª cidade do segundo cromossomo pai (p_2) e começa-se a construir o segundo cromossomo filho O_2, que ao final será:

  O_2=(4 1 2 8 5 6 7 3 9)

- observe que o operador *Cycle Crossover* sempre preserva a posição de elementos dos cromossomos pais.

Operador ERX

- O operador *Edge Recombination* (ERX) foi desenvolvido especialmente para o PCV, ele prioriza o fator *adjacência*.
- Outra característica deste operador é a de que um cromossomo filho deve ser construído, sempre que possível, a partir das arestas presentes em ambos os pais.
- No ERX, o significativo número de arestas transferidas de cromossomos pais para filhos é uma consequência do seguinte procedimento:
  - **criar uma lista de arestas de ambos os pais. A lista de arestas produz para cada cidade, todas as outras cidades a ela conectadas em pelo menos um dos cromossomos pais. Isto significa que para cada cidade existem, no mínimo duas e no máximo quatro cidades conectadas.**

Operador ERX

- **Exemplo:** considere duas rotas do PCV ou equivalentemente na representação por caminhos, dois cromossomos.

  p_1=*(1 2 3 4 5 6)*

  p_2=*(3 4 1 6 2 5)*

- a lista de arestas será:

  cidade 1: (1 2) (6 1) (4 1)

  cidade 2: (1 2) (2 3) (6 2) (2 5)

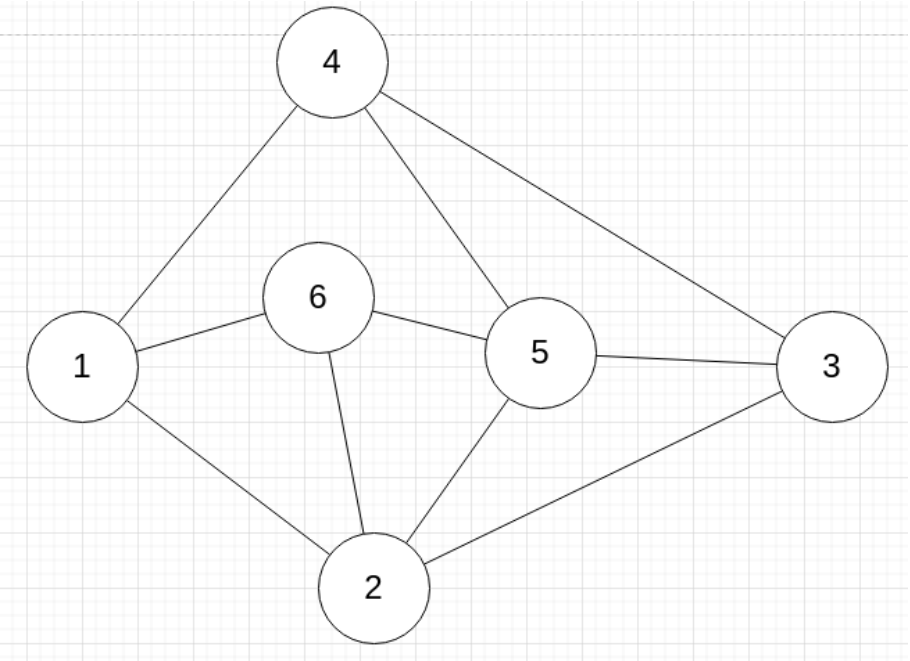
  cidade 3: (2 3) (3 4) (5 3)

  cidade 4: (3 4) (4 5) (4 1)

  cidade 5: (4 5) (5 6) (2 5) (5 3)

  cidade 6: (5 6) (6 1) (6 2)

- supondo-se que *(i j)* = *(j i)*.



Operador ERX

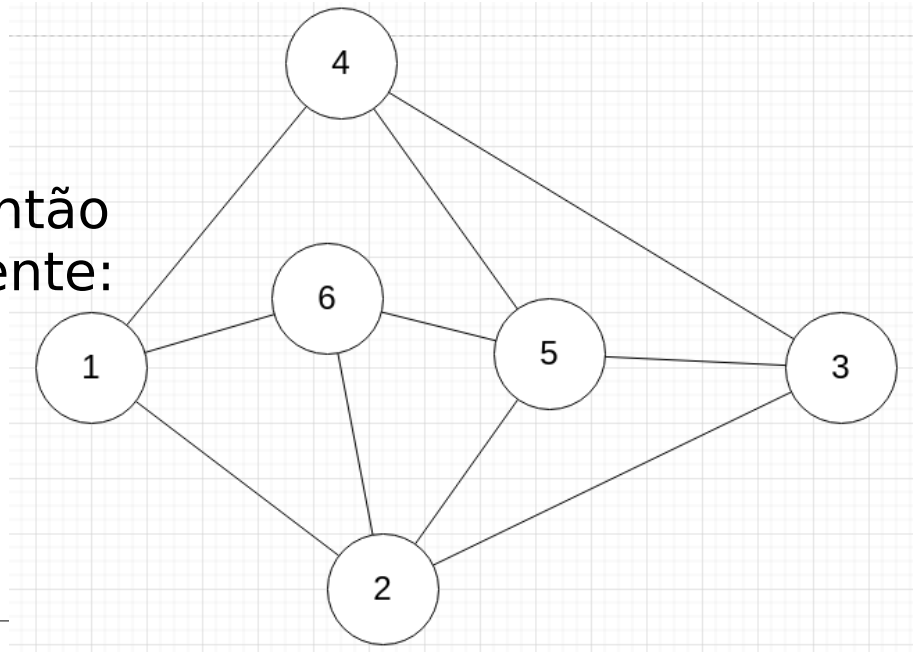
- **Construção do Cromossomo Filho:**
  - a partir de dois cromossomos pais, pode-se gerar um ou dois filhos;
  - selecione a cidade inicial de um dos dois pais. A cidade inicial deve ser a que tiver menos arestas. Então, compara-se o número de arestas das cidades (1 e 3):

    p_1=(1 2 3 4 5 6)

    p_2=(3 4 1 6 2 5)
  - como ambas empataram, então escolhe-se uma aleatoriamente:

    para este exemplo escolhemos a cidade 1, então:

    O_1=(1 X X X X X)



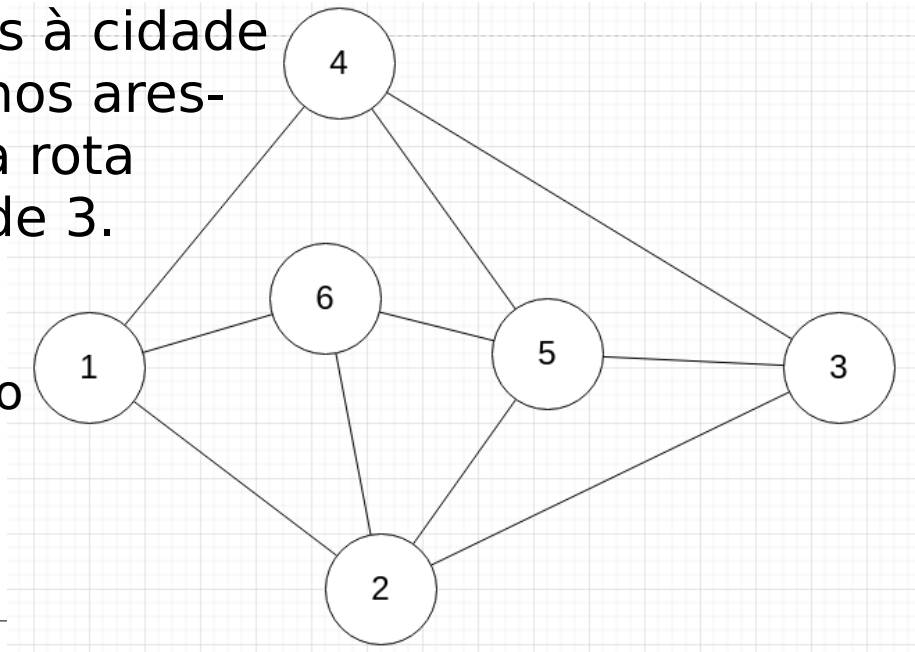
Operador ERX

- **Construção do Cromossomo Filho:**
  - a cidade 1 está diretamente ligada às cidades: 2, 4 e 6. Então, a próxima cidade do cromossomo O_1 será, dentre estas, a que tiver menos arestas. Como 4 e 6 empataram, escolhemos a cidade 4:

    $O_1=(1\ 4\ X\ X\ X\ X)$
  - Dentre as cidades conectadas à cidade 4, escolhe-se a que tiver menos arestas e que ainda não esteja na rota parcial. Selecionou-se a cidade 3.

    $O_1=(1\ 4\ 3\ X\ X\ X)$
  - seguindo o mesmo procedimento escolhemos a cidade 2:

    $O_1=(1\ 4\ 3\ 2\ X\ X)$

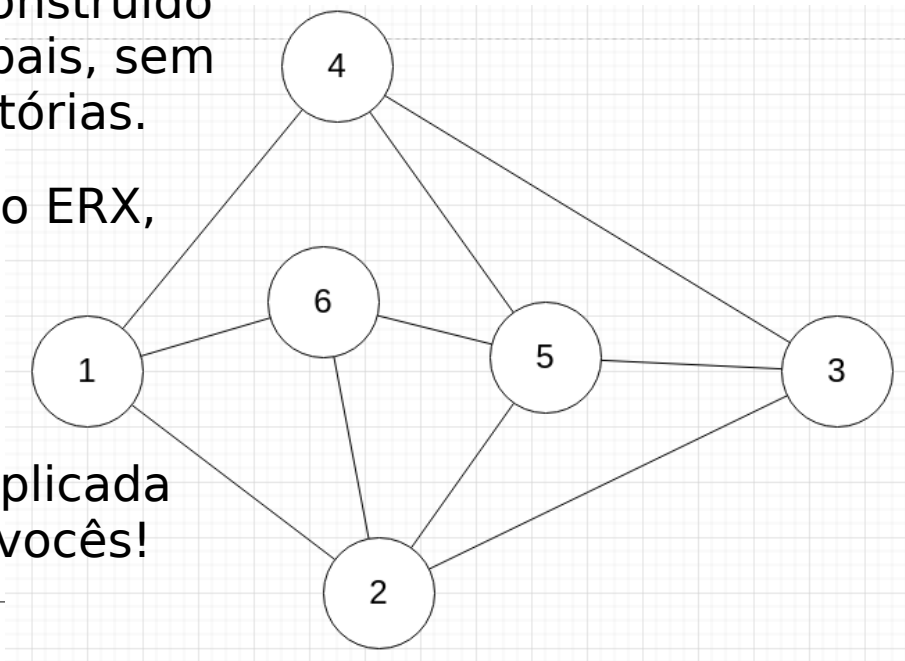


Operador ERX

- **Construção do Cromossomo Filho:**
  - a próxima cidade escolhida é a 6:

    O_1=(*1 4 3 2 6 X*)
  - e finalmente a cidade 5 é incluída na rota:

    O_1=(*1 4 3 2 6 5*).
- Observe que o cromossomo O_1 foi construído inteiramente de arestas de um dos pais, sem a necessidade de gerar arestas aleatórias.
- Por fim, existe ainda uma variante do ERX, chamada EERX (*Enhanced Edge Recombination*) que prioriza as arestas comuns aos dois pais, para ser herdada pelo cromossomo filho. Mas, esta variante não será explicada aqui. Fica como dever de casa para vocês!



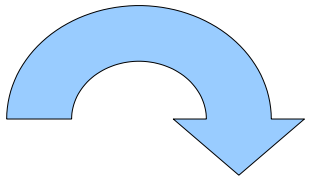
Bibliografia

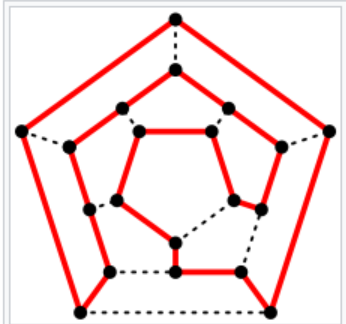
- "heurística", in Dicionário Priberam da Língua Portuguesa, 2008-2021. Disponível em: <https://dicionario.priberam.org/heur%C3%Adstica>. Acesso em: 09 Nov. 2021.
- "meta", in Dicionário Priberam da Língua Portuguesa, 2008-2021. Disponível em: <https://dicionario.priberam.org/meta>. Acesso em: 09 Nov. 2021.
- SALIBA JÚNIOR, E. **Sistema Multiagente Bioinspirado para Otimização Combinatorial**. 2010. 117 f. Dissertação (Mestrado em Modelagem Matemática e Computacional) Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 2010.
- SOUZA, Marcone Jamilson Freitas. **Inteligência Computacional para Otimização**. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm>. Acesso em: 20 Out. 2021.
- SOUZA, Marcone J. F. e PENNA, Puca H. V. Introdução. **Notas de aula de Técnicas Meta-heurísticas para Otimização Combinatória**. Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, 2021. Disponível em <www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/Introducao.pptx>. Acesso em: 09 Nov. 2021.

Caminho hamiltoniano

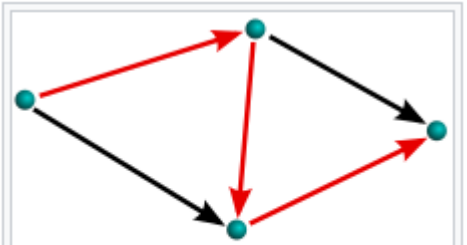
- Um caminho hamiltoniano é um caminho que permite passar por todos os vértices de um grafo G, não repetindo nenhum. Ou seja, passar por todos uma única vez. Caso esse caminho seja possível descrever um ciclo, este é denominado ciclo hamiltoniano (ou circuito hamiltoniano) em G. E um grafo que possua tal circuito é chamado de grafo hamiltoniano.

(CAMINHO hamiltoniano. *In:* WIKIPÉDIA: a enciclopédia livre. [São Francisco, CA: Fundação Wikimedia], 2023. Disponível em: https://pt.wikipedia.org/wiki/Caminho_hamiltoniano. Acesso em: 30 maio 2023.)





Um ciclo Hamiltoniano em um dodecaedro. Como todos os sólidos platônicos, o dodecaedro é Hamiltoniano.



O caminho vermelho é hamiltoniano.