Relatório da implementação do problema do caixeiro viajante (PCV)

Átila Bernardo Mota Sousa

Departamento de Computação
Universidade Federal de Sergipe (UFS) – Aracaju, SE – Brasil

Resumo. *Este relatório descreve a implementação em Python do problema do caixeiro viajante através do algoritmo de Bellmore & Nemhauser, heurística Twice-Around e heurística de Christofides. Também são apresentadas descrições do problema, dos algoritmos e os resultados e conclusões dos testes realizados com diferentes bases de dados.*

1. Descrição do problema

Esse problema descreve um viajante que deseja visitar todos os vértices de um grafo somente uma única vez e depois retornar ao vértice inicial, sendo que esse ciclo deverá ser o menor possível. Levando essa questão para os conceitos de grafos, o problema refere-se a encontrar *ciclos hamiltonianos* de custo mínimo em um grafo.

2. Descrição dos algoritmos

São conhecidos diversos algoritmos e heurísticas para a resolução desse problema. Nesse trabalho em específico foram feitas as implementações do algoritmo de *Bellmore & Nemhauser*, heurística *Twice-Around* e heurística de *Christofides*.

O algoritmo de *Bellmore & Nemhauser* é o mais simples dos três e utiliza o método guloso. Esse algoritmo inicia com um vértice inicial qualquer e a partir dele encontra o seu vizinho mais próximo, depois o insere no ciclo e repete esse processo para o próximo vértice, tudo isso em um laço que se repete até que o ciclo atinja seu limite superior de vértices.

Já as heurísticas *Twice-Around* e de *Christofides* são mais complexas e utilizam outros conceitos de grafos para auxiliar na resolução do problema. Esses conceitos são o de *árvores geradoras mínimas* (*MST*) e o de *ciclos eulerianos*. Na implementação em questão foram utilizados o algoritmo de *Prim* para as árvores geradoras mínimas e o algoritmo de *Hierholzer* para selecionar os ciclos eulerianos.

A heurística do *Twice-Around* consiste inicialmente em determinar as *árvores geradoras mínimas* e posteriormente dobrar suas arestas para torná-la *euleriana*. Depois essa árvore com as arestas duplicadas é enviada para o algoritmo de *Prim* que identificará um ciclo euleriano. Através desse ciclo gerado é realizado um método chamado *TW(.)*.

Na etapa *TW(.)* são selecionados sequencialmente os vértices no *ciclo euleriano* e caso eles não estejam no vetor *H* que armazena o *ciclo hamiltoniano*, serão inseridos nele. Após isso o vértice atual será removido do *ciclo euleriano* e o processo será repetido até que o *ciclo euleriano* esteja vazio.

A heurística de *Christofides* é bastante semelhante ao *Twice-Around*, com uma diferença na etapa de tratamento da *árvore geradora mínima*. Enquanto o *Twice-Around*

somente dobra as arestas da árvore, a heurística de *Christofides* busca *matchings-perfeitos* dentro dessa árvore. Esse *matching* é feito selecionando os vértices de grau ímpar do grafo e depois verificando qual a combinação de menor custo entre eles, ao final do processo todos os vértices terão grau par e o grafo será *euleriano*. Por fim é invocado o procedimento *TW(.)* e definido o *ciclo hamiltoniano* de custo mínimo.

3. Resultados encontrados

Os resultados obtidos foram a validação dos algoritmos através dos exemplos mostrados no livro*[2]* e posteriormente a aplicação dos algoritmos nas bases de dados disponibilizadas, exibindo os *ciclos hamiltonianos de custo mínimo* de cada algoritmo e seus respectivos custos e tempos de execução.

3.1. Validação dos algoritmos

Para validação dos algoritmos foram selecionados 2 grafos de exemplos do livro de *Goldbarg[2]*. O primeiro grafo se refere a um exemplo do algoritmo de *Bellmore & Nemhauser,* enquanto o segundo vem de um exemplo das heurísticas de *Christofides* e *Twice-Around*.

3.1.1. Grafo 1

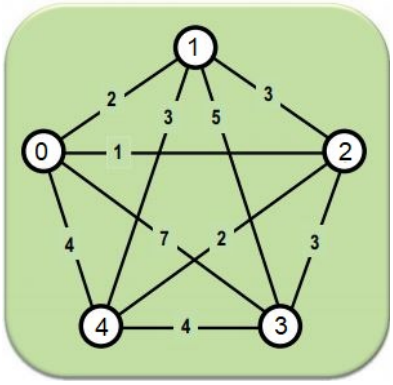

Figura 1. Grafo 1

```
Grafo 1:
0 -> 1 -> 4 -> 2 -> 3 -> NULL
1 -> 0 -> 2 -> 3 -> 4 -> NULL
2 -> 1 -> 3 -> 0 -> 4 -> NULL
3 -> 2 -> 4 -> 0 -> 1 -> NULL
4 -> 3 -> 0 -> 1 -> 2 -> NULL
Bellmore & Nemhauser:
[0, 2, 4, 1, 3, 0]
custo = 18
Twice-Around:
[0, 2, 4, 3, 1, 0]
custo = 14
Christofides:
[0, 2, 3, 4, 1, 0]
custo = 13
```

Figura 2. Saída do grafo 1

3.1.2. Grafo 2

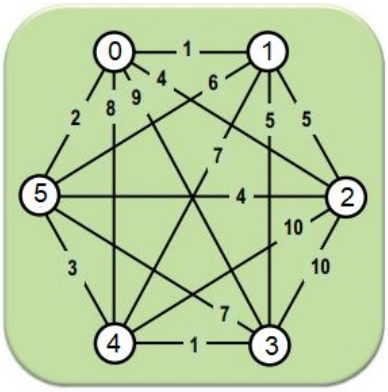

Figura 3. Grafo 2

```
Grafo 2:
0 ->  1 ->  4 ->  2 ->  3 ->  5 ->  NULL
1 ->  0 ->  2 ->  3 ->  4 ->  5 ->  NULL
2 ->  1 ->  3 ->  0 ->  4 ->  5 ->  NULL
3 ->  2 ->  4 ->  0 ->  1 ->  5 ->  NULL
4 ->  3 ->  0 ->  1 ->  2 ->  5 ->  NULL
5 ->  0 ->  1 ->  2 ->  3 ->  4 ->  NULL
Bellmore & Nemhauser
[0, 1, 2, 5, 4, 3, 0]
custo = 23
Twice-around:
[0, 2, 5, 4, 3, 1, 0]
custo = 18
Christofides:
[0, 1, 2, 3, 4, 5, 0]
custo = 22
```

Figura 4. Saída do grafo 2

3.2. Aplicação nas bases de dados

As bases de dados utilizadas*[3]* foram as tabelas de distância *pd01_d.txt, gr17_d.txt, fri26_d.txt, dantzig42_d.txt* e *att48_d.txt*. Essas bases possuem respectivamente 15, 17, 26, 42 e 48 vértice e consistem em matrizes de adjacências de grafos completos.

3.2.1. Base de dados pd01_d.txt

```
Grafo da base de dados pd01_d.txt:
0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
1 ->  0 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
2 ->  0 ->  1 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
3 ->  0 ->  1 ->  2 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
4 ->  0 ->  1 ->  2 ->  3 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
5 ->  0 ->  1 ->  2 ->  3 ->  4 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
6 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
7 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
8 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  9 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
9 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  10 ->  11 ->  12 ->  13 ->  14 ->  NULL
10 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  11 ->  12 ->  13 ->  14 ->  NULL
11 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  12 ->  13 ->  14 ->  NULL
12 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  13 ->  14 ->  NULL
13 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  14 ->  NULL
14 ->  0 ->  1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 ->  10 ->  11 ->  12 ->  13 ->  NULL
Bellmore & Nemhauser
[0, 12, 1, 14, 8, 4, 6, 2, 11, 13, 9, 7, 5, 3, 10, 0]
custo = 291
tempo de execucao = 0.00699996948242 segundos
Twice-around:
[0, 12, 1, 7, 9, 11, 13, 2, 5, 3, 10, 14, 8, 6, 4, 0]
custo = 412
tempo de execucao = 0.010999917984 segundos
Christofides:
[0, 10, 3, 5, 7, 13, 11, 2, 9, 1, 8, 6, 4, 14, 12, 0]
custo = 349
tempo de execucao = 0.0120000839233 segundos
```

Figura 5. Saída da base de dados pd01_d.txt

3.2.2. Base de dados gr17_d.txt

```
Grafo da base de dados gr17_d.txt:
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
1 -> 0 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
2 -> 0 -> 1 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
3 -> 0 -> 1 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
4 -> 0 -> 1 -> 2 -> 3 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
5 -> 0 -> 1 -> 2 -> 3 -> 4 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
6 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
7 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
8 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
9 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
10 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
11 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 12 -> 13 -> 14 -> 15 -> 16 -> NULL
12 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 13 -> 14 -> 15 -> 16 -> NULL
13 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 14 -> 15 -> 16 -> NULL
14 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 15 -> 16 -> NULL
15 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 16 -> NULL
16 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> NULL
Bellmore & Nemhauser
[0, 12, 3, 6, 7, 5, 16, 13, 14, 2, 10, 4, 9, 1, 8, 11, 15, 0]
custo = 2187
tempo de execucao = 0.00600004196167 segundos
Twice-around:
[0, 12, 6, 16, 13, 14, 2, 10, 9, 4, 1, 7, 5, 3, 8, 11, 15, 0]
custo = 2396
tempo de execucao = 0.0250000953674 segundos
Christofides:
[0, 12, 6, 5, 7, 16, 13, 14, 2, 10, 9, 4, 1, 15, 11, 8, 3, 0]
custo = 2406
tempo de execucao = 0.0210001468658 segundos
```

Figura 6. Saída da base de dados gr17_d.txt

3.2.3. Base de dados fri26_d.txt

```
24 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 ->
23 -> 25 -> NULL
25 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 ->
23 -> 24 -> NULL
Bellmore & Nemhauser
[0, 14, 13, 9, 10, 12, 11, 8, 6, 7, 15, 18, 19, 17, 16, 20, 21, 25, 22, 23, 24, 2, 1, 3, 4, 5, 0]
custo = 1112
tempo de execucao = 0.00999999046326 segundos
Twice-around:
[0, 14, 13, 9, 10, 12, 11, 8, 15, 18, 19, 17, 16, 20, 21, 22, 24, 23, 25, 6, 7, 4, 5, 3, 2, 1, 0]
custo = 1143
tempo de execucao = 0.144999980927 segundos
Christofides:
[0, 1, 2, 13, 9, 8, 3, 4, 5, 6, 7, 15, 18, 19, 17, 16, 20, 21, 25, 22, 23, 24, 11, 12, 10, 14, 0]
custo = 1010
tempo de execucao = 0.0490000247955 segundos
```

Figura 7. Saída da base de dados fri26_d.txt

3.2.4. Base de dados dantzig42_d.txt

```
41 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 ->
23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 30 -> 31 -> 32 -> 33 -> 34 -> 35 -> 36 -> 37 -> 38 -> 39 -> 40 -> NULL
Bellmore & Nemhauser
[1, 0, 40, 41, 39, 38, 37, 36, 34, 33, 30, 29, 31, 32, 28, 27, 26, 25, 24, 23, 9, 8, 7, 6, 5, 4, 3, 2, 35, 20, 21, 22, 16, 15, 17, 18, 19, 12, 13, 14, 11, 10,
 1]
custo = 955
tempo de execucao = 0.0169999599457 segundos
Twice-around:
[1, 0, 41, 40, 39, 38, 37, 36, 35, 34, 33, 30, 31, 32, 29, 27, 28, 4, 3, 2, 7, 24, 23, 26, 25, 10, 22, 21, 20, 16, 15, 17, 18, 19, 14, 13, 12, 11, 9, 8, 6, 5,
 1]
custo = 870
tempo de execucao = 0.15499997139 segundos
Christofides:
[1, 39, 4, 3, 2, 7, 24, 9, 23, 25, 26, 10, 11, 22, 20, 21, 16, 15, 12, 13, 14, 17, 18, 19, 28, 27, 29, 30, 5, 6, 8, 32, 31, 33, 34, 36, 35, 37, 38, 40, 0, 41,
 1]
custo = 861
tempo de execucao = 0.155999898911 segundos
```

Figura 8. Saída da base de dados dantzig42_d.txt

3.2.5. Base de dados att48_d.txt

```
47 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 ->
23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 30 -> 31 -> 32 -> 33 -> 34 -> 35 -> 36 -> 37 -> 38 -> 39 -> 40 -> 41 -> 42 -> 43 -> 44 -> 45 -
> 46 -> NULL
Bellmore & Nemhauser
[10, 22, 13, 24, 12, 20, 46, 19, 32, 45, 14, 11, 39, 8, 0, 7, 37, 30, 43, 17, 6, 27, 35, 29, 5, 36, 18, 26, 42, 16, 2, 21, 15, 40, 33, 28, 4, 47, 38, 31, 23,
9, 41, 25, 3, 34, 44, 1, 10]
custo = 40942
tempo de execucao = 0.0139999389648 segundos
Twice-around:
[10, 46, 20, 11, 14, 39, 8, 37, 30, 43, 17, 35, 6, 27, 5, 29, 36, 18, 26, 42, 16, 0, 21, 15, 2, 7, 32, 45, 19, 22, 13, 33, 40, 24, 38, 47, 41, 9, 25, 3, 34, 4
4, 23, 4, 28, 1, 31, 12, 10]
custo = 46201
tempo de execucao = 0.181999921799 segundos
Christofides:
[10, 12, 24, 1, 28, 4, 47, 38, 41, 9, 25, 3, 44, 34, 23, 31, 13, 20, 46, 11, 14, 8, 37, 30, 43, 17, 35, 6, 27, 5, 29, 36, 18, 26, 42, 16, 19, 32, 45, 39, 0, 7
, 21, 2, 15, 40, 33, 22, 10]
custo = 41853
tempo de execucao = 0.197000026703 segundos
```

Figura 9. Saída da base de dados att48_d.txt

4. Análise e discussão dos resultados

Nos resultados obtidos da validação dos algoritmos, foi observado no grafo 1 que o ciclo hamiltoniano de custo mínimo resultante do algoritmo de *Bellmore & Nemhauser* foi exatamente igual ao mostrado no exemplo do livro*[2]* e com o mesmo custo (18). Ainda nesse grafo, os 2 outros algoritmos acharam ciclos de custo ainda menor e foram mais eficientes nesse quesito.

No grafo 2, utilizado no livro como exemplo para as heurísticas de *Christofides* e *Twice-Around*, foram encontrados ciclos de custo ainda menor do que os apresentados nos exemplos do livro*[2]*. Nesse grafo, o algoritmo *Twice-Around* foi o mais eficiente na métrica custo mínimo (16).

Já nos resultados das bases de dados, em termos de custos de caminho resultante dos três algoritmos, se pode ver que não é possível saber ao certo qual algoritmo resultará no menor custo. Em algumas bases de dados e exemplos, o algoritmo de *Bellmore & Nemhauser* apresentou os ciclos de menores custos, enquanto em outros o vencedor foi a heurística de *Christofides* ou o *Twice-Around*.

Essa incerteza no custo do ciclo dos três algoritmos se deve ao fato de que o desempenho de cada um pode variar de acordo com o grafo. Além disso os algoritmos contém componentes aleatórios, que podem gerar ciclos de diferentes custos em diferentes execuções.

Um resultado curioso se encontra na saída da base de dados *pd01_d.txt.* Como está exibido na figura 3, o custo mínimo do ciclo hamiltoniano do algoritmo de *Bellmore & Nemhauser* foi 291, que é o menor ciclo possível no grafo, descrito no link da base de dados*[3]*.

Outro ponto interessante de se observar é que, na bibliografia*[2]* é explicado que a heurística de *Christofides* garante que a solução produzida por seu algoritmo é no máximo 1,5 vezes pior que a solução ótima do problema. É possível verficiar esse fato nas figuras, analisando os custos dessa heurística e os comparando com os custos mínimos apresentados na base de dados*[3]*.

Em termos de tempo de execução, obteve-se um resultado mais constante. Em geral, é possível observar nas figuras apresentadas que o algoritmo de *Bellmore & Nemhauser* foi o mais rápido, enquanto o *Twice-Around* e a heurística de *Christofides* tiveram tempos próximos.

Analisando a complexidade do algoritmo, pode-se deduzir que os resultados do tempo de execução estão coerentes. O algoritmo de *Bellmore & Nemhauser* por ser o mais simples e direto apresentou os resultados mais rápido, mesmo tendo a mesma complexidade do *Twice-Around* ($O(n^2)$).

Entre o *Twice-Around* ($O(n^2)$) e o algoritmo de *Christofides* ($O(n^3)$), na maioria casos o tempo de execução obedeceu a lógica da complexidade, isto é, o de *Christofides* foi mais lento. Porém em alguns grafos é possível analisar nas figuras que o *Twice-Around* foi ainda mais demorado que *Christofides*. Para analisar essa questão, foram colocados mais medidores de tempo de execução em cada parte específica dos algoritmos.

O desfecho dessa verificação foi que, mesmo o algoritmo de *Christofides* tendo uma complexidade maior por ser dominado pelo *matching-perfeito*, o algoritmo de *Hierholzer* demorava mais tempo de execução no *Twice-Around*, justamente pelas árvores terem suas arestas dobradas. Portanto no exemplo prático, ou seja, no caso médio, em algumas situações o algoritmo de *Hierholzer* acabou dominando o tempo de execução dos algoritmos.

5. Conclusão

Diante da análise dos resultados obtidos, é possível concluir que a validação dos algoritmos, diante dos exemplos apresentados no livro, está correta. Já na comparação entre os algoritmos, em termos de custo de caminho, não há um algoritmo dentre os três que possa ser dito superior com total exatidão. Na característica do tempo de execução, conclui-se que o algoritmo de *Bellmore & Nemhauser* é o mais rápido, também por ser o mais simples. Sobre os outros dois algoritmos, a conclusão é que seus tempos de execução são bastante próximos.

Referências

[1] Vídeo-aulas disponíveis no youtube do professor Renê Gusmão do Departamento de Computação da UFS. Disponível em:

<https://www.youtube.com/watch?v=54oAIViZChA&list=PLFmoA27GocLc87HfGQtGfsQveHlUIErXc&index=2&ab_channel=Prof.Ren%C3%AAGusm%C3%A3oProf.Ren%C3%AAGusm%C3%A3o>

[2] GOLDBARG, Marco; GOLDBARG, Elizabeth. Grafos: conceitos, algoritmos e aplicações. Rio de Janeiro: Elsevier, 2012.

[3] TSP Data for the Traveling Salesperson Problem. Disponível em:

<https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html>