

Ciclos hamiltonianos e o problema do caixeiro viajante

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá



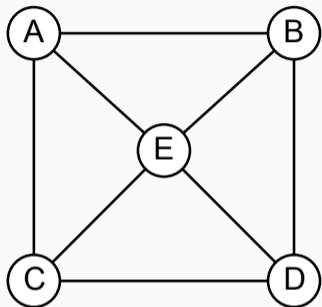
Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-Compartilhado 4.0 Internacional.

<http://github.com/malbarbo/na-grafos>

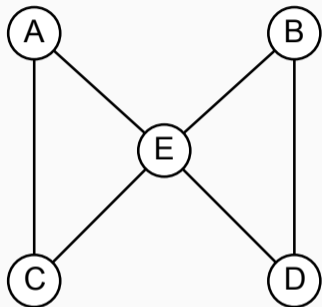
Um **caminho hamiltoniano** é um caminho que passa por cada vértice do grafo exatamente uma vez

Um **ciclo hamiltoniano** é um ciclo que passa por cada vértice do grafo exatamente uma vez (exceto o primeiro vértice que é visitado duas vezes)

Um **grafo hamiltoniano** é um grafo que tem um ciclo hamiltoniano



Este grafo é hamiltoniano:
A, C, E, D, B, A



Este grafo não é hamiltoniano

Como determinar se um grafo é hamiltoniano?

Teorema de Ore

- Uma condição suficiente (mas não necessária) para que um grafo $G = (V, E)$ seja hamiltoniano é que a soma dos graus de cada par de vértices não adjacentes seja no mínimo $|V|$

Teorema de Dirac

- Uma condição suficiente (mas não necessária) para que um grafo $G = (V, E)$ possua um ciclo hamiltoniano, é que o grau de cada vértice em G seja pelo menos igual a $\frac{|V|}{2}$

E algoritmos?

O problema de determinar se um grafo é hamiltoniano é NP-completo, portanto não são conhecidos algoritmos de tempo polinomial para resolvê-lo.

Podemos abordar a questão como um problema de otimização e utilizar algoritmos heurísticos.

Dado um grafo conexo com peso nas arestas, o **problema do caixeiro viajante** (do inglês *travelling salesman problem* - TSP) consiste em encontrar um ciclo hamiltoniano de peso mínimo.

Aplicações

- Planejamento de entregas
- Perfuração de placas de circuito impresso
- Sequenciamento de DNA (subproblema)

O TSP é NP-Difícil, o que implica que, a menos que $P = NP$, não existem algoritmos exatos de tempo polinomial para resolver este problema. Desta forma, é comum estudar algoritmos que não são exatos (não garantem encontrar o ótimo) mas executam em tempo polinomial.

Veremos os seguintes algoritmos

- Algoritmo aproximativo baseado na árvore geradora mínima
- Algoritmo heurístico construtivo vizinho mais próximo
- Algoritmo heurístico melhorativo 2-opt e 3-opt

Ideia

- Construir uma árvore geradora mínima é fazer um percurso pré-ordem

Característica

- Algoritmo aproximativo com fator de aproximação 2

Requisito

- A função de custo c tem que respeitar a desigualdade de triângulos (o caminho mais curto entre dois vértices não passa por vértice intermediário) isto é, para todo vértice $u, v, w \in V$

$$c(u, w) \leq c(u, v) + c(v, w)$$

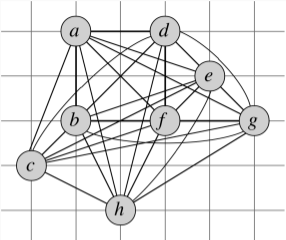
TSP-MST(G, c)

- 1 selecionar um vértice $r \in G.V$ para ser a raiz
- 2 computar a árvore geradora mínima T usando MST-PRIM(G, c, r)
- 3 seja H a lista de vértices ordenado de acordo com uma visitação em pré-ordem da árvore T
- 4 **return** o ciclo hamiltoniano H

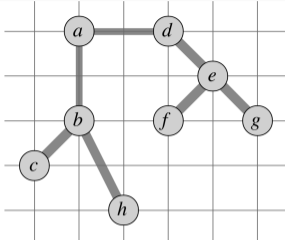
Análise do tempo de execução

- Com uma implementação do MST-PRIM usando arranjo e busca linear para fila de prioridade, o tempo de execução é $\Theta(V^2)$

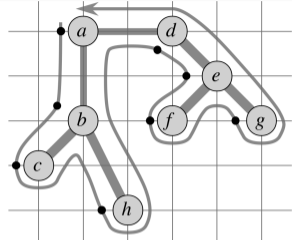
Algoritmo baseado na árvore geradora mínima



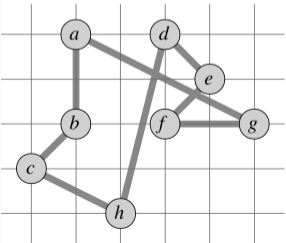
(a)



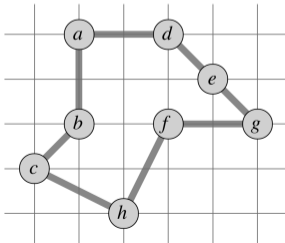
(b)



(c)



(d)



(e)

Algoritmo baseado na árvore geradora mínima

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>	—	2.0	3.2	2.0	3.2	2.8	4.5	4.1
<i>b</i>	2.0	—	1.4	2.8	3.2	2.0	4.0	2.2
<i>c</i>	3.2	1.4	—	4.2	4.5	3.2	5.1	2.2
<i>d</i>	2.0	2.8	4.2	—	1.4	2.0	2.8	4.1
<i>e</i>	3.1	3.2	4.5	1.4	—	1.4	1.4	3.6
<i>f</i>	2.8	2.0	3.1	2.0	1.4	—	2.0	2.2
<i>g</i>	4.5	4.0	5.1	2.8	1.4	2.0	—	3.6
<i>h</i>	4.1	2.2	2.2	4.1	3.6	2.2	3.6	—

Caminho encontrado por TSP-MST $\langle a, b, c, h, d, e, f, g, a \rangle$, custo = 19,1

Melhor caminho $\langle a, b, c, h, f, g, e, d, a \rangle$, custo = 14,7

Prova do fator de aproximação 2

Discutido em sala (veja a seção 35.2.1 do livro do Cormen)

Ideia

- Começar com um vértice qualquer
- A cada passo, escolher o vértice mais próximo do último vértice visitado
- Voltar para o primeiro vértice

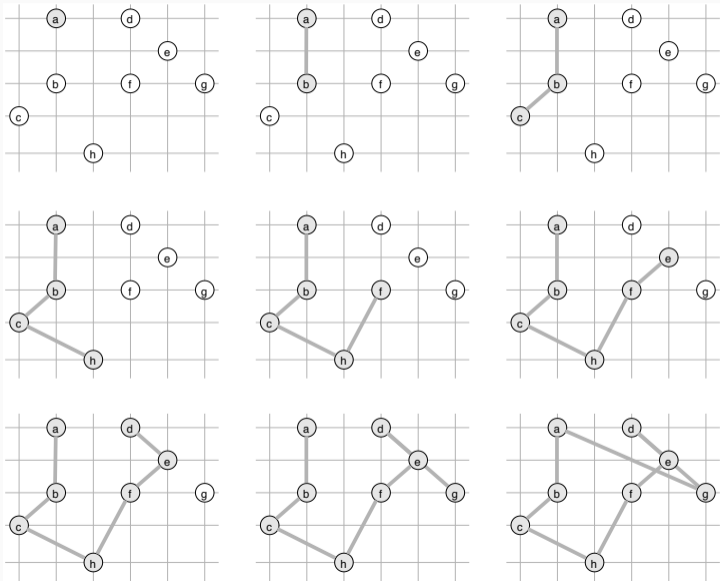
TSP-NN(G, c)

- 1 selecionar um vértice inicial $v_0 \in G.V$
- 2 $C = \langle v_0 \rangle$ // caminho contendo v_0
- 3 $v = v_0$
- 4 **while** C não inclui todos os vértices de G
- 5 $u =$ vértice mais próximo de v
 que não está em C
- 6 $C = C \cup \langle u \rangle$
- 7 $v = u$
- 8 $C = C \cup \langle v_0 \rangle$ // fecha o ciclo
- 9 **return** C

Análise do tempo de execução

- Encontrar o vértice mais próximo que não foi visitado (linha 5) usando uma busca linear custa $O(V)$
- Como cada vértice (a menos do primeiro) é colocado uma vez no ciclo, o tempo total de execução do algoritmo é $O(V^2)$

Heurística do vizinho mais próximo



Heurística do vizinho mais próximo

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>	—	2.0	3.2	2.0	3.2	2.8	4.5	4.1
<i>b</i>	2.0	—	1.4	2.8	3.2	2.0	4.0	2.2
<i>c</i>	3.2	1.4	—	4.2	4.5	3.2	5.1	2.2
<i>d</i>	2.0	2.8	4.2	—	1.4	2.0	2.8	4.1
<i>e</i>	3.1	3.2	4.5	1.4	—	1.4	1.4	3.6
<i>f</i>	2.8	2.0	3.1	2.0	1.4	—	2.0	2.2
<i>g</i>	4.5	4.0	5.1	2.8	1.4	2.0	—	3.6
<i>h</i>	4.1	2.2	2.2	4.1	3.6	2.2	3.6	—

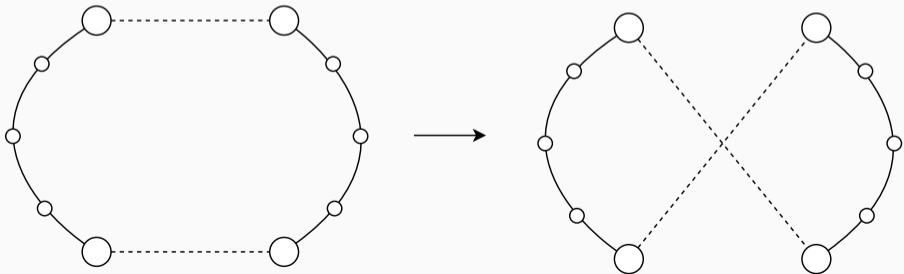
Caminho encontrado por TSP-MST $\langle a, b, c, h, f, e, d, g, a \rangle$, custo = 17,9

Melhor caminho $\langle a, b, c, h, f, g, e, d, a \rangle$, custo = 14,7

Usada para melhorar uma solução (obtida por algum algoritmo construtivo)

Ideia

- Eliminar 2 arestas não adjacentes e reconectar o caminho usado duas outras arestas, verificar se houve melhora
- Repetir este processo para todos os pares de arestas e realizar a melhor troca



Seja w a função de custo e t um percurso.

Vamos supor que o percurso t é representado por um vetor (índices $1..|V|$) de vértices. A aresta $(t_{|V|}, t_1)$ está implícita nesta representação.

Seja (t_a, t_b) e (t_c, t_d) duas arestas não adjacentes de t (a, b, c, d são os índices em t).

Seja t' o percurso (válido) obtido a partir de t trocando a aresta (t_a, t_b) por (t_a, t_c) e (t_c, t_d) por (t_b, t_d) .

Seja $\delta = w(t_a, t_b) + w(t_c, t_d) - w(t_a, t_c) - w(t_b, t_d)$ a diferença do custo de t e t' (quando maior a diferença, maior a melhoria).

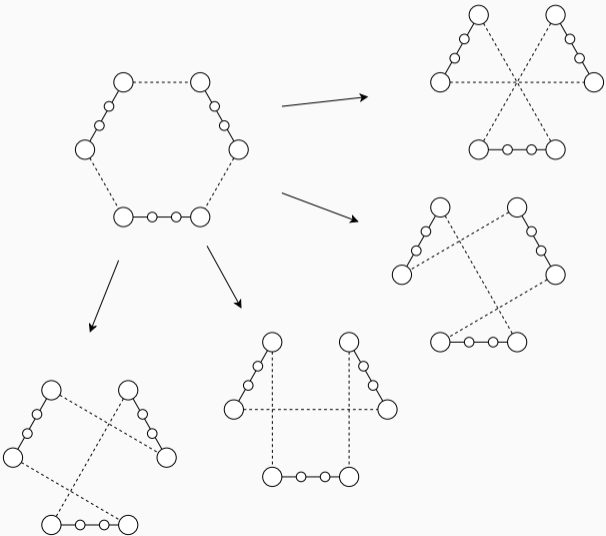
Seja δ_{\max} o maior valor entre os δ 's de todas as trocas de arestas.

Heurística 2-opt

```
tsp-2opt(t, w)
1 loop
2    $\delta_{\max} = 0$ 
3   best = nil
4   for a = 1..(n - 2)
5     b = a + 1
6     lim = n if a  $\neq$  1 else n - 1
7     for c = (a + 2)..lim
8       d = c + 1 if c  $\neq$  n else 1
9        $\delta = w(t_a, t_b) + w(t_c, t_d) - w(t_a, t_c) - w(t_b, t_d)$ 
10      if  $\delta > \delta_{\max}$ 
11         $\delta_{\max} = \delta$ 
12        best = a, b, c, d
13  if  $\delta_{\max} \neq 0$ 
14    a, b, c, d = best
15    t = (t - {(ta, tb), (tc, td)})  $\cup$  {(ta, tc), (tb, td)}
16  else
17    return t
```

Heurística 3-opt

Mesmo princípio do 2-opt, mas seleciona 3 arestas



Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 35.2.1.

Caminho hamiltoniano. Wikipédia. https://pt.wikipedia.org/wiki/Caminho_hamiltoniano

Problema do caminho hamiltoniano. Wikipédia.

https://en.wikipedia.org/wiki/Hamiltonian_path_problem

Problema do caixeiro viajante. Wikipédia.

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Algoritmo vizinho mais próximo. Wikipédia.

https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

Algoritmo de Christofides. Wikipédia. https://en.wikipedia.org/wiki/Christofides_algorithm