Algoritmos de aproximação

Marina Andretta

ICMC-USP

5 de novembro de 2019

Baseado no livro Uma introdução sucinta a Algoritmos de Aproximação, de M. H. Carvalho, M. R. Cerioli, R. Dahab, P. Feofiloff, C. G. Fernandes, C. E. Ferreira, K. S. Guimarães, F. K. Miyazawa, J. C. Piña Jr., J. A. R. Soares e Y. Wakabayashi.

Introdução

Problemas de otimização têm o objetivo de encontrar um ponto ótimo (mínimo ou máximo) de uma função definida sobre um certo domínio.

Os problemas de otimização combinatória são aqueles que têm domínio finito.

Apesar de os elementos do domínio geralmente poderem ser facilmente enumerados, testar todos os elementos na busca pelo melhor mostra-se inviável na prática, pois o domínio é tipicamente muito grande.

Introdução

O desenvolvimento de algoritmos de aproximação surgiu em resposta à dificuldade computacional de muitos dos problemas de otimização combinatória, que são *NP*-difíceis.

Nestes casos, pode valer a pena trocar a otimalidade por uma aproximação de boa qualidade que possa ser eficientemente calculada.

Esse compromisso entre perda de otimalidade e ganho em eficiência é o paradigma dos algoritmos de aproximação.

Introdução

É importante notar que um algoritmo de aproximação não é simplesmente uma heurística: ele garante encontrar, eficientemente, um elemento do domínio cujo valor tem uma relação pré-estabelecida com o valor ótimo.

É importante mencionar também o aparecimento de certos resultados negativos de aproximabilidade: para alguns problemas, aproximar é tão difícil quanto resolver.

Em termos mais técnicos, alguns problemas não admitem algoritmos de aproximação com razão melhor que um certo limiar, a menos que $P = NP$.

Problemas de otimização

Um problema de otimização tem três ingredientes principais:

- um conjunto de instâncias;
- um conjunto $Sol(I)$ de soluções viáveis (ou factíveis) para cada instância I;
- uma função que atribui um número $val(S)$ (chamado valor de S) a cada solução viável S.

Problemas de otimização

Quando o conjunto $Sol(I)$ das soluções viáveis associado a uma instância I é vazio, dizemos que a instância é inviável (ou infactível).

Um problema de minimização está interessado nas soluções viáveis de valor mínimo, enquanto um problema de maximização está interessado nas soluções viáveis de valor máximo.

Quando uma dessas alternativas (mínimo ou máximo) está subentendida, dizemos simplesmente valor ótimo e problema de otimização.

Exemplo

Problema: encontrar um ciclo hamiltoniano de custo mínimo em um grafo com custos nas arestas.

Uma instância deste problema consiste em um grafo G e uma função c que associa um número não-negativo a cada aresta de G.

O conjunto das soluções viáveis de uma instância (G, c) é o conjunto de todos os ciclos hamiltonianos de G.

O valor de um ciclo hamiltoniano C é $val(C) := \sum_{e \in C} c_e$.

Problemas de otimização

Uma solução viável cujo valor é ótimo é chamada solução ótima.

O valor de qualquer das soluções ótimas de uma instância I será denotado por $opt(I)$.

Portanto,

$$opt(I) := val(S^*),$$

em que S^* é uma solução ótima de I.

Algoritmos de aproximação

Considere um problema de otimização em que $val(S) \geq 0$ para toda solução viável S de qualquer instância do problema.

Seja A um algoritmo que, para toda instância viável I do problema, devolve uma solução viável $A(I)$ de I.

Se o problema é de minimização e

$$val(A(I)) \leq \alpha opt(I)$$

para toda instância I, dizemos que A é uma α-aproximação para o problema.

Algoritmos de aproximação

Dizemos que α é uma razão de aproximação do algoritmo.

É claro que $\alpha \geq 1$, uma vez que o problema é de minimização.

No caso de problema de maximização, basta refazer a definição com

$$val(A(I)) \geq \alpha opt(I).$$

É claro que, neste caso, $0 < \alpha < 1$.

Algoritmos de aproximação

Uma 1-aproximação para um problema de otimização é um algoritmo exato para o problema.

Observe que um algoritmo A é uma α-aproximação para um problema de minimização (maximização) se α é um limitante superior (inferior) para a razão entre $val(A(I))$ e $opt(I)$ para uma instância arbitrária I do problema.

Como o valor de $opt(I)$ é, em geral, tão difícil de calcular quanto uma solução ótima do problema, para demonstrar que um algoritmo é uma α-aproximação é essencial ter bons limitantes para o valor de $opt(I)$.

Notação básica

Dada uma função c que associa um número a cada elemento e de um conjunto finito E, denotamos por $c(F)$ a soma dos valores de c nos elementos de $F \subseteq E$:

$$c(F) := \sum_{f \in F} c_f .$$

Problema de Escalonamento

Um problema bastante conhecido nos contextos de produção industrial e de sistemas operacionais é o de escalonamento (*scheduling*) de tarefas em máquinas.

Estamos interessados em uma das versões mais simples do problema: dadas m máquinas idênticas e n tarefas com tempos de execução pré-determinados, encontrar uma atribuição das tarefas às máquinas que minimize o tempo máximo de operação de qualquer uma das máquinas (*makespan*).

Problema de Escalonamento

Formalmente, o problema do escalonamento em máquinas idênticas (*multiprocessor scheduling problem*) consiste no seguinte:

Problema $\textsc{Escalonamento}(m, n, t)$: dados inteiros positivos m, n e um tempo t_i em $\mathbb{Q}_{\geq}$ para cada i em $\{1, ..., n\}$, encontrar uma partição $\{M_1, ..., M_m\}$ de $\{1, ..., n\}$ que minimize $\max_j t(M_j)$.

Problema de Escalonamento

De acordo com nossa notação, $t(M_j) := \sum_{i \in M_j} t_i$.

Dizemos que uma partição de $\{1, ..., n\}$ em m blocos é um escalonamento e que o número $\max_j t(M_j)$ é o custo do escalonamento.

Com essa terminologia, o problema pode ser formulado como: dados m, n e t, encontrar um escalonamento de custo mínimo.

Problema de Escalonamento

Este problema é *NP*-difícil mesmo para duas máquinas, ou seja, quando $m = 2$.

O primeiro algoritmo de aproximação para o problema foi descrito e analisado por Graham e usa um critério muito simples: alocar as tarefas uma a uma, destinando cada tarefa à máquina menos ocupada.

Por esse critério, a escolha da máquina que vai receber determinada tarefa não depende dos tempos das tarefas que ainda não foram atribuídas a nenhuma máquina.

Algoritmo de aproximação para Escalonamento

Algoritmo Escalonamento-Graham(m, n, t):

1 para j de 1 a m, faça $M_j \leftarrow \emptyset$;

2 para i de 1 a n, faça

3 seja k uma máquina tal que $t(M_k)$ é mínimo;

4 faça $M_k \leftarrow M_k \cup \{i\}$;

5 devolva $\{M_1, ..., M_m\}$.

Algoritmo de aprox. para ESCALONAMENTO - exemplo

Considere uma instância do problema ESCALONAMENTO com

- 3 máquinas ($m = 3$),
- 7 tarefas ($n = 7$) e
- t_i, $i = 1, ..., 7$, dados por

t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6

Algoritmo de aprox. para ESCALONAMENTO - exemplo

t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6

M_1

M_2

M_3

Algoritmo de aprox. para ESCALONAMENTO - exemplo

$\vee$

t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6

M_1 t_1

M_2

M_3

Algoritmo de aprox. para ESCALONAMENTO - exemplo

$\vee$

t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6

M_1 t_1

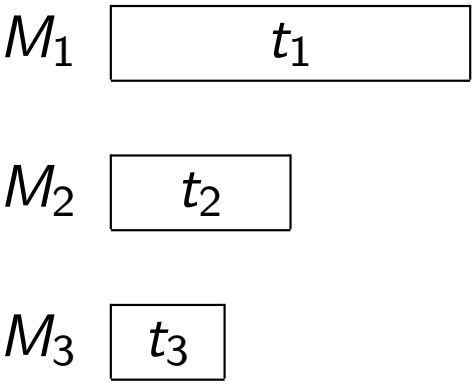
M_2 t_2

M_3

Algoritmo de aprox. para ESCALONAMENTO - exemplo

$\vee$

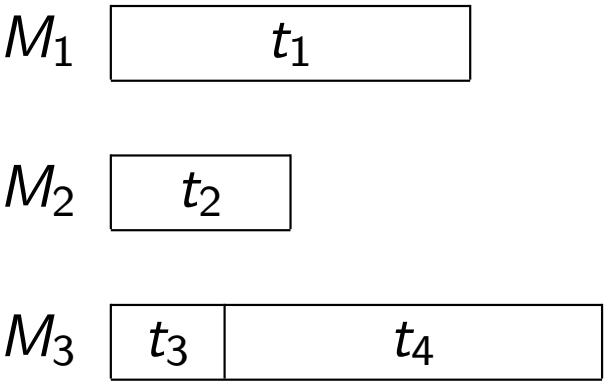
t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6



Algoritmo de aprox. para ESCALONAMENTO - exemplo

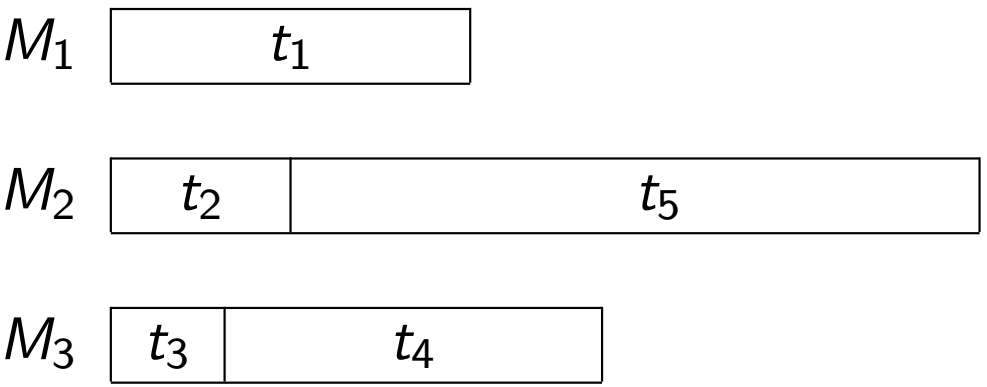
$\vee$

t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6



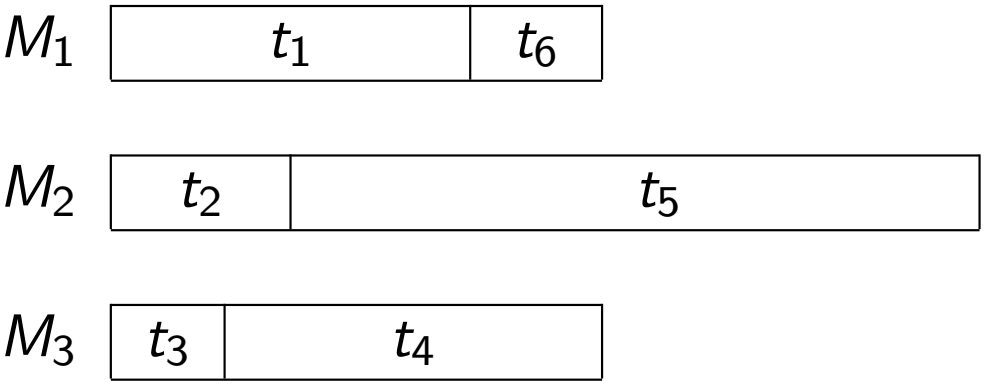
Algoritmo de aprox. para ESCALONAMENTO - exemplo

				∨		
t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6



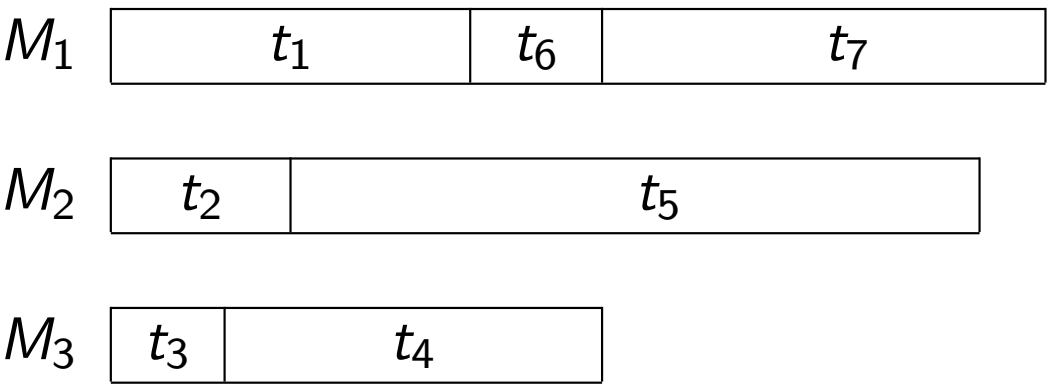
Algoritmo de aprox. para ESCALONAMENTO - exemplo

					∨	
t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6



Algoritmo de aprox. para ESCALONAMENTO - exemplo

						∨
t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6



Algoritmo de aproximação para Escalonamento

Claramente, ao final do algoritmo Escalonamento-Graham, $\{M_1, ..., M_m\}$ é uma partição de $\{1, ..., n\}$, ou seja, um escalonamento.

Há dois limitantes simples para o custo $opt(m, n, t)$ de um escalonamento ótimo: o tempo da tarefa mais longa e o custo que obteríamos se pudéssemos distribuir as tarefas por igual entre as m máquinas.

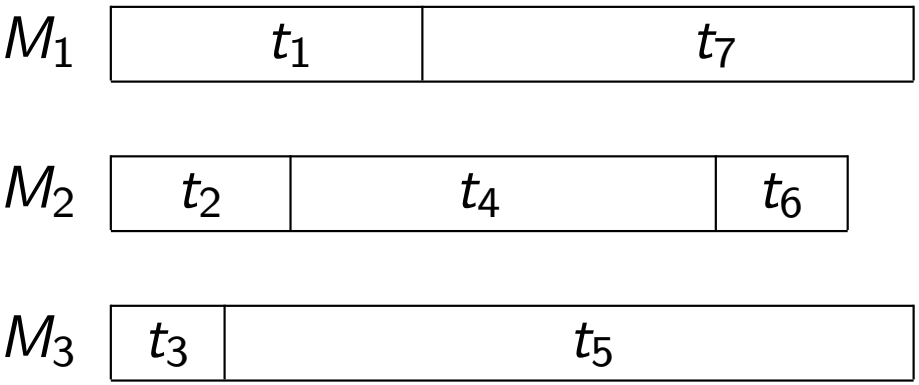
Assim, temos

$$opt(m, n, t) \geq \max_i t_i \quad \text{e} \quad opt(m, n, t) \geq \frac{1}{m} \sum_{i=1}^{n} t_i.$$

Outro escalonamento para o exemplo

Na instância do exemplo, um possível escalonamento é

t_1	t_2	t_3	t_4	t_5	t_6	t_7
4	2	1	5	9	2	6



Outro escalonamento para o exemplo

Como

$$\frac{1}{m}\sum_{i=1}^{n} t_i = \frac{1}{3}(4+2+1+5+9+2+6) = \frac{29}{3} = 9.666...,$$

e o escalonamento obtido tem valor 10, sabemos que ele é ótimo.

Algoritmo de aproximação para Escalonamento

Teorema: O algoritmo Escalonamento-Graham é uma 2-aproximação polinomial para o Escalonamento(m, n, t), quando $n \geq m$.

Algoritmo de aproximação para Escalonamento

Demonstração: claramente, o algoritmo consome tempo polinomial em n, já que $m \leq n$. Ou seja, ele é um algoritmo polinomial.

Vamos mostrar agora que ele é uma 2-aproximação polinomial para o Escalonamento(m, n, t).

Seja τ o valor de $t(M_k)$ imediatamente antes da execução da linha 4 do algoritmo em uma iteração qualquer.

É claro que $\tau \leq t(M_j)$ para todo j.

Assim,

$$\tau \leq \frac{1}{m} \sum_{j=1}^{m} t(M_j) \leq \frac{1}{m} \sum_{i=1}^{n} t_i.$$

Como $\frac{1}{m} \sum_{i=1}^{n} t_i \leq opt(m, n, t)$, segue que

$$\tau \leq opt(m, n, t).$$

Algoritmo de aproximação para Escalonamento

Assim, imediatamente depois da execução da linha 4 do algoritmo, temos

$$t(M_k) = \tau + t_i.$$

Como $\max_i t_i \leq opt(m, n, t)$, temos

$$t(M_k) \leq 2opt(m, n, t).$$

Algoritmo de aproximação para Escalonamento

Este limitante vale no fim de cada iteração, ou seja, ela vale cada vez que uma tarefa i é atribuida a uma máquina k. Portanto, o limitante se aplica a cada uma das máquinas.

Logo, no fim da última iteração,

$$\max_j t(M_j) \leq 2opt(m, n, t).$$

Portanto, o algoritmo Escalonamento-Graham é uma 2-aproximação polinomial para o Escalonamento(m, n, t), quando $n \geq m$.

Algoritmo de aproximação para Escalonamento

Como visto, o algoritmo processa os dados sem conhecimento prévio da sua totalidade.

Várias situações reais demandam esse tipo de abordagem, como o escalonamento de tarefas em processadores, que precisa de algoritmos rápidos e que forneçam soluções viáveis cujo valor seja próximo do valor ótimo (como acontece com o algoritmo Escalonamento-Graham).

Esse foi o primeiro algoritmo de aproximação de que se tem notícia, o que o torna um marco nessa teoria.

Problema do caixeiro viajante

Lembre-se que um ciclo hamiltoniano é um ciclo que contém todos os vértices do grafo.

O problema do caixeiro viajante (*traveling salesman problem*), denotado por TSP, é definido da seguinte maneira:

Problema $\mathrm{TSP}(G, c)$: Dados um grafo G e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, determinar um ciclo hamiltoniano C que minimize $c(C)$.

Problema do caixeiro viajante

Esse é talvez o mais famoso problema de otimização combinatória, em parte graças às conexões com vários outros problemas de otimização.

Ele é NP-difícil mesmo se $c_e \in \{1, 2\}$ para toda aresta e.

Além disso, não se conhece um algoritmo de aproximação com razão constante para o problema.

Nos restringimos a um caso particular do TSP que admite algoritmo de aproximação com razão constante.

Caixeiro viajante métrico

Suponha que o grafo G é completo e temos um custo c_{ij} associado a cada par ij de vértices.

Dizemos que os custos satisfazem a desigualdade triangular se

$$c_{ik} \leq c_{ij} + c_{jk}$$

para quaisquer três vértices i, j e k.

Caixeiro viajante métrico

O TSP restrito ao conjunto de instâncias (G, c) em que G é completo e c satisfaz a desigualdade triangular é conhecido como problema do caixeiro viajante métrico e será denotado aqui por TSPM.

Este problema também é NP-difícil. Além disso, foi provado que, a menos que $P = NP$, a melhor razão de aproximação polinomial possível para este problema é de $\frac{123}{122}$.

Preliminares

Antes de apresentarmos dois algoritmos de aproximação para o TSPM, precisamos de algoritmos polinomiais para resolver três problemas importantes:

- árvore geradora de custo mínimo;
- circuito euleriano;
- emparelhamento perfeito.

Problema da árvore geradora de custo mínimo

Problema $\mathrm{MST}(G, c)$: Dados um grafo G e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, encontrar uma árvore geradora de custo mínimo.

Existem algoritmos simples e eficientes para construir uma árvore geradora de custo mínimo em um grafo conexo.

Vamos designar por MST um algoritmo qualquer desse tipo.

Problema da árvore geradora de custo mínimo

Um algoritmo para resolver este problema, com $G = (V, E)$ um grafo conexo, é o algoritmo de Kruskal, proposto em 1956.

Algoritmo MST-KRUSKAL(G, c):

1 Faça $T \leftarrow \emptyset$;

2 faça $A \leftarrow E$;

3 enquanto $A \neq \emptyset$ e T não é uma árvore geradora, faça:

4 seja e uma aresta de A com menor custo c_e;

5 faça $A \leftarrow A \setminus \{e\}$;

6 se $T \cup \{e\}$ não contém um circuito

7 então $T \leftarrow T \cup \{e\}$;

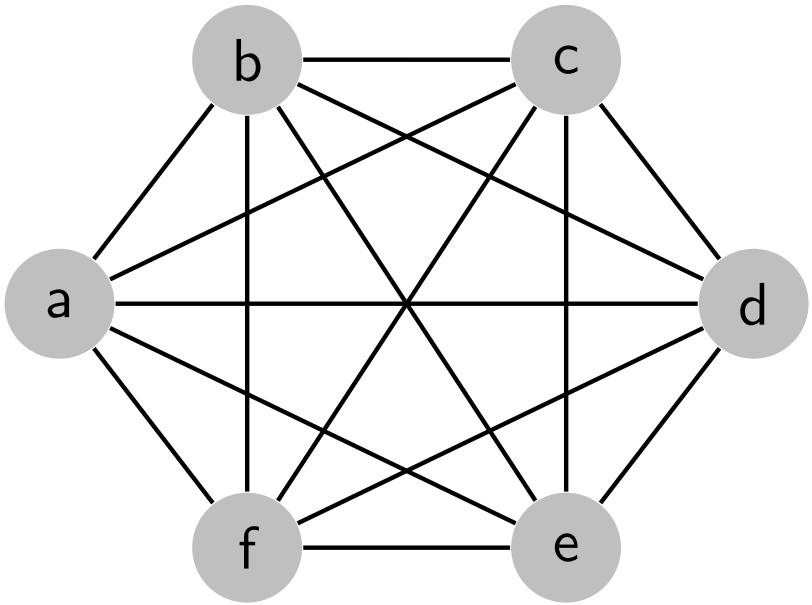
8 devolva T.

Problema da árvore geradora de custo mínimo

Claramente, este algoritmo é polinomial no número de vértices e arestas de G.

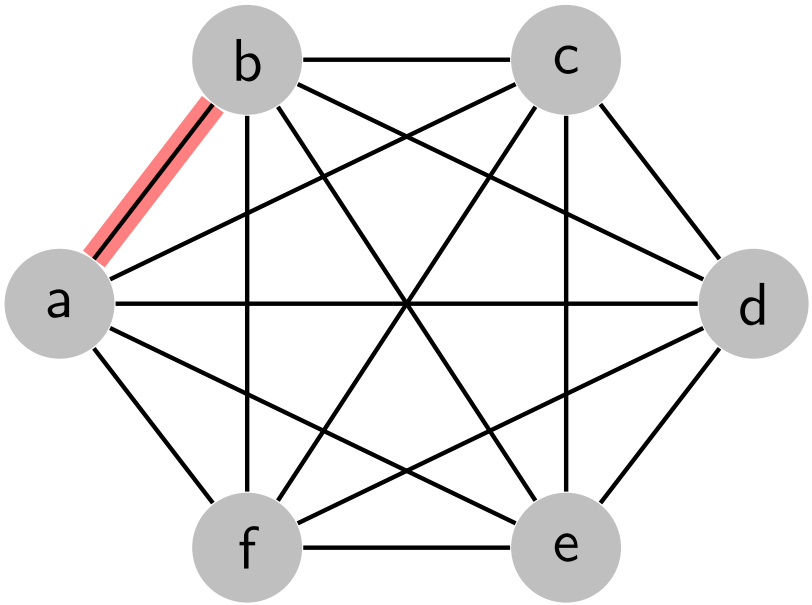
Vejamos como ele funciona através de um exemplo.

Problema da árvore geradora de custo mínimo



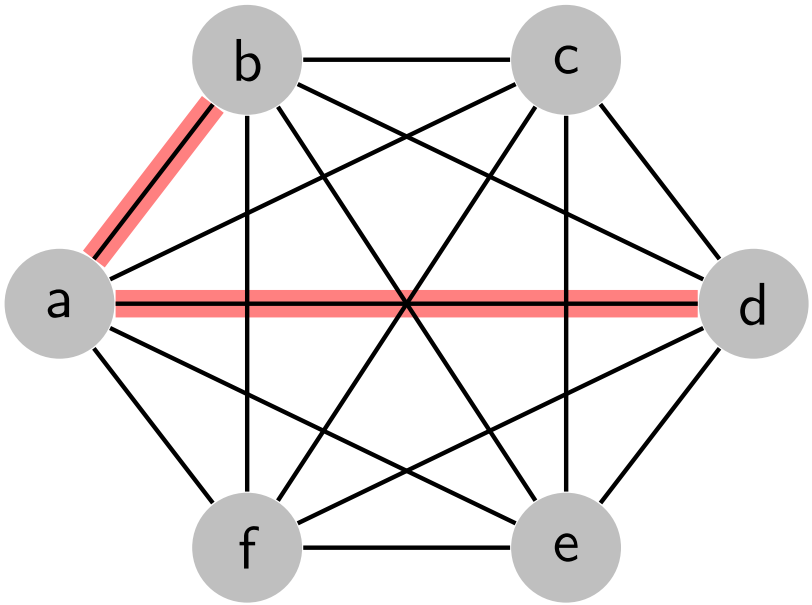
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



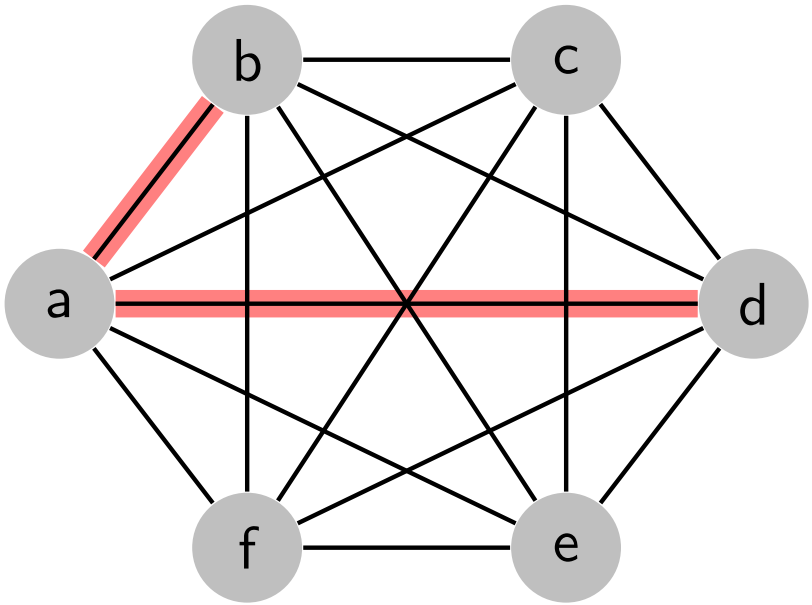
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



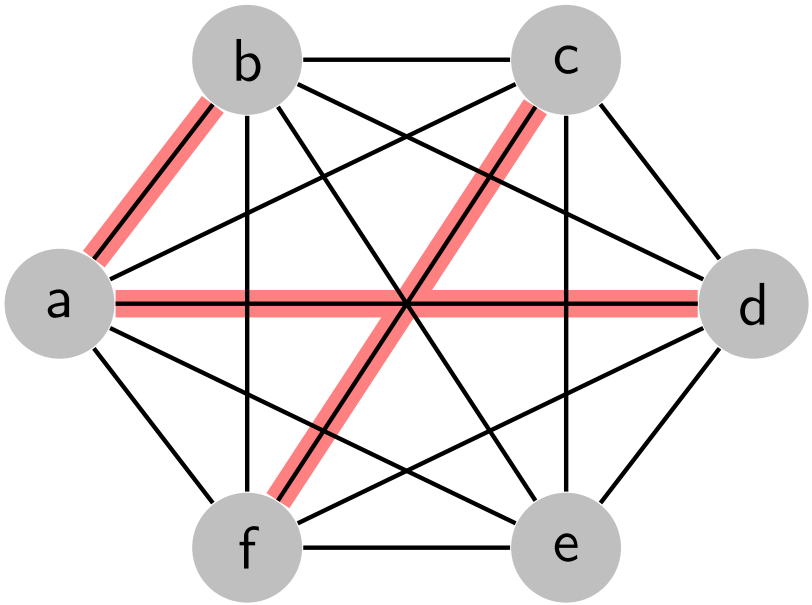
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



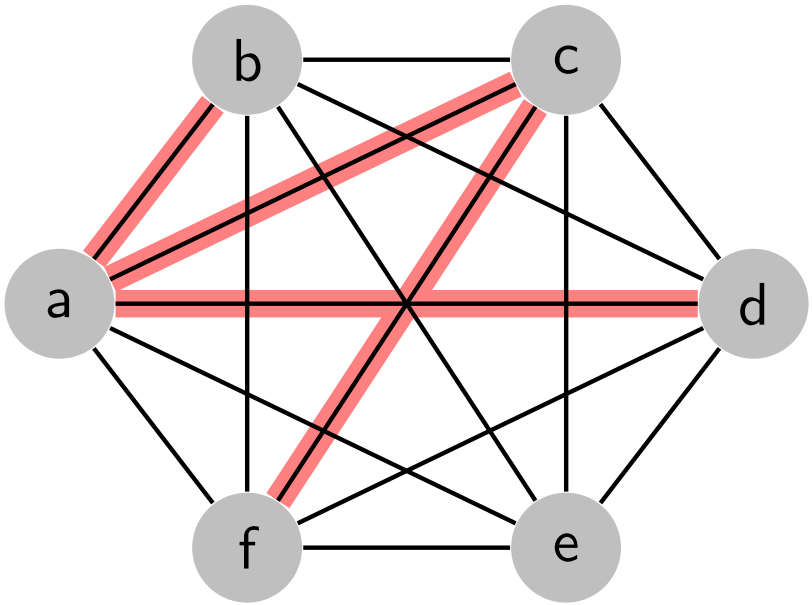
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



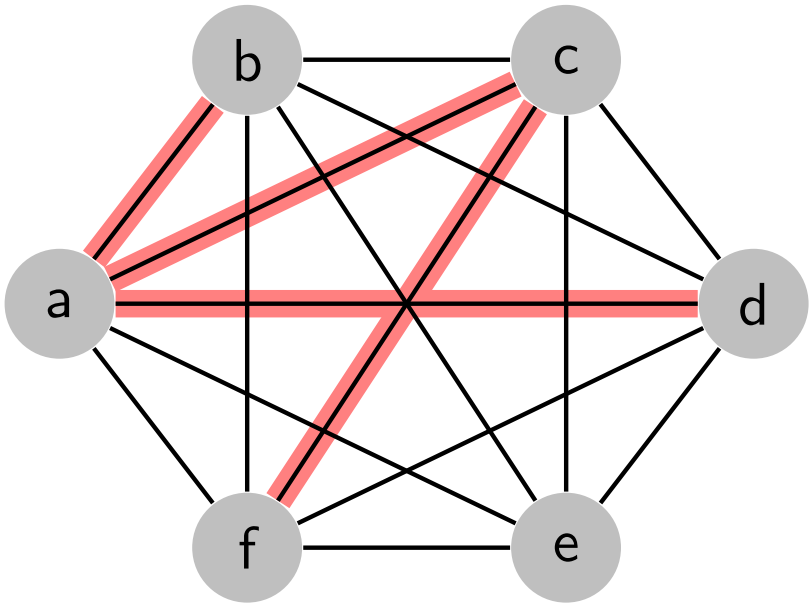
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



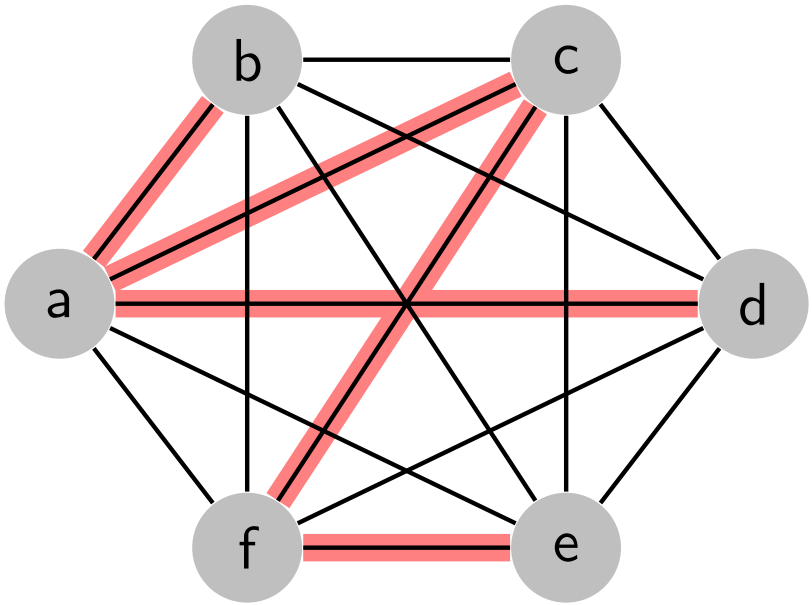
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



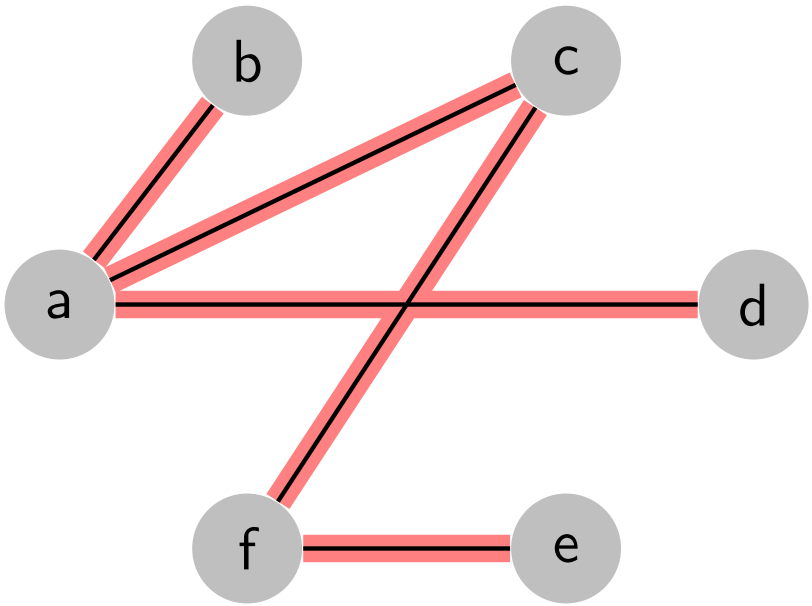
	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



	a	b	c	d	e	f
a	-	1	2	1	7	2
b	1	-	7	1	4	3
c	2	7	-	3	5	1
d	1	1	3	-	8	5
e	7	4	5	8	-	2
f	2	3	1	5	2	-

Problema da árvore geradora de custo mínimo



	a	b	c	d	e	f
a		1	2	1		
b	1					
c	2					1
d	1					
e						2
f			1		2	

Temos que $c(T) = 7$.

Problema do circuito euleriano

Um circuito euleriano em um grafo ou multigrafo G é qualquer circuito que contém todas as arestas de G.

Um multigrafo conexo G tem um circuito euleriano se e somente se cada um de seus vértices tem grau par.

São bem conhecidos os algoritmos que constroem um circuito euleriano em um multigrafo conexo sem vértices de grau ímpar. Vamos designar por EULER um algoritmo qualquer desse tipo.

Problema do circuito euleriano

Um algoritmo para resolver este problema para um grafo G conexo, com todos os vértices com grau par, é o proposto por Hierholzer, em 1873.

Problema do circuito euleriano

Algoritmo EULER-HIERHOLZER(G):

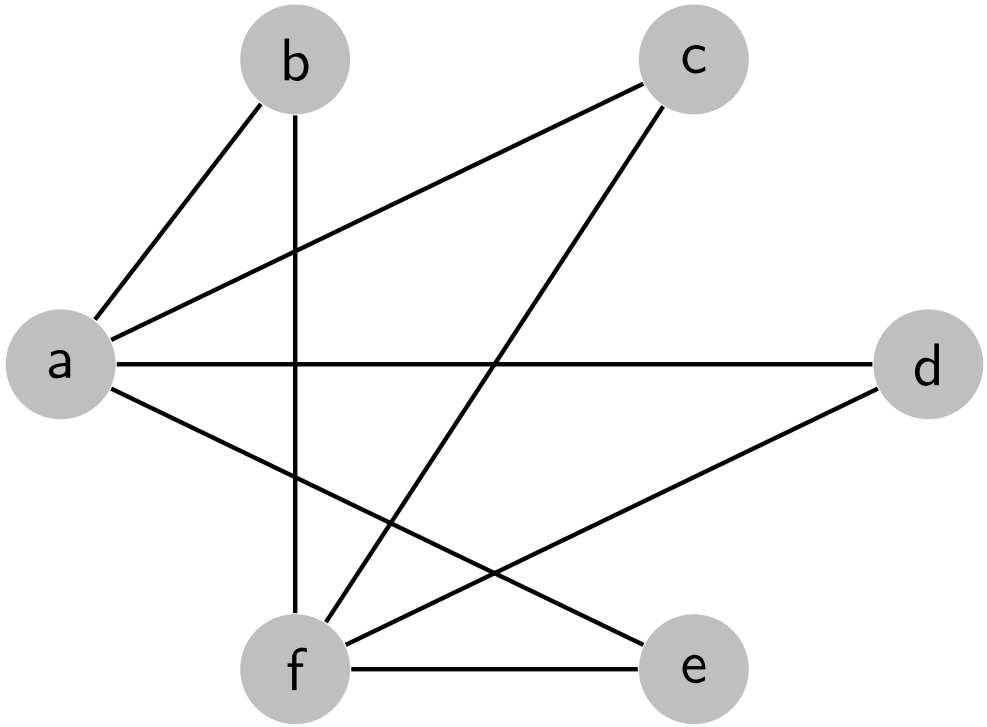
1 Faça $C \leftarrow \emptyset$ e $A \leftarrow E$;
2 seja v um vértice qualquer de G;
3 acrescente v à sequência de vértices C;
4 enquanto $A \neq \emptyset$, faça:
5 se não existe nenhuma aresta vw em A,
6 então escolha um vértice v de C tal que exista $vw \in A$;
7 escolha uma aresta vw de A;
7 acrescente w depois de v na sequência de vértices C;
8 faça $v \leftarrow w$;
9 faça $A \leftarrow A \setminus \{vw\}$;
10 devolva C.

Problema do circuito euleriano

Claramente, o consumo de tempo do algoritmo EULER-HIERHOLZER é proporcional ao número de arestas do grafo G.

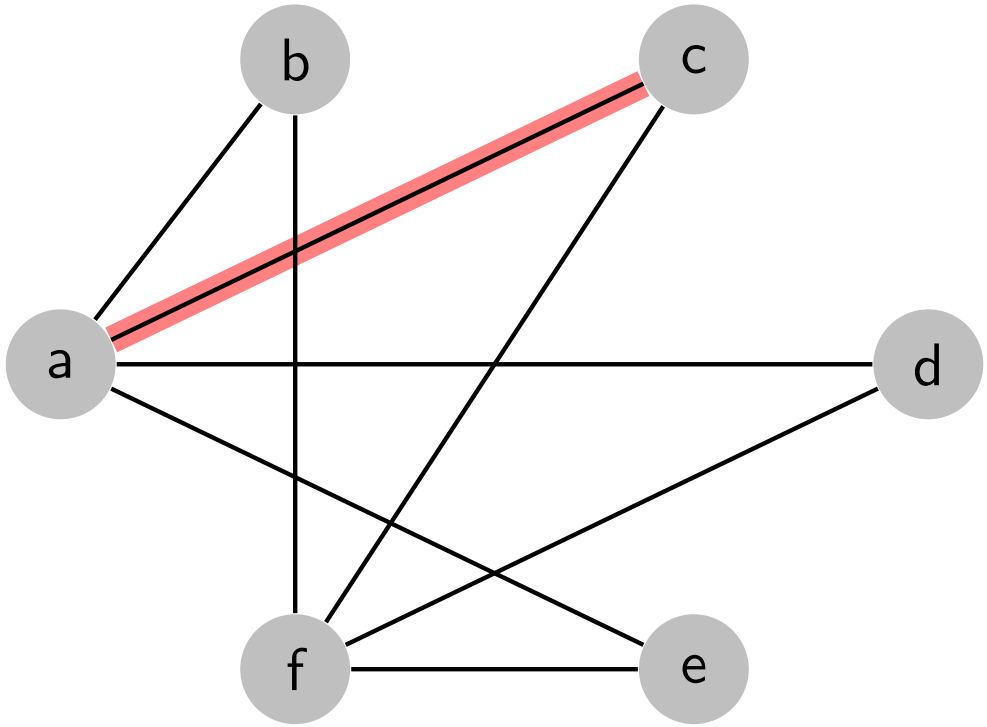
Vejamos um exemplo da execução do algoritmo.

Problema do circuito euleriano



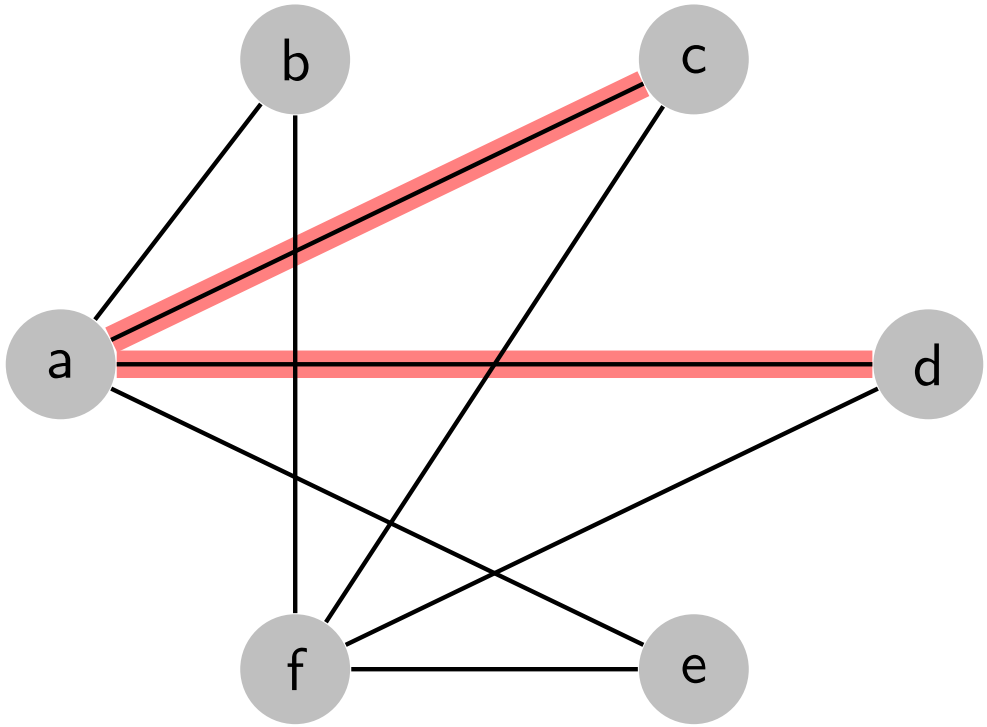
$$C = (c$$

Problema do circuito euleriano



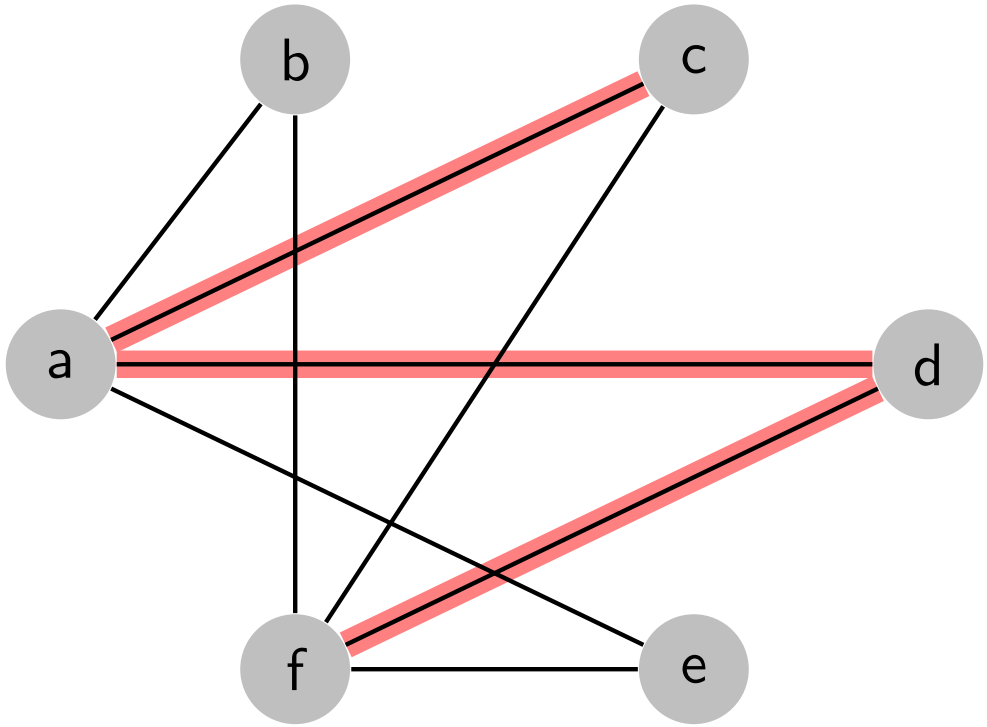
$$C = (c, a$$

Problema do circuito euleriano



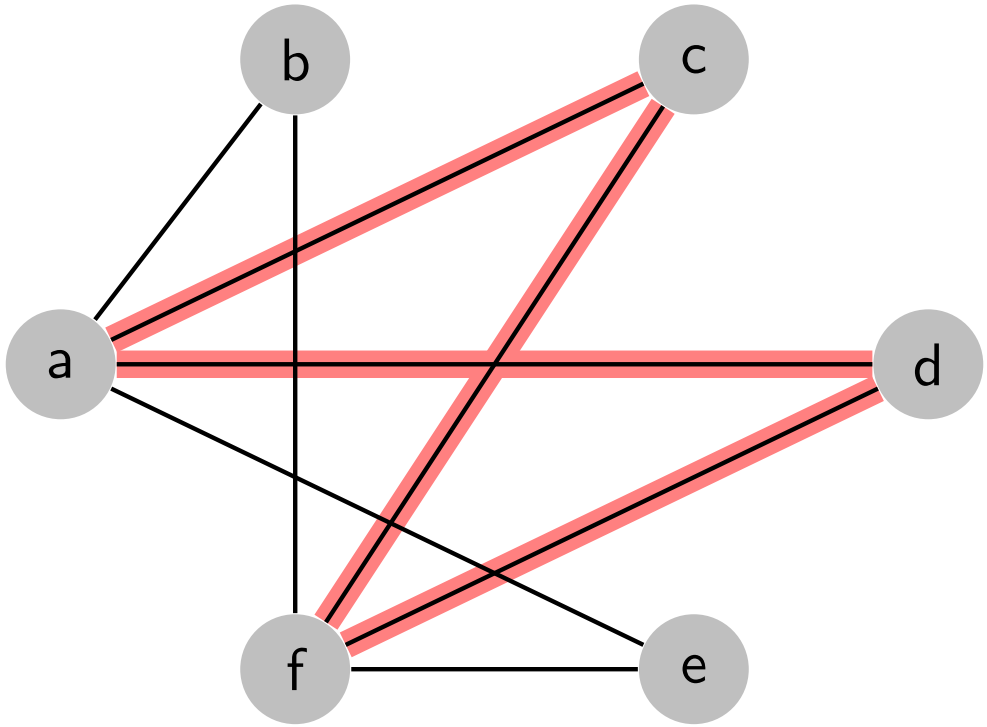
$$C = (c, a, d$$

Problema do circuito euleriano



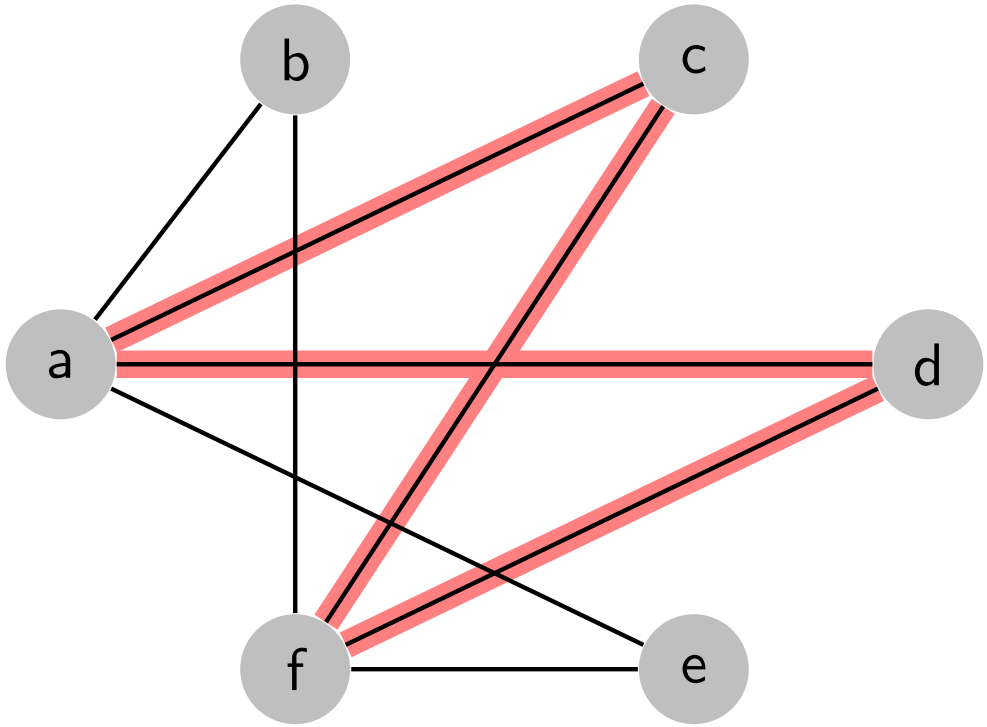
$$C = (c, a, d, f$$

Problema do circuito euleriano



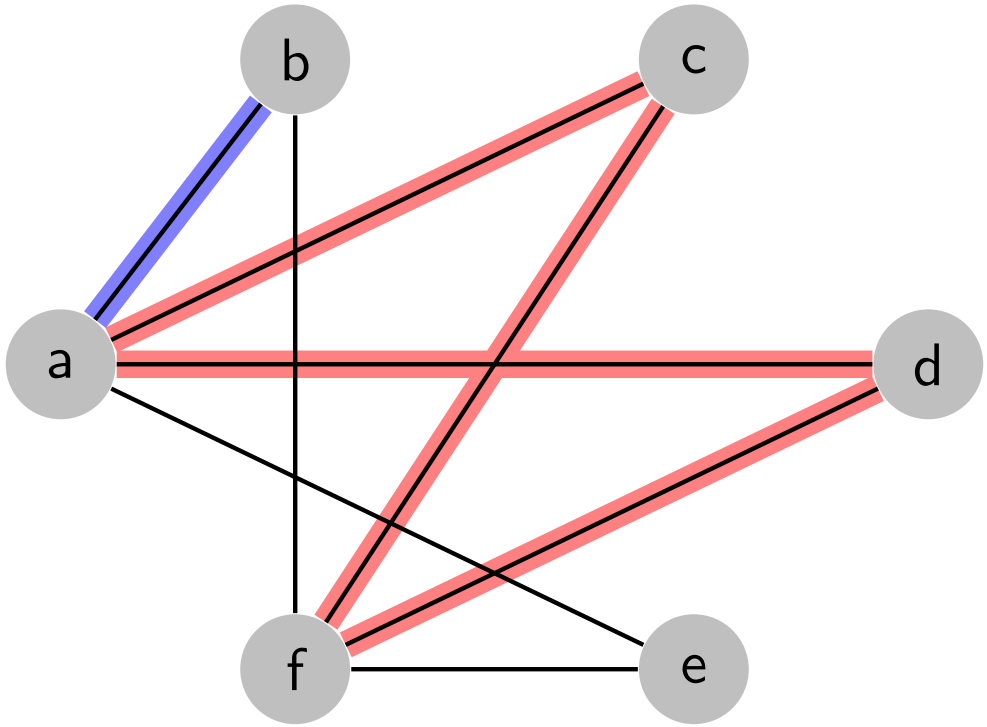
$$C = (c, a, d, f, c)$$

Problema do circuito euleriano



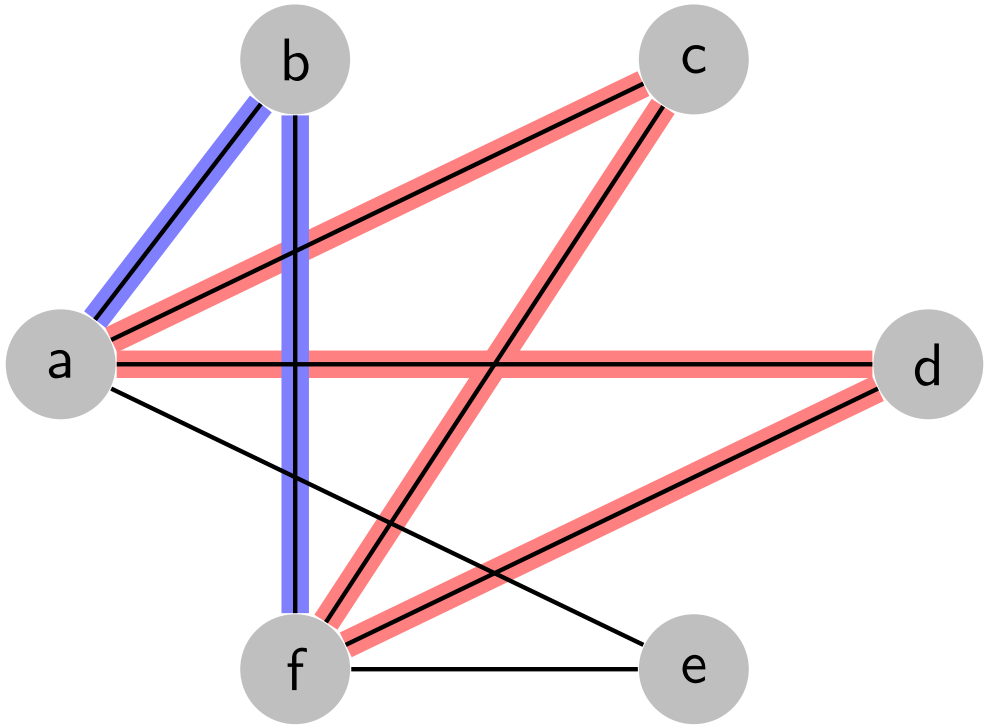
$$C = (c, a, d, f, c)$$

Problema do circuito euleriano



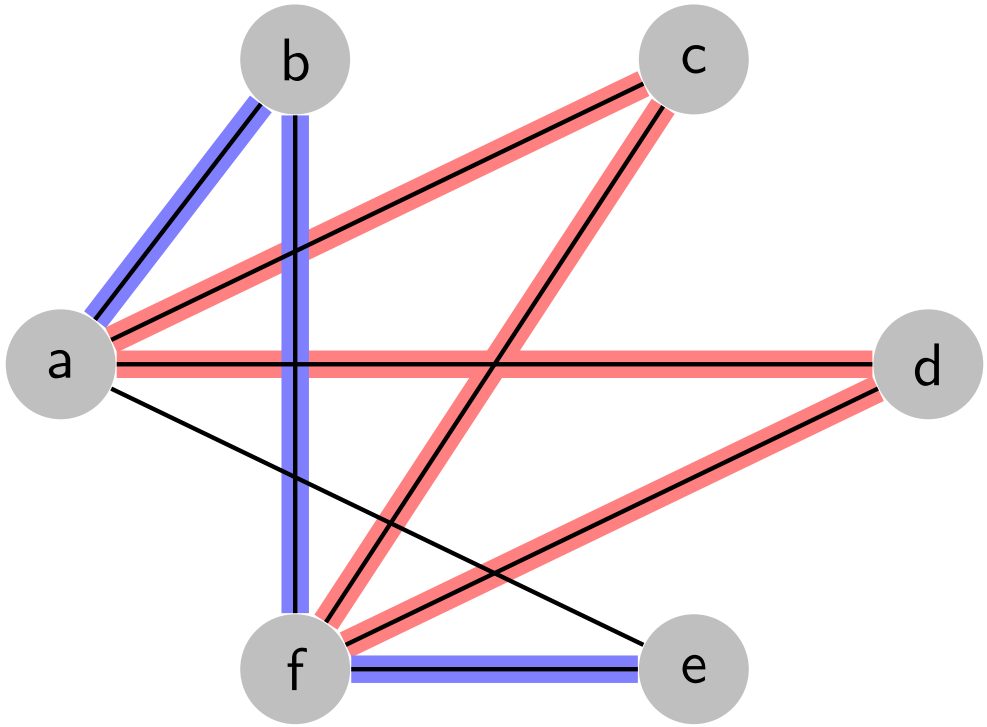
$$C = (c, a, b, d, f, c)$$

Problema do circuito euleriano



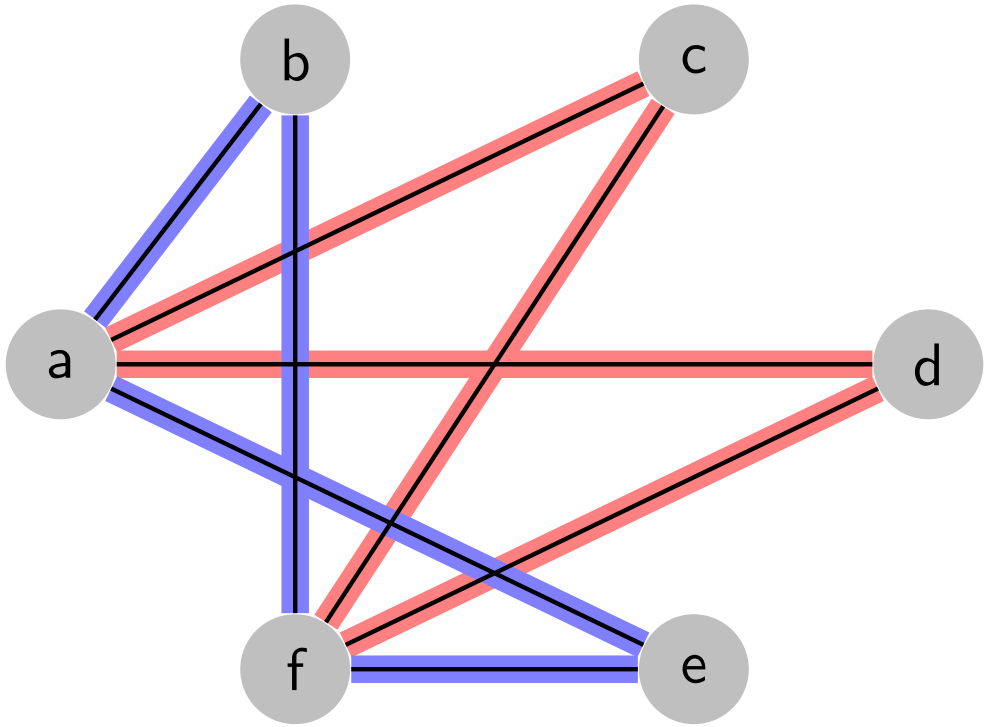
$$C = (c, a, b, f, d, f, c)$$

Problema do circuito euleriano



$$C = (c, a, b, f, e, d, f, c)$$

Problema do circuito euleriano



$$C = (c, a, b, f, e, a, d, f, c)$$

Problema do emparelhamento perfeito

Um emparelhamento em um grafo G é um conjunto de arestas sem extremos em comum, ou seja, cada vértice pertence a no máximo uma das arestas do emparelhamento.

Um emparelhamento M é perfeito se todo vértice de G pertence a alguma aresta de M.

O algoritmo de Edmonds, que denotaremos por EDMONDS, encontra um emparelhamento perfeito de custo mínimo em tempo $O(n^3)$, com n o número de vértices do grafo.

Algoritmos de aproximação para o TSPM

Agora que já vimos como resolver alguns problemas, vamos definir dois algoritmos de aproximação para o TSPM.

A estratégia utilizada pelos dois algoritmos tem quatro passos:

1. construir uma árvore geradora T de G;
2. acrescentar novas arestas a T para obter um novo grafo T' cujos vértices têm grau par;
3. obter um circuito euleriano P em T';
4. obter um ciclo hamiltoniano em G a partir de P.

A diferença entre os dois algoritmos está apenas na política adotada para acrescentar novas arestas à árvore T.

Algoritmos de aproximação para o TSPM

Uma árvore geradora de custo mínimo, calculada no passo 1 da estratégia, dá uma boa delimitação inferior para o valor ótimo do problema $\mathrm{TSPM}(G, c)$: se removemos uma aresta de um ciclo hamiltoniano temos uma árvore geradora de custo não superior ao do ciclo.

Portanto,

$$opt(G, c) \geq c(T).$$

Algoritmos de aproximação para o TSPM

O passo 2 da estratégia pode ser formalizado da seguinte maneira: para qualquer conjunto F de pares não-ordenados de vértices de T, seja $T + F$ o multigrafo $(V_T, E_T \dot{\cup} F)$, em que $E_T \dot{\cup} F$ denota o multiconjunto que tem duas cópias de cada elemento de $E_T \cap F$.

Como o grafo G, do qual T é uma árvore geradora, é completo, cada aresta do multigrafo $T + F$ tem um custo bem definido.

Após a execução do passo 2, temos a garantia que T' tem algum circuito euleriano. O passo 3 encontra um destes circuitos.

Algoritmos de aproximação para o TSPM

O passo 4 da estratégia transforma um ciclo gerador, ou seja, um ciclo que contém todos os vértices do multigrafo, em um ciclo hamiltoniano.

Para isso, basta extrair uma subsequência maximal sem vértices repetidos da sequência $(v_0, v_1, ..., v_m)$ de vértices do ciclo gerador.

Algoritmos de aproximação para o TSPM

Algoritmo ATALHO(P):

1 Seja $P = (v_0, v_1, ..., v_m, v_0)$;

2 $w_0 \leftarrow v_0$

3 $n \leftarrow 0$

4 para i de 1 a m, faça:

5 se $v_i \notin \{w_0, ..., w_n\}$

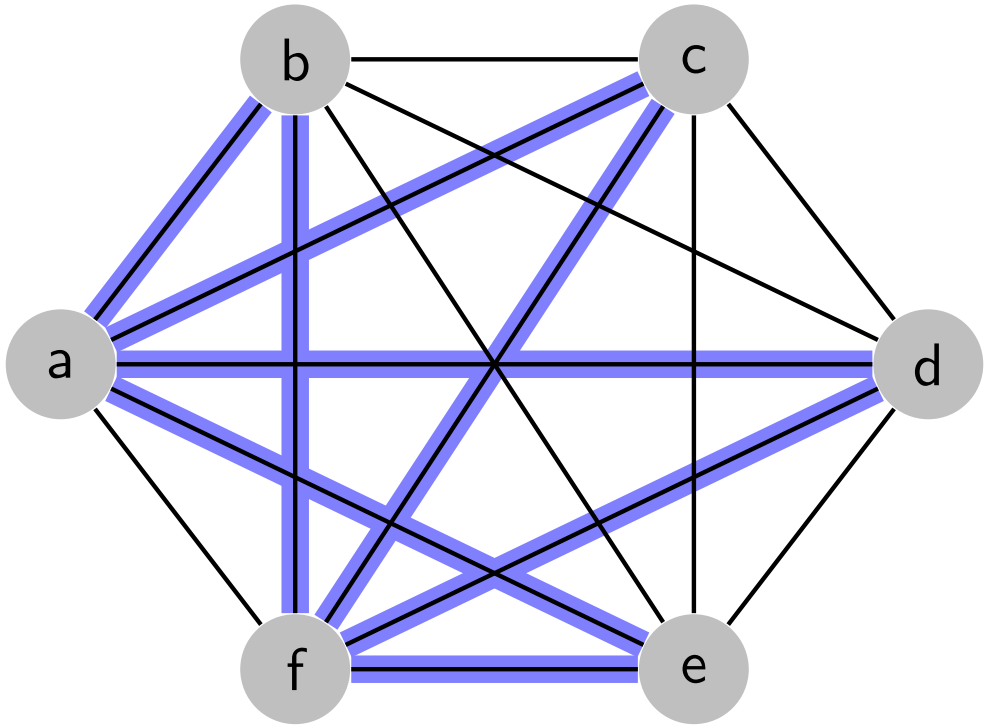
6 então $n \leftarrow n + 1$;

7 $w_n \leftarrow v_i$;

8 faça $C \leftarrow (w_0, w_1, ..., w_n, w_0)$;

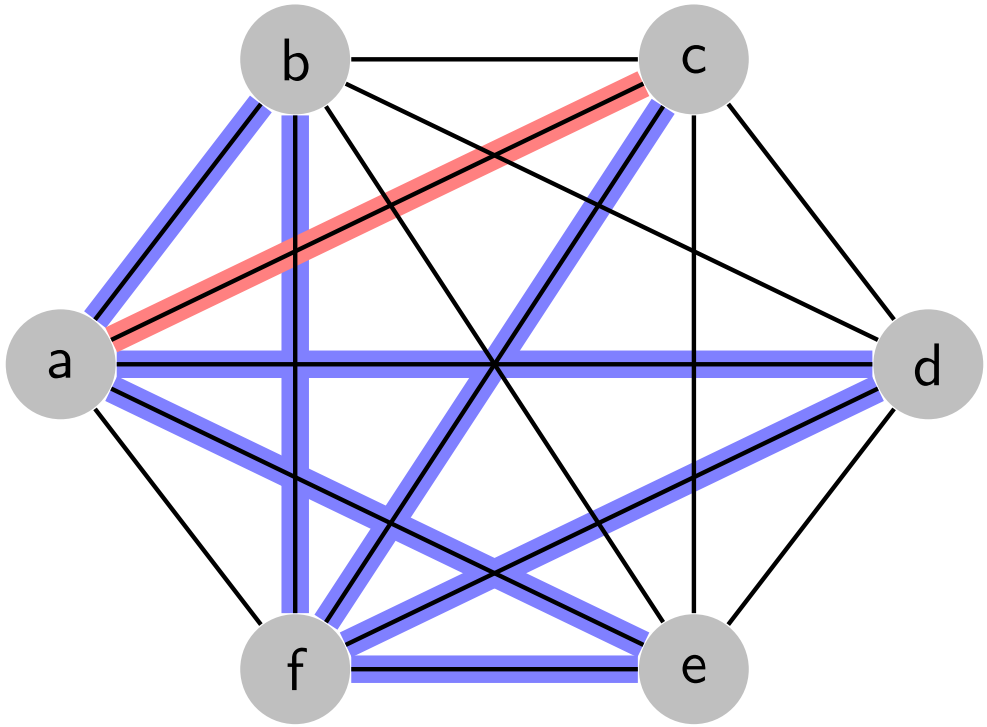
9 devolva C.

Algoritmos de aproximação para o TSPM



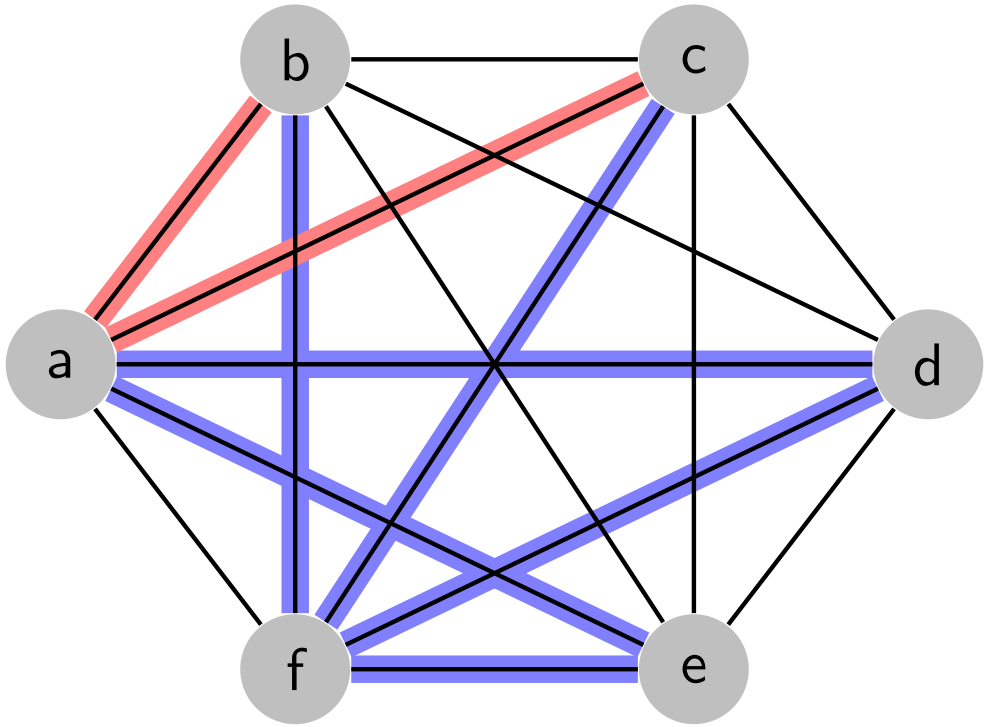
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c$$

Algoritmos de aproximação para o TSPM



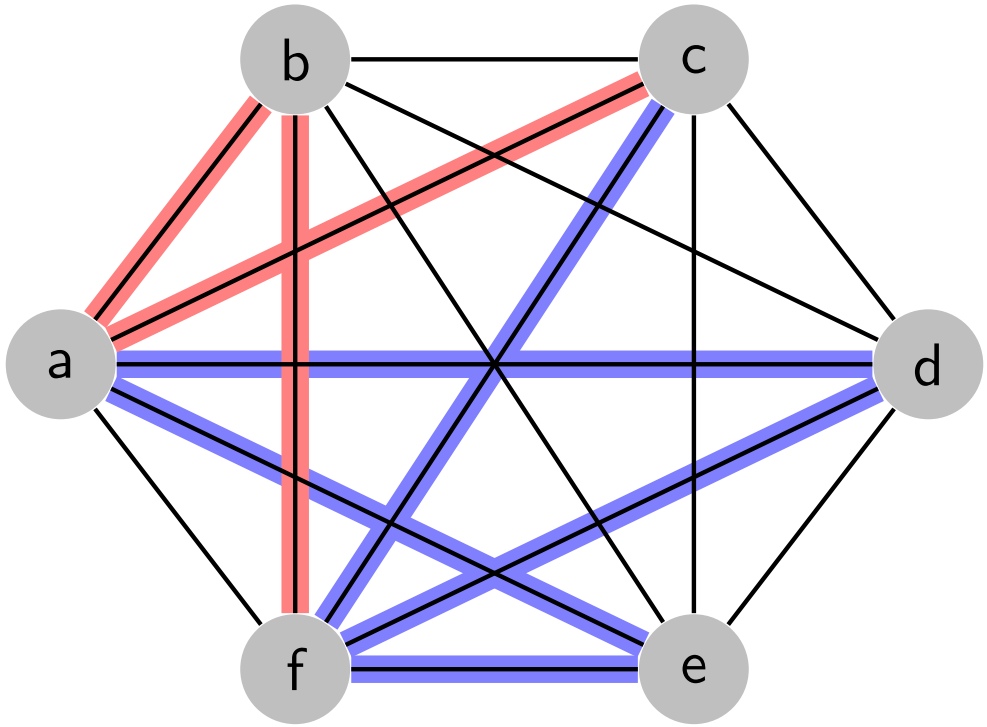
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a$$

Algoritmos de aproximação para o TSPM



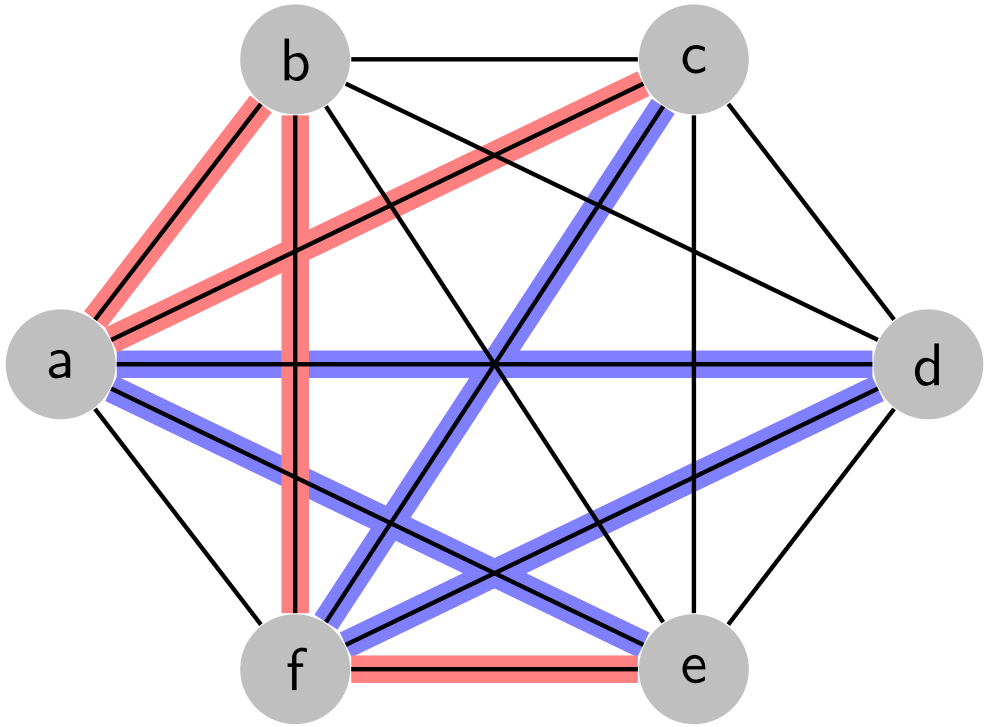
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b$$

Algoritmos de aproximação para o TSPM



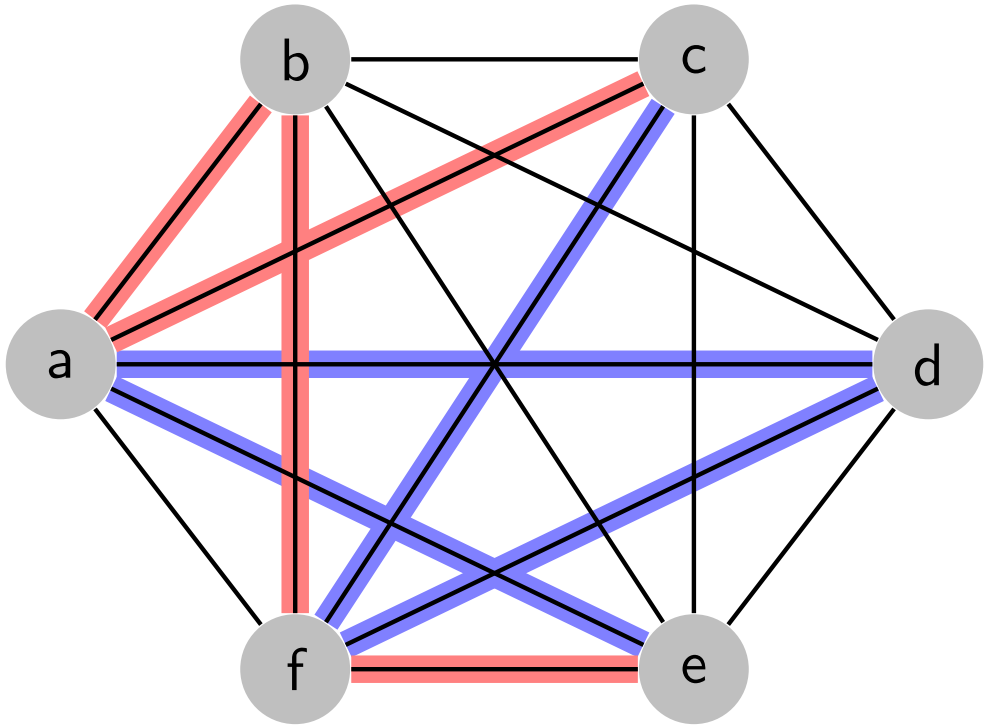
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f$$

Algoritmos de aproximação para o TSPM



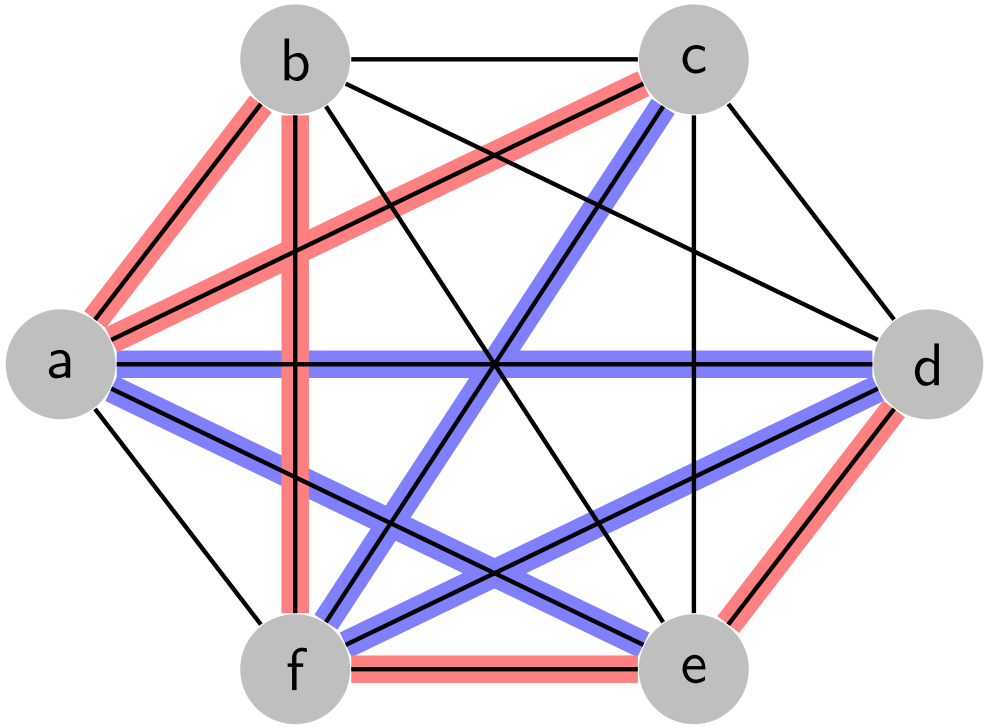
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e$$

Algoritmos de aproximação para o TSPM



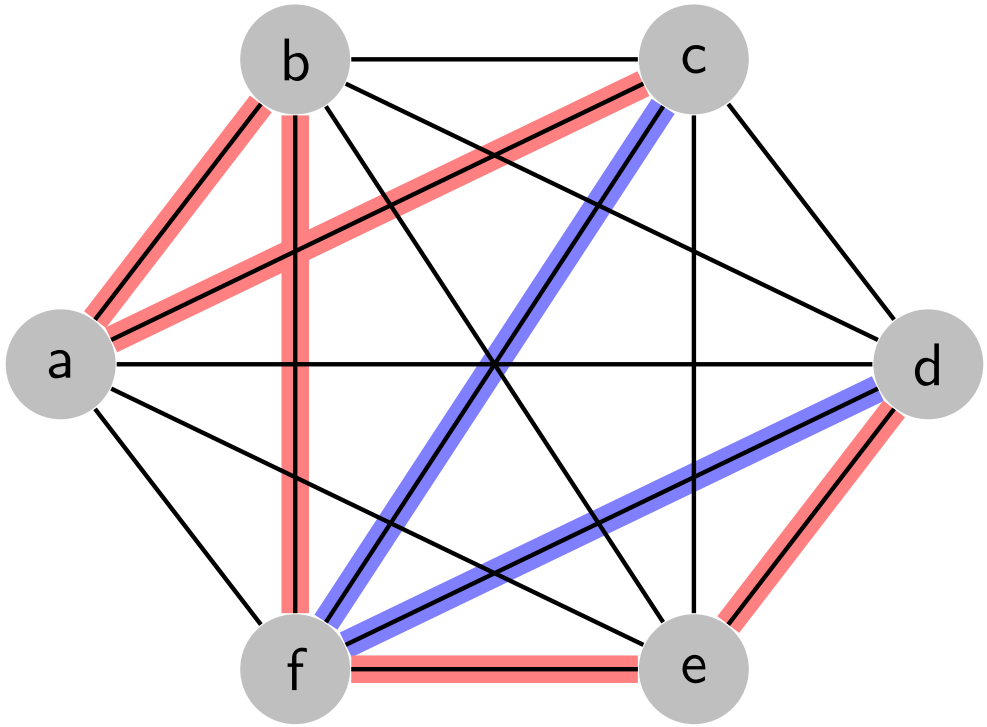
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e$$

Algoritmos de aproximação para o TSPM



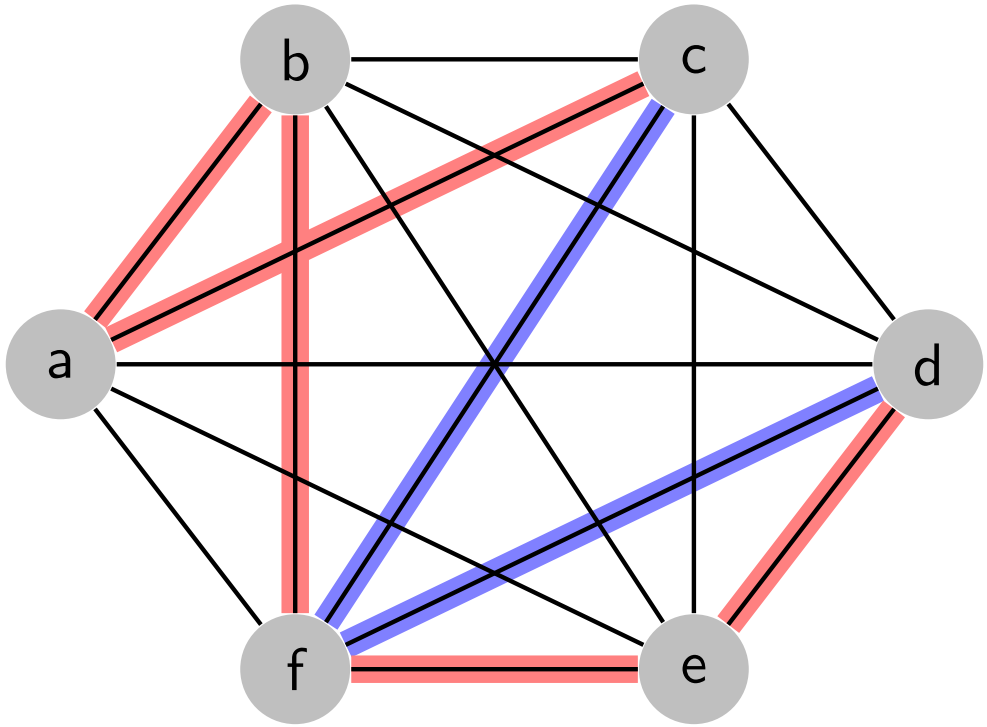
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e, d$$

Algoritmos de aproximação para o TSPM



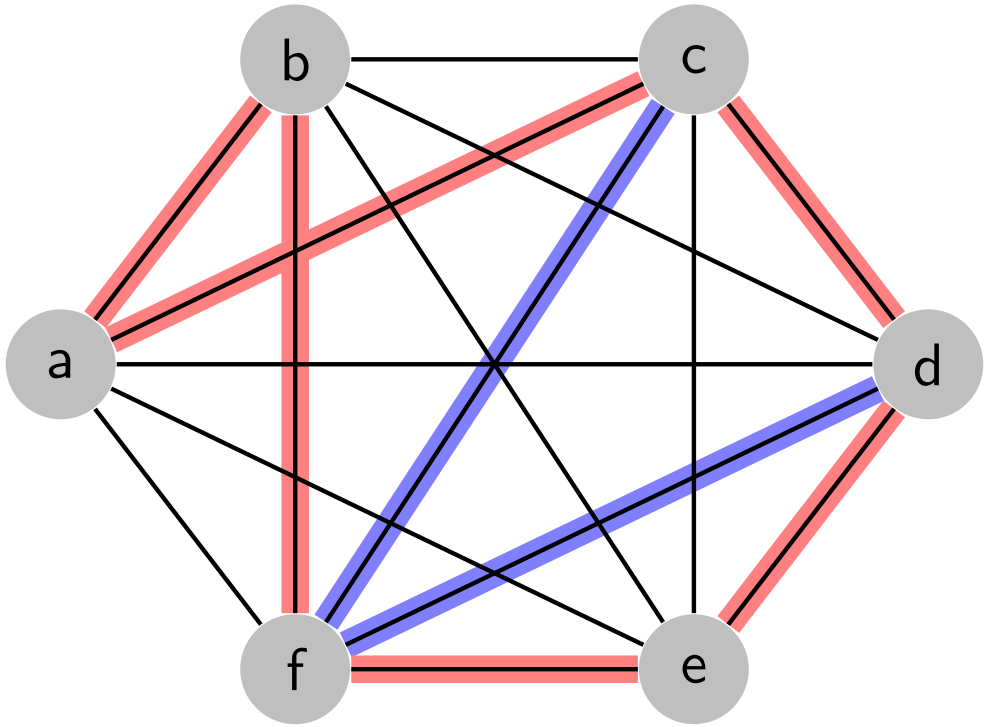
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e, d$$

Algoritmos de aproximação para o TSPM



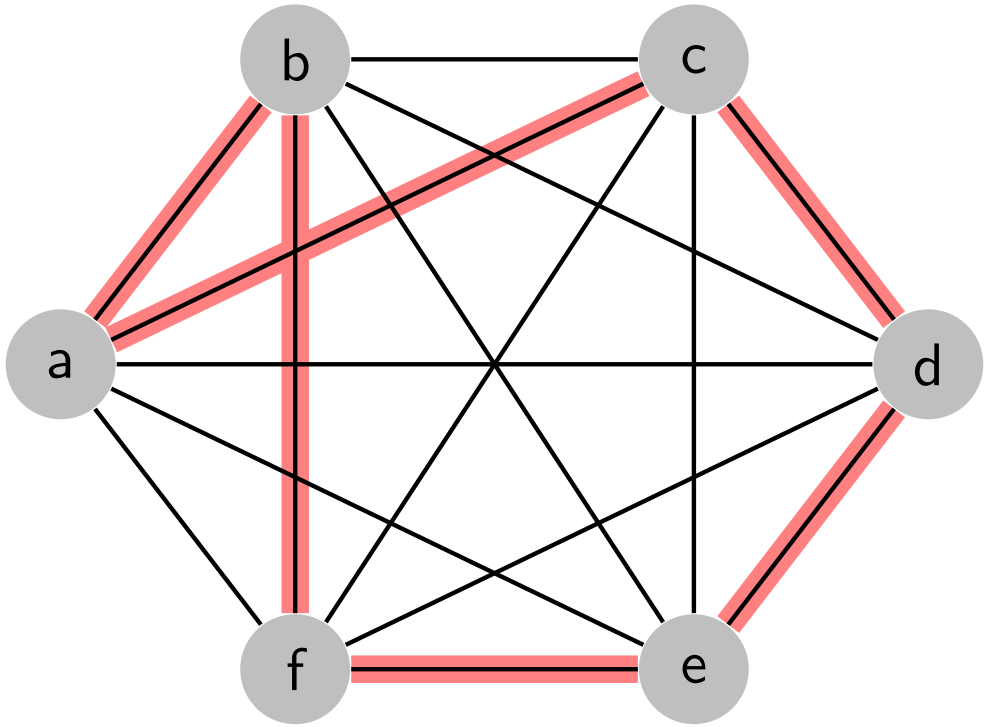
$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e, d$$

Algoritmos de aproximação para o TSPM



$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e, d, c)$$

Algoritmos de aproximação para o TSPM



$$P = (c, a, b, f, e, a, d, f, c) \Rightarrow C = (c, a, b, f, e, d, c)$$

Algoritmos de aproximação para o TSPM

Como o grafo G é completo, a sequência $C = (w_0, w_1, ..., w_n, w_0)$ define um ciclo.

O ciclo C contém todos os vértices do grafo, pois o circuito P contém todos os vértices.

Cada par (w_j, w_{j+1}) de vértices consecutivos em C é ligado por um segmento $(v_i, v_{i+1}, ..., v_{i+p})$ em P. Graças à desigualdade triangular, o custo da aresta $w_j w_{j+1}$ não é maior que o custo do segmento.

Portanto, o custo do ciclo resultante C não é maior que o do circuito dado P. Ou seja,

$$c(C) \leq c(P).$$

O tempo gasto por ATALHO é proporcional ao número de arestas do circuito P, ou seja, ao número de arestas do grafo.

Algoritmo de Rosenkrantz, Stearns e Lewis

No algoritmo descrito a seguir, apresentado em um artigo de Rosenkrantz, Stearns e Lewis, o multigrafo T' (passo 2 da estratégia) é obtido por meio da duplicação de cada uma das arestas da árvore geradora T.

Algoritmo $\mathrm{TSPM\text{-}RSL}(G, c)$:

1 $T \leftarrow \mathrm{MST}(G, c)$;

2 $T' \leftarrow T + E_T$;

3 $P \leftarrow \textsc{Euler}(T')$;

4 $C \leftarrow \textsc{Atalho}(P)$;

5 devolva C.

Algoritmo de Rosenkrantz, Stearns e Lewis

Evidentemente, todo vértice de T' tem grau par e, portanto, T' tem um circuito euleriano.

O algoritmo EULER determina um tal circuito.

Como o conjunto de vértices de T' é V_G, o circuito euleriano P é gerador.

O ciclo C devolvido por ATALHO na linha 4 é, então, um ciclo hamiltoniano de G.

Algoritmo de Rosenkrantz, Stearns e Lewis

Teorema 1: O algoritmo TSPM-RSL é uma 2-aproximação polinomial para o TSPM.

Demonstração: Como P é um circuito euleriano em $T + E_T$, temos que $c(P) = 2c(T)$.

Como $opt(G, c) \geq c(T)$ e $c(C) \leq c(P)$,

$$c(C) \leq c(P) = 2c(T) \leq 2opt(G, c).$$

A linha 1 do algoritmo consome tempo polinomial. As demais linhas consomem tempo $O(|V_G|)$, pois o número de arestas de T' é menor que $2|V_G|$. Ou seja, o algoritmo TSPM-RSL é polinomial.

Algoritmo de Christofides

O algoritmo de Christofides acrescenta à árvore geradora um emparelhamento perfeito no subgrafo de G induzido pelos vértices que têm grau ímpar em T.

Algoritmo TSPM-CHRISTOFIDES(G, c):

1 $T \leftarrow \text{MST}(G, c)$;

2 Seja I o conjunto dos vértices de grau ímpar de T;

3 $M \leftarrow \text{EDMONDS}(G[I], c)$;

4 $T' \leftarrow T + M$;

5 $P \leftarrow \text{EULER}(T')$;

6 $C \leftarrow \text{ATALHO}(P)$;

7 devolva C.

Algoritmo de Christofides

Como M é um emparelhamento perfeito em $G[I]$, todo vértice de $T + M$ tem grau par e, portanto, o multigrafo T' na linha 4 tem um ciclo euleriano.

O ciclo é gerador pois T é geradora.

Na linha 6 do algoritmo, C é um ciclo hamiltoniano de G.

Algoritmo de Christofides

Teorema 2: O algoritmo TSPM-CHRISTOFIDES é uma $\frac{3}{2}$-aproximação polinomial para o TSPM.

Demonstração: Precisamos mostrar que C tem custo no máximo $\frac{3}{2} opt(G, c)$.

Temos que $c(C) \leq c(P)$. Além disso,

$$c(P) = c(T') = c(T) + c(M).$$

Como $opt(G, c) \geq c(T)$, temos que

$$c(C) \leq c(T) + c(M) \leq opt(G, c) + c(M).$$

Algoritmo de Christofides

Precisamos mostrar agora que

$$c(M) \leq \frac{1}{2} opt(G, c) \Rightarrow opt(G, c) \geq 2c(M).$$

Seja C^* uma solução ótima para o TSPM.

Note que, como I é o conjunto de vértices de grau ímpar de T, $|I|$ é par.

Sejam $u_1, u_2, ..., u_{2k}$ os vértices de I na ordem em que aparecem em C^*.

Algoritmo de Christofides

Como G é completo, a sequência $D := (u_1, u_2, ..., u_{2k}, u_1)$ é um ciclo em $G[I]$.

Em outras palavras, D pode ser obtido de C^* pela substituição de cada segmento de C^* que liga u_i a u_{i+1} pela aresta $u_i u_{i+1}$ de G.

A desigualdade triangular garante que $c(D) \leq c(C^*)$.

Algoritmo de Christofides

Além disso, como D tem comprimento par, E_D é a união de dois emparelhamentos perfeitos em $G[I]$ mutuamente disjuntos, digamos M' e M''.

Como M é um emparelhamento perfeito de custo mínimo,

$$2c(M) \leq c(M') + c(M'').$$

Logo,

$$2c(M) \leq c(M') + c(M'') = c(D) \leq c(C^*) = opt(G, c),$$

como gostaríamos.

Algoritmo de Christofides

A linha 3 consome tempo $O(|V_G|^3)$, enquanto que as demais linhas consomem tempo polinomial no número de vértices e arestas de G.

Portanto, o algoritmo TSPM-CHRISTOFIDES é polinomial.

Proposto em 1976, TSPM-CHRISTOFIDES é ainda o melhor algoritmo de aproximação conhecido para o TSPM.

O algoritmo TSPM-RSL pode ser uma boa alternativa, já que ele consome menos tempo que o TSPM-CHRISTOFIDES e é bem mais simples, pois não envolve a determinação de um emparelhamento perfeito de custo mínimo.