Engenharia de Algoritmos e Otimização Combinatória

Mário César San Felice

(com materiais da Profa. Carla Negri Lintzmayer do CMCC-UFABC
e do Prof. Flávio Keidi Miyazawa do IC-Unicamp)

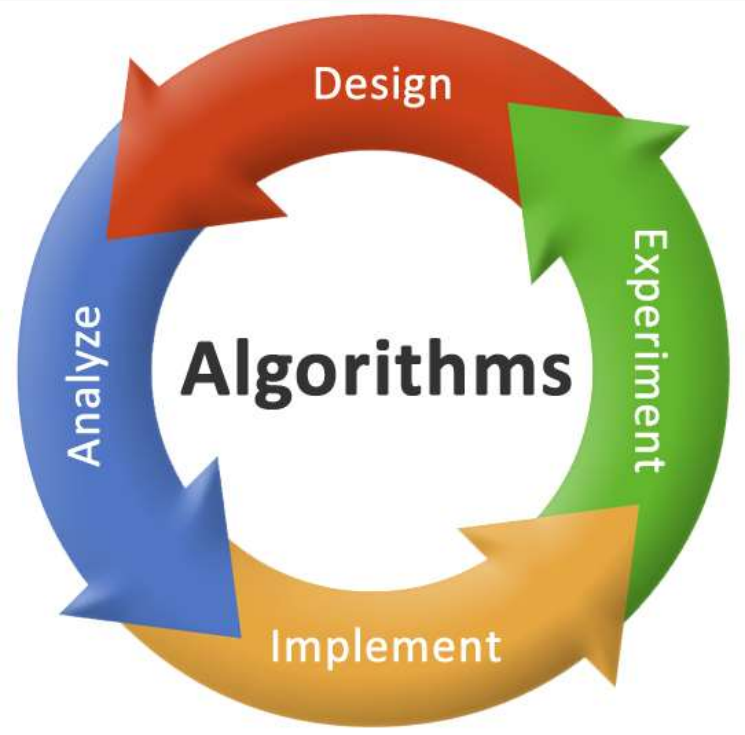
12 de Junho de 2018

Universidade Federal de São Carlos
Departamento de Computação

Engenharia de Algoritmos

Engenharia de Algoritmos

Uma abordagem focada na pesquisa e desenvolvimento de algoritmos eficientes baseada no ciclo



Engenharia de Algoritmos

Expandindo cada etapa do ciclo, temos

Projeto: compreender o problema e projetar um algoritmo para resolvê-lo.
Foco na simplicidade do algoritmo, que muitas vezes aumenta a confiabilidade e aumenta eficiência.

Análise: analisar matematicamente tanto a eficiência quanto a qualidade das soluções do algoritmo.
Evitar descartar algoritmos interessantes, como os que usam aleatoriedade, apenas em função da dificuldade em analisá-los.

Engenharia de Algoritmos

Expandindo cada etapa do ciclo, temos

Implementação: implementar o algoritmo usando estruturas de dados e rotinas auxiliares adequadas.
Considerar detalhes de hardware como localidade temporal e espacial.

Experimentos: testar o algoritmo com diversas instâncias, para verificar empiricamente seu desempenho em qualidade e tempo.
Comparar os resultados com aqueles de outros algoritmos para o mesmo problema.

Engenharia de Algoritmos

Destacamos que informações obtidas em qualquer fase é usada nas demais, pois novos conhecimentos sobre o problema e o algoritmo podem originar melhorias.

A abordagem de engenharia de algoritmos é particularmente útil para produzir bibliotecas e arcabouços (frameworks) confiáveis para facilitar a adoção pela comunidade de melhores soluções para problemas difíceis.

Otimização Combinatória

Problemas de Otimização

Um problema de otimização é caracterizado por

entrada: definição dos dados que são recebidos.

soluções viáveis: atribuição de valores às variáveis do problema que satisfazem certas restrições do mesmo.

função objetivo: alguma função que atribui valores às soluções viáveis.

objetivo: encontrar uma solução que possui o melhor valor dentre todas as soluções viáveis.

Problemas de Otimização Combinatória

Nos problemas de otimização, melhor pode ser

maior: buscamos uma solução que maximiza o "lucro" calculado pela função objetivo (problema de maximização).

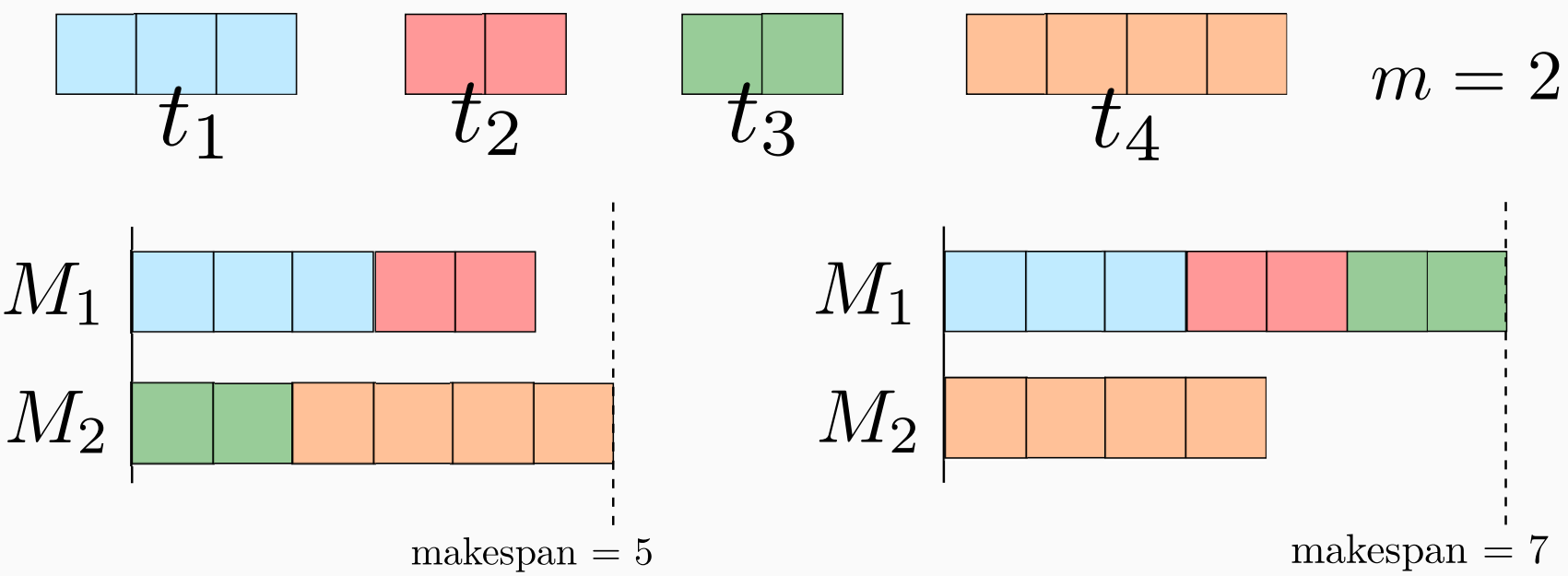
menor: buscamos uma solução que minimiza o "custo" calculado pela função objetivo (problema de minimização).

Nos problemas de otimização combinatória as variáveis pertencem a um espaço discreto.

Além disso, o conjunto de soluções viáveis é finito (enumerável), mas em geral muito grande.

Exemplo: escalonamento

Dadas n tarefas, cada uma com uma duração, alocá-las em m máquinas minimizando a maior soma de tempos.

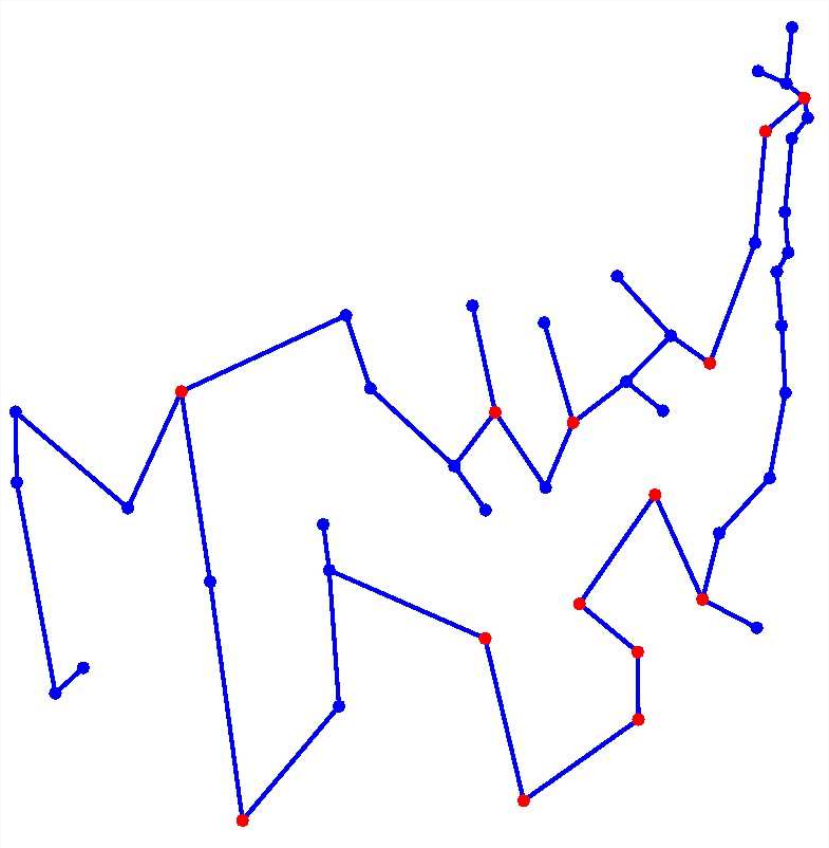


Mais Exemplos de Problemas

Projeto de redes

Suponha que você está projetando uma rede de servidores:

- alguns computadores importantes devem sempre estar conectados
- outros podem servir de nó intermediário
- temos um custo para conectar diretamente dois computadores



Projeto de redes

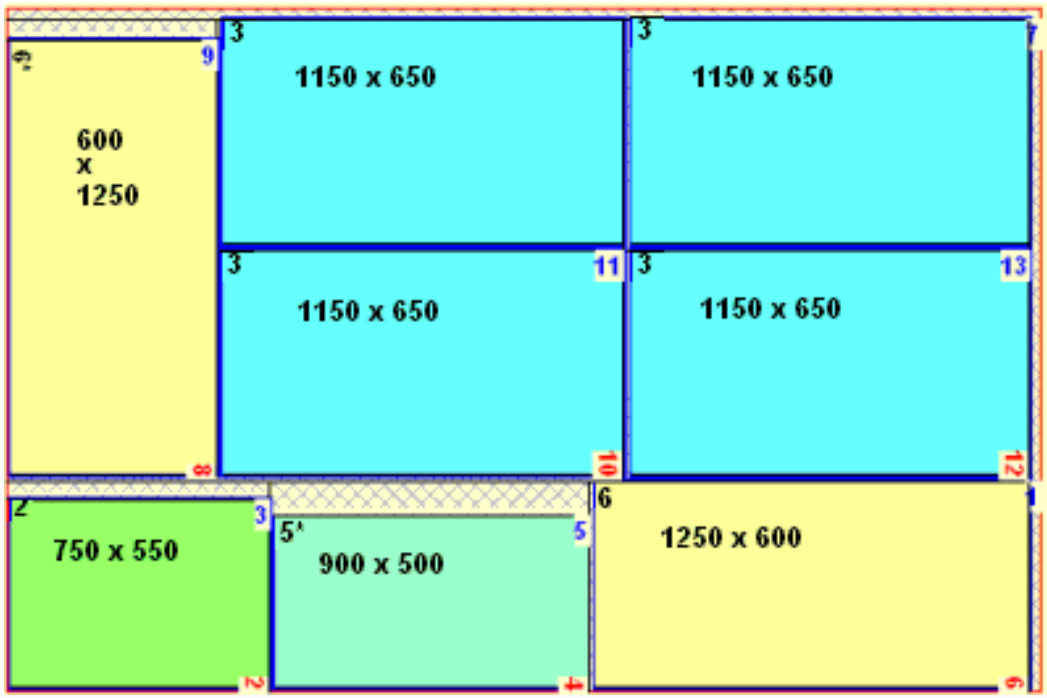
Árvore de Steiner

Entrada: grafo $G = (V, E)$ com $V = R \cup S$, onde R são vértices requeridos e S são vértices de Steiner, e função w de peso nas arestas.
Soluções viáveis: árvores que conectam todos os vértices em R que podem ter ou não vértices em S.
Função objetivo: soma dos pesos das arestas da árvore.
Objetivo: encontrar solução de custo mínimo.

Problemas de corte

Suponha que você está construindo um prédio:

- as janelas e divisórias de vidro têm quantidades e tamanhos diferentes
- você especifica as dimensões de cada pedaço
- a vidraçaria deve cortar os pedaços em placas de vidro de tamanhos fixos



Problemas de corte

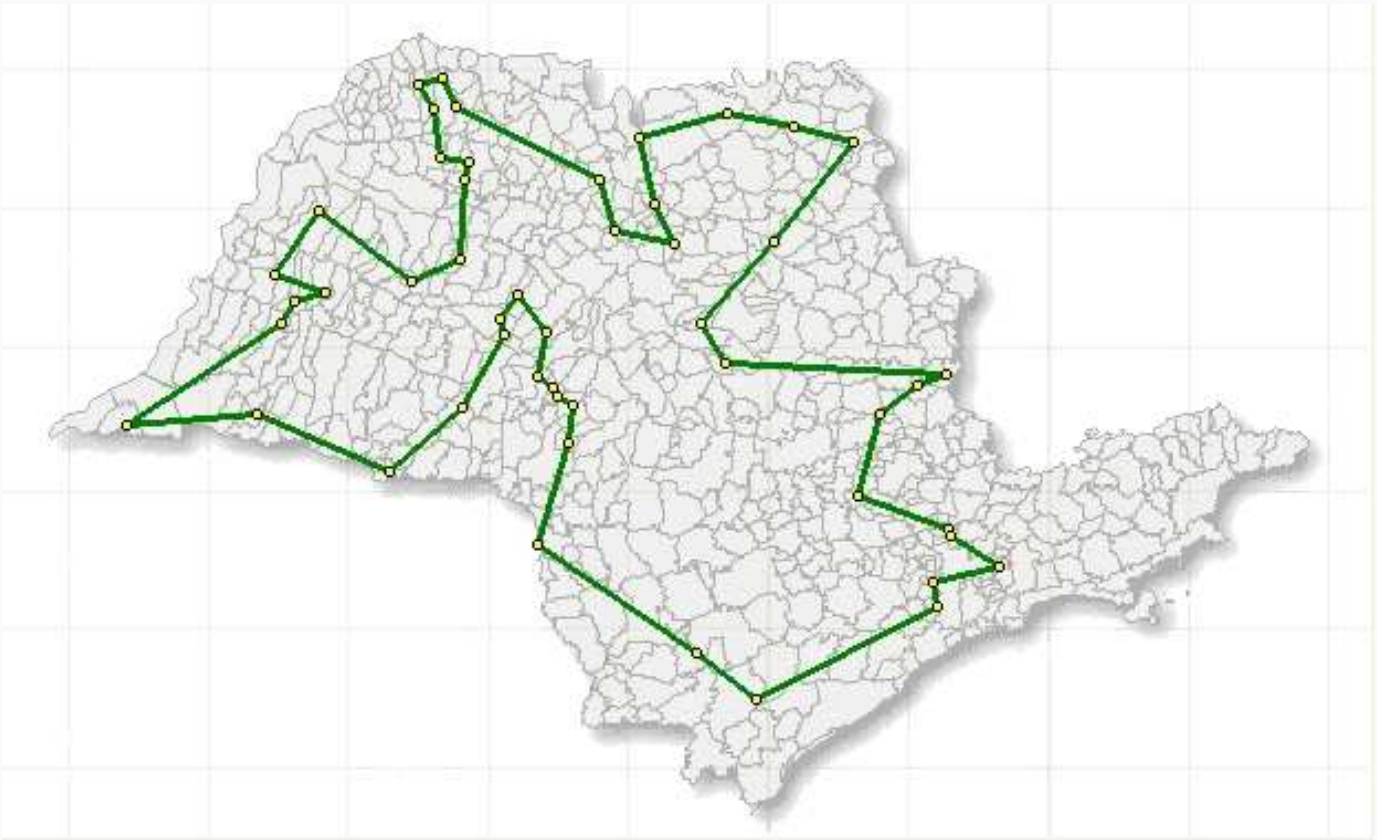
Bin Packing

Entrada: conjunto $L = \{1, 2, \ldots, n\}$ de itens retangulares, tendo cada item i largura w_i e altura h_i, e dois números W e H que indicam, respectivamente, a largura e a altura de um recipiente retangular.
Soluções viáveis: partição L_1, L_2, $\ldots$, L_q de L tal que os itens em L_k cabem completamente em uma bin $W \times H$ e não se sobrepõem.
Função objetivo: quantidade de bins utilizadas.
Objetivo: encontrar solução de custo mínimo.

Problemas de percursos

Suponha que você está planejando uma viagem pelo estado:

- você tem uma lista de cidades que deseja visitar
- você vai sair de São Paulo e voltar para São Paulo
- você não quer visitar a mesma cidade duas vezes



Problemas de percursos

Caixeiro Viajante (TSP)

Entrada: Grafo $G = (V, E)$ conexo e função w de peso nas arestas.

Soluções viáveis: ciclos hamiltonianos de G.

Função objetivo: soma dos pesos das arestas do ciclo.

Objetivo: encontrar solução de custo mínimo.

Diagramação de propagandas

Suponha que você está projetando um site:

- você vai manter uma região retangular na lateral da tela, de largura fixa, para mostrar propagandas
- as propagandas têm mesma largura, porém alturas distintas
- cada propaganda fornece um lucro diferente se for apresentada

Lorem ipsum per vulputate malesuada ullamcorper viverra sollicitudin risus sapien, torquent turpis metus ultricies rutrum pretium sollicitudin per aliquam, elit dapibus euismod quis mollis odio platea morbi. nam molestie tortor accumsan eros viverra nulla, nibh sodales maecenas sapien felis, mattis libero id fusce porta. consequat nisi semper neque nam tincidunt congue dolor elementum malesuada, netus orci nulla vulputate aenean ante iaculis imperdiet conubia lorem, hendrerit habitasse aliquam quis scelerisque pharetra mauris aliquet. erat a tincidunt vitae tristique hendrerit lacus etiam elit habitasse, pharetra elementum cubilia sapien facilisis egestas curabitur semper, placerat ut rhoncus metus donec inceptos potenti sed. Donec conubia phasellus cubilia luctus curae dictum est quisque at proin nam justo, tempus condimentum morbi mi rutrum pellentesque non pharetra habitant turpis. feugiat sem faucibus donec fringilla maecenas molestie ultricies aenean, imperdiet lacus iaculis vel mi scelerisque venenatis vivamus, porttitor integer etiam molestie porttitor condimentum rhoncus. facilisis placerat nam hendrerit convallis curabitur aenean aliquam aenean, at

Diagramação de propagandas

Problema da Mochila

Entrada: conjunto de n itens $\{1, 2, \ldots, n\}$, cada um com um peso w_i e valor v_i, e um inteiro W.

Soluções viáveis: conjuntos de itens $S \subseteq \{1, 2, \ldots, n\}$ tais que $\sum_{i \in S} w_i \leq W$.

Função objetivo: $\sum_{i \in S} v_i$.

Objetivo: encontrar solução de valor máximo.

Dificuldades

Técnica de solução: força-bruta

Algoritmos de força-bruta enumeram todas as soluções guardando a melhor possível.

Eles de fato resolvem o problema...

Porém usam muito esforço computacional para encontrar uma solução, sem explorar as estruturas combinatórias do problema.

Técnica de solução: força-bruta

Como exemplos, considere o tamanho do espaço de busca dos seguintes problemas.

TSP: qualquer sequência dos n vértices é candidato à solução

- gere as $n!$ sequências de vértices
- cada uma delas é um ciclo hamiltoniano
- calcule seu custo e compare com o melhor já encontrado

Mochila: qualquer subconjunto dos n elementos é candidato à solução

- gere os 2^n possíveis subconjuntos
- para cada um, teste se os itens cabem na mochila
- caso caibam, calcule seu custo e compare com o melhor já encontrado

Comparando tempos

Suponha um computador que executa 4 bilhões de instruções por segundo, i.e., 4GHz.

Seja n o tamanho da entrada e $f(n)$ a quantidade de instruções do algoritmo.

$f(n)$	$n = 10$	$n = 25$	$n = 50$	$n = 100$
n	0.000000003 seg	0.000000006 seg	0.000000012 seg	0.000000025 seg
n^2	0.000000025 seg	0.000000156 seg	0.000000625 seg	0.0000025 seg
n^3	0.00000025 seg	0.000003906 seg	0.00003125 seg	0.00025 seg
n^5	0.000025 seg	0.002441406 seg	0.078125 seg	2.5 seg
2^n	0.000000256 seg	0.008388608 seg	3.3 dias	10^{13} anos
$n!$	0.0009072 seg	122965 milênios	-	-

Complexidade

Algoritmo eficiente: tem complexidade de tempo polinomial no tamanho n da entrada ($O(n^k)$, para alguma constante k).

Problemas de decisão: problemas cuja resposta é SIM ou NÃO.

Classe P: problemas de decisão que podem ser resolvidos por algoritmos eficientes.

Conjectura de Edmonds de 1965: não existe algoritmo que resolve o TSP em tempo polinomial.

Complexidade

Redução: um problema A é redutível a B se podemos usar um algoritmo que resolve B para resolver A (A é tão difícil quanto B).

Classe NP: problemas de decisão cuja solução SIM pode ser verificada em tempo polinomial.

Classe NP-Completo: problemas Q tais que $Q \in$ NP e todo problema em NP é redutível a Q.

Questão P versus NP: se houver um algoritmo que resolve qualquer Q NP-Completo em tempo polinomial, todos os problemas em NP são resolvidos também, i.e., estão em P.

Classe NP-Difícil: problemas Q tais que todo problema em NP é redutível a Q.

Problemas em P e em NP-Difícil/NP-Completo

Problemas em P	**Problemas NP-Completos ou NP-Difíceis**
Caminho mínimo	Caminho mais longo
Árvore geradora mínima	Árvore de Steiner
2-coloração	3-coloração
Circuito euleriano	Circuito Hamiltoniano
Mochila Fracionária	Mochila

Abordagens

Se P $\neq$ NP, não podemos ter algoritmos para problemas NP-Difíceis que, simultaneamente,

1. encontrem soluções ótimas
2. em tempo polinomial
3. para qualquer entrada.

Então basta não exigir essas três coisas!

Algumas abordagens possíveis são

- algoritmos exatos,
- algoritmos de aproximação,
- heurísticas,
- algoritmos parametrizados.

Abordagem por algoritmos exatos

Algoritmos exatos

Procuram pela solução ótima considerando as estruturas combinatórias dos problemas.

Consideram técnicas como

- algoritmos gulosos,
- programação dinâmica,
- *branch and bound*,
- programação linear / programação linear inteira,
- programação por restrições.

Branch and Bound

Combina enumeração (branch) com poda de soluções não promissoras (bound).

A estratégia é percorrer uma árvore de enumeração e sempre que chegarmos a um ramo que não é promissor ou é inviável, podá-lo sem percorrê-lo.

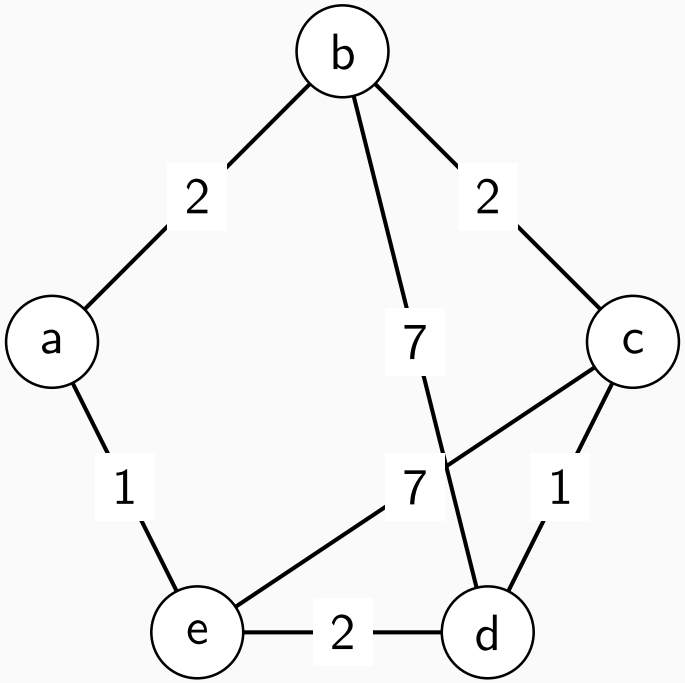
É importante pensar em

- como fazer a enumeração,
- como percorrer a árvore (qual ordem), e
- como podar a árvore (quais condições testar, como garantir que não alcançaríamos soluções melhores)

Branch and Bound para o TSP

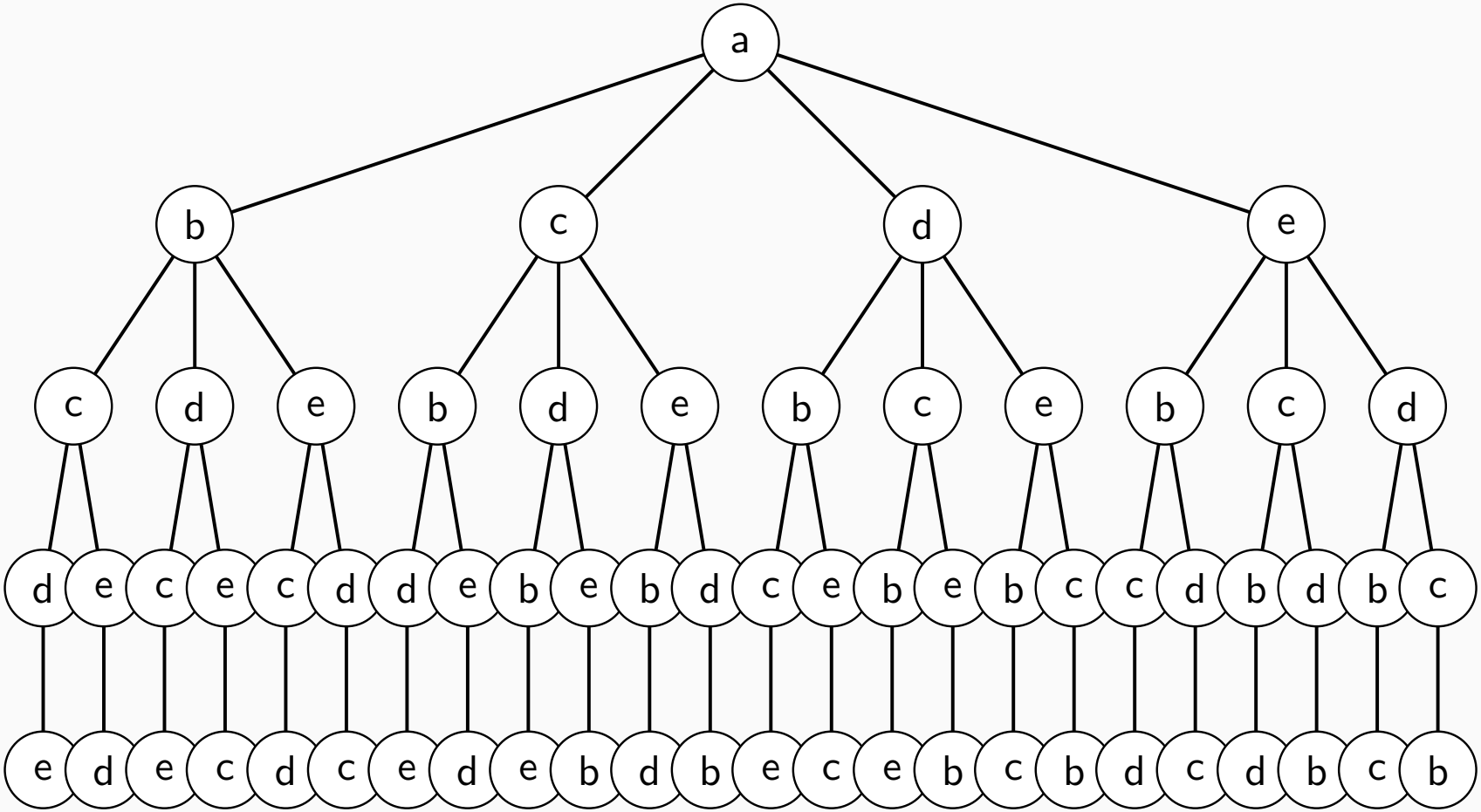
Ideia: vamos enumerar todas as sequências de vértices possíveis sendo que cada nó da árvore de enumeração vai representar um vértice do grafo e cada ramificação na árvore vai representar uma aresta do grafo que foi percorrida.

Considere o seguinte grafo:



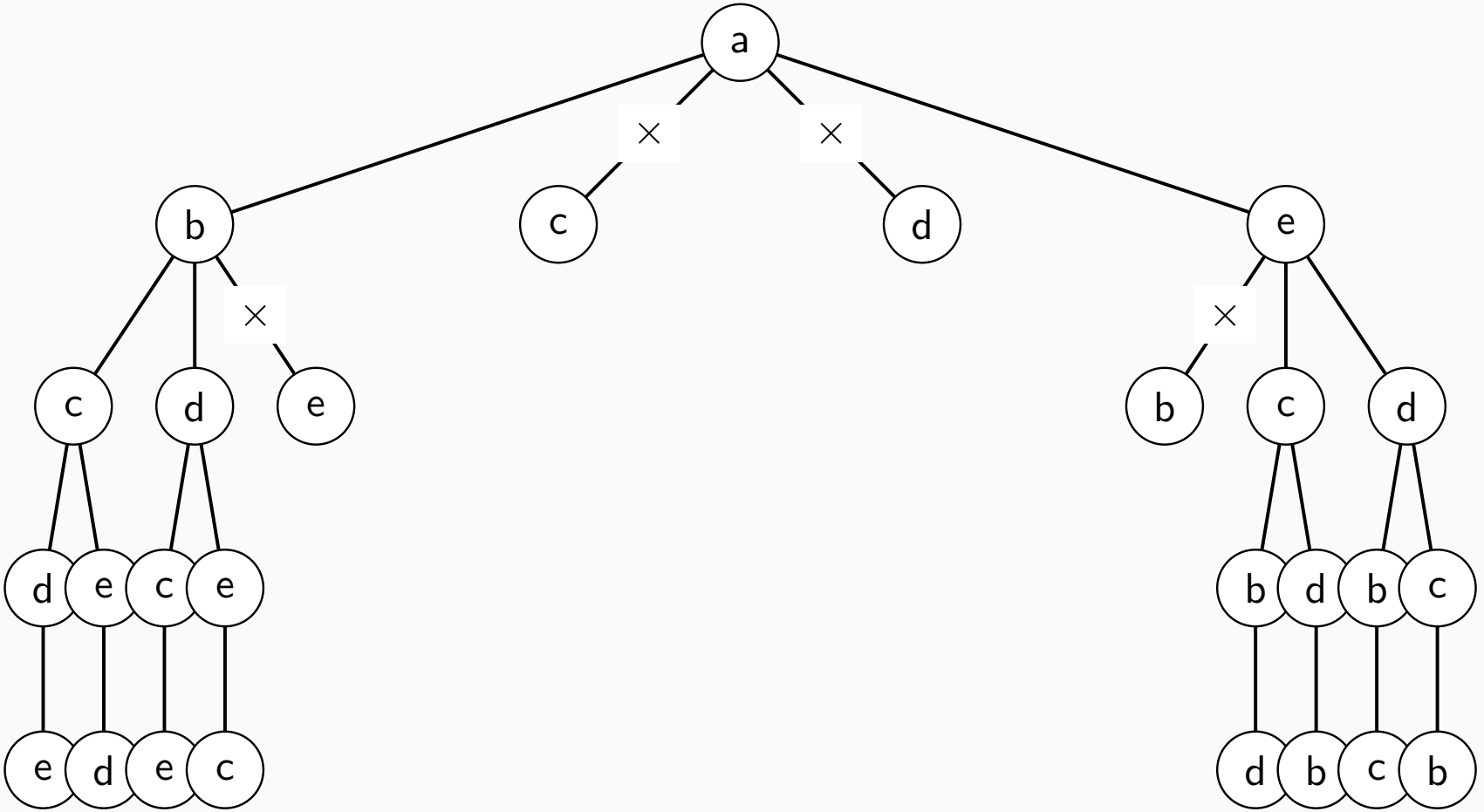
Branch and Bound para o TSP

Árvore com todas as sequências possíveis de vértices:



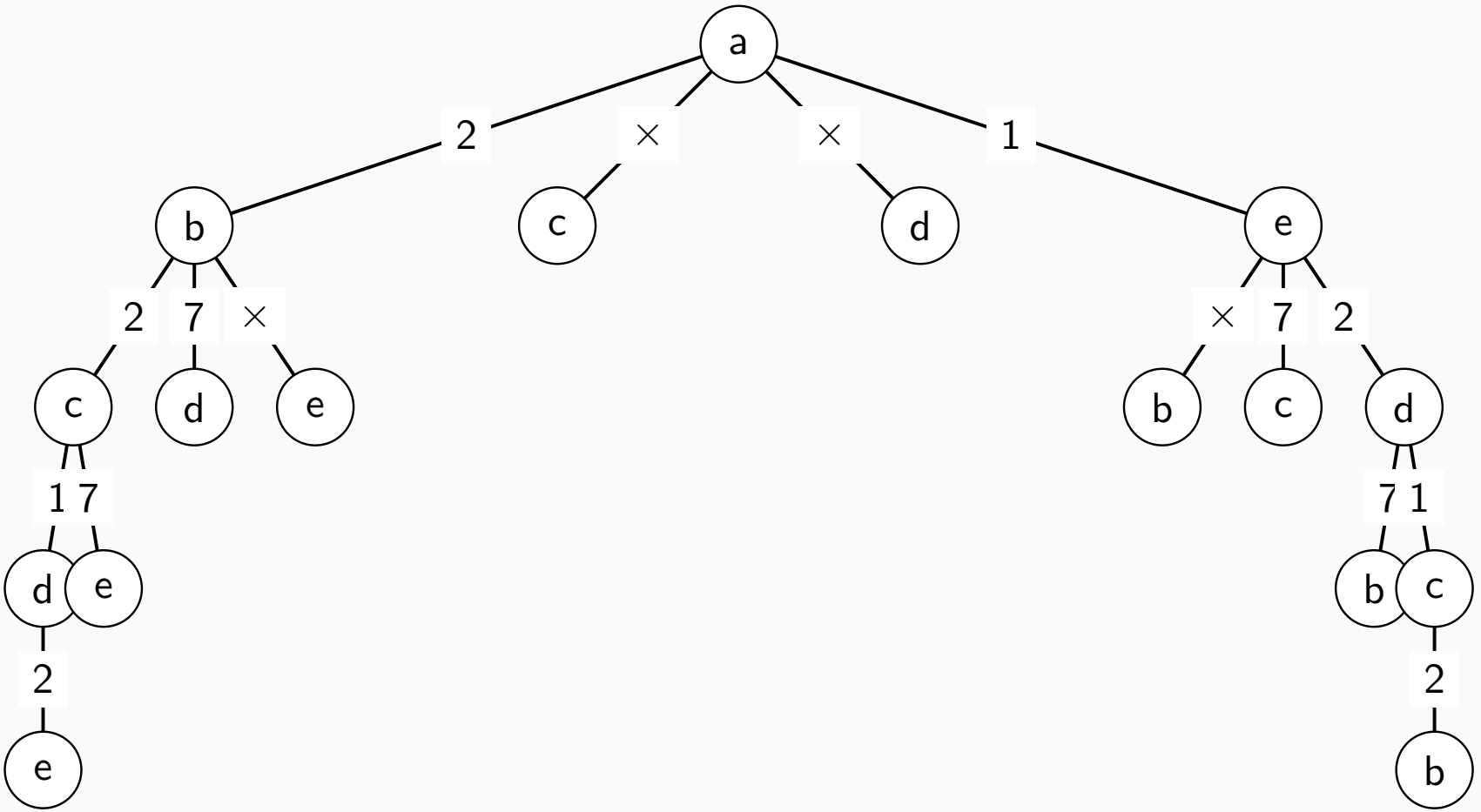
Branch and Bound para o TSP

Árvore podada por soluções inviáveis:



Branch and Bound para o TSP

Árvore podada por soluções não promissoras:



Programação Linear

Modelo matemático que descreve problemas de otimização.

Um programa linear possui um conjunto de variáveis de decisão, um conjunto de restrições e uma função objetivo.

Cada restrição corresponde a uma equação linear sobre as variáveis de decisão.

Dizemos que uma atribuição de valores para as variáveis de decisão é uma solução viável se todas as restrições são satisfeitas.

A forma padrão de um programa linear para um problema de minimização que tem n variáveis e m restrições é

$$\begin{aligned} \text{minimizar} \quad & \sum_{j=1}^{n} c_j x_j & \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 & \forall j \in \{1, \ldots, n\} \end{aligned}$$

onde a_{ij}, b_i e c_j são constantes, x_j são as variáveis de decisão, $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ é uma restrição e $\sum_{j=1}^{n} c_j x_j$ é a função objetivo.

Formulação PLI da Mochila

Sejam x_i variáveis binárias que indicam se o item i foi escolhido ou não.

$$\begin{aligned} \text{maximizar} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\} \qquad \forall i \in \{1,\ldots,n\} \end{aligned}$$

Formulação PLI do TSP

Sejam x_e variáveis binárias que indicam se a aresta e pertence ao ciclo da solução ou não.

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e \in E} w(e) x_e & \\
\text{sujeito a} \quad & \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \emptyset \\
& x_e \in \{0, 1\} & \forall e \in E
\end{aligned}
$$

Programação Linear

Programação linear dá solução exata para muitos problemas, mas também é muito utilizada para dar soluções aproximadas.

Faz parte da abordagem por algoritmos exatos mas também da abordagem por algoritmos de aproximação.

Programas lineares podem ser resolvidos em tempo polinomial, mas programas lineares inteiros são problemas NP-difíceis.

Abordagem por algoritmos de aproximação

Algoritmos de aproximação

Geram soluções viáveis em tempo polinomial e dão garantia no custo da solução gerada com relação ao custo da solução ótima.

Seja A um algoritmo para um dado problema que executa em tempo polinomial, $A(I)$ o custo da solução gerada por A para uma entrada I e $OPT(I)$ o custo da solução ótima para uma entrada I.

Um algoritmo A é uma f-aproximação se

- $A(I) \leq f \cdot OPT(I)$ para toda instância I e o problema é de minimização, ou
- $A(I) \geq f \cdot OPT(I)$ para toda instância I e o problema é de maximização.

Algoritmo de aproximação para o Escalonamento

Escalonamento

Entrada: conjunto de tarefas $\{1, \ldots, n\}$, onde uma tarefa i tem tempo de processamento t_i, e m máquinas idênticas.

Soluções viáveis: partição das tarefas em m conjuntos M_1, M_2, $\ldots$, M_m.

Função objetivo: $\max_{1 \leq j \leq m} \sum_{i \in M_j} t_i$ (makespan).

Objetivo: encontrar solução de custo mínimo.

Algoritmo de aproximação para o Escalonamento

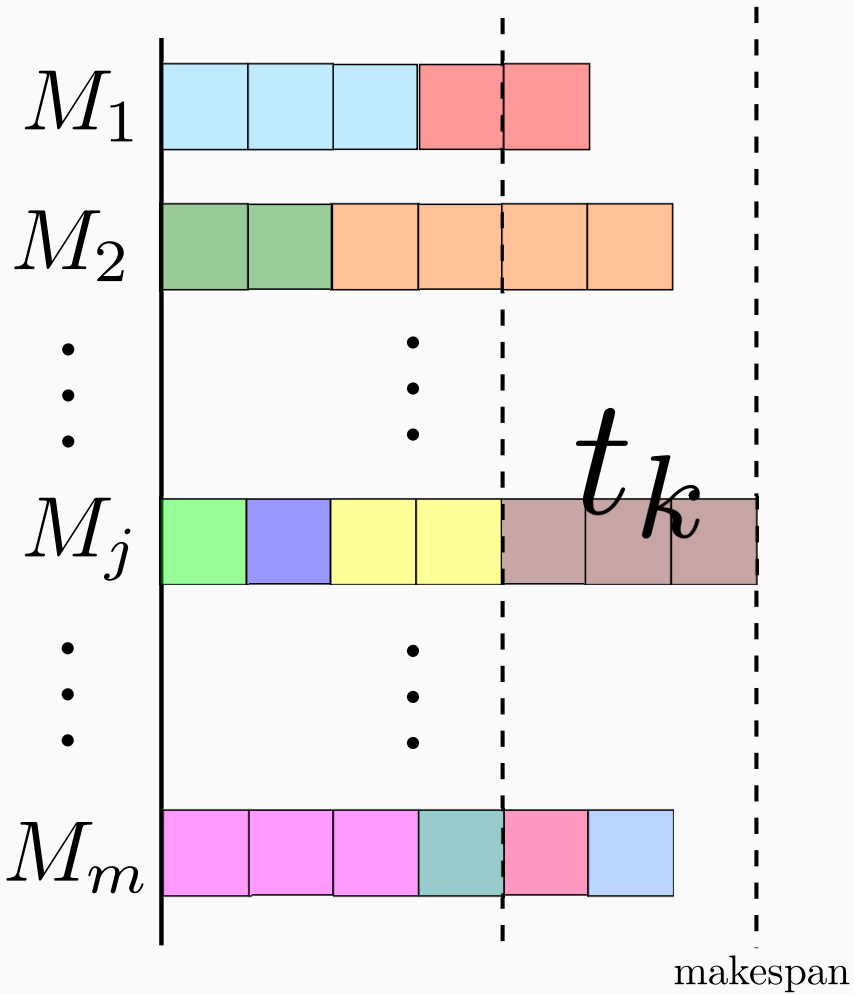
Ideia: gulosamente, alocar uma tarefa à máquina que tem o menor tempo de processamento no momento

1: **função** ApproxEscalona(n, t, m)
2: faça $M_j = \emptyset$ para todo $1 \leq j \leq m$
3: **para** $i \leftarrow 1$ até n **faça**
4: seja M_j a máquina tal que $\sum_{i \in M_j} t_i$ é mínimo
5: $M_j \leftarrow M_j \cup \{i\}$
6: **devolve** $\max_{1 \leq j \leq m} \sum_{i \in M_j} t_i$

Algoritmo de aproximação para o Escalonamento

Análise:

Seja M_j a máquina que define o makespan e seja t_k a última tarefa que foi alocada a essa máquina.



Algoritmo de aproximação para o Escalonamento

Note que todas as outras máquinas estão carregadas até o momento $(\sum_{i \in M_j} t_i) - t_k$. Então

$$\left(\left(\sum_{i \in M_j} t_i \right) - t_k \right) \cdot m \leq \sum_{i=1}^{n} t_i$$

De onde

$$\left(\sum_{i \in M_j} t_i \right) - t_k \leq \left(\sum_{i=1}^{n} t_i \right) / m \leq OPT(I)$$

Assim,

$$\begin{aligned} \textsc{AproxEscalona}(I) &= \textstyle\sum_{i \in M_j} t_i \\ &\leq OPT(I) + t_k \\ &\leq OPT(I) + t_{max} \\ &\leq OPT(I) + OPT(I) \\ &= 2OPT(I) \end{aligned}$$

De onde vemos que esse algoritmo é uma 2-aproximação.

Abordagem por heurísticas

Heurísticas

São algoritmos que geram uma solução viável em tempo polinomial mas não dão garantias de qualidade no custo da solução gerada.

Podem ser

- **construtivas**, que normalmente adotam estratégias gulosas para construir as soluções, ou
- **de busca local**, que partem de uma solução inicial e tentam modificá-la, melhorando-a, até atingir algum critério de parada.

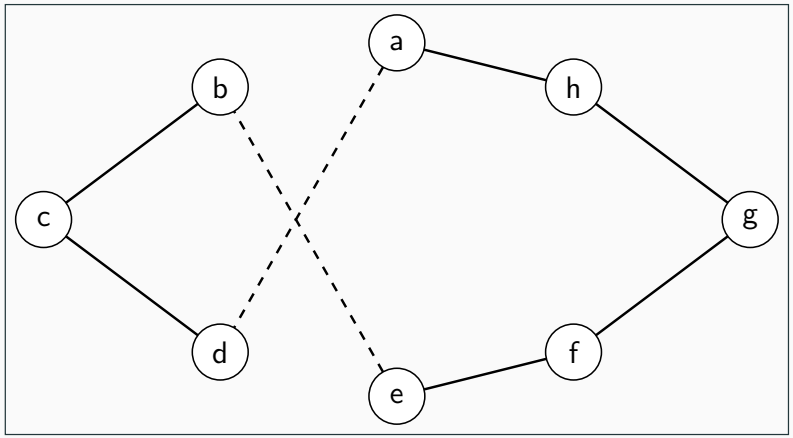
Heurística de busca local para o TSP

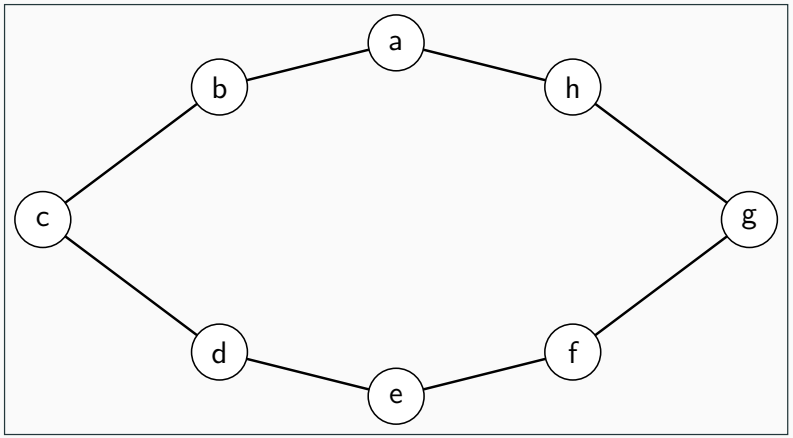
Vizinhança k-OPT de um ciclo hamiltoniano C: ciclos hamiltonianos C' que são obtidos a partir de C removendo k arestas de C e inserindo outras k arestas.

1: **função** $\text{K-OPT-TSP}(G = (V, E), w)$
2: encontra um ciclo hamiltoniano inicial C
3: **enquanto** for possível **faça**
4: procure um ciclo C' na vizinhança k-OPT de C tal que $w(C') < w(C)$
5: se não encontrou C', pare
6: $C \leftarrow C'$
7: **devolve** C

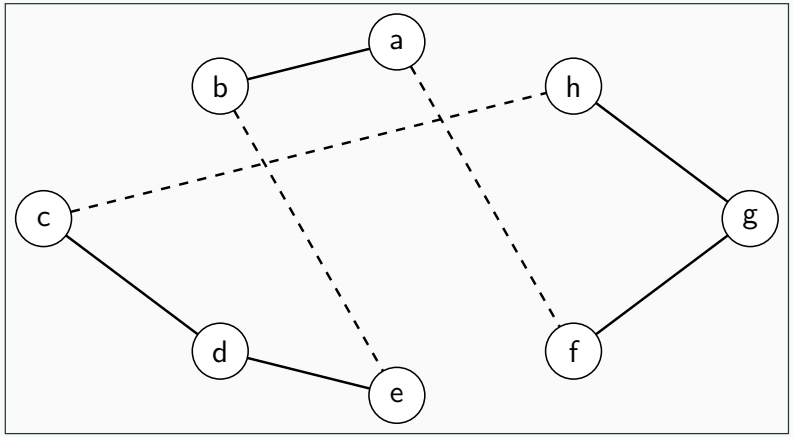
Heurística de busca local para o TSP

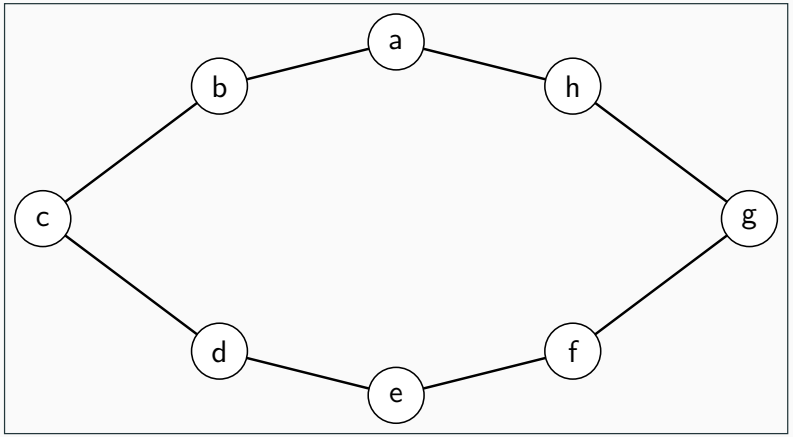
Exemplo de troca 2-OPT:





Exemplo de troca 3-OPT:





Referências i

- Problemas:
  - http://ic.unicamp.br/~fkm/problems/combopt.html
  - http://www.csc.kth.se/~viggo/wwwcompendium/wwwcompendium.html
- Abordagens exatas:
  - http://ic.unicamp.br/~fkm/lectures/otimo.pdf
  - http://ic.unicamp.br/~fkm/lectures/progint.pdf
- Abordagens heurísticas:
  - http://ic.unicamp.br/~fkm/lectures/heuristicas.pdf
- Complexidade parametrizada:
  - http://www2.ic.uff.br/~ueverton/files/complexidade_parametrizada.pdf
  - Livro Fundamentals of Parameterized Complexity, de Rod Downey e Michael R. Fellows

Referências ii

- Algoritmos de aproximação:
  - `http://ic.unicamp.br/~fkm/lectures/livro.pdf`
  - Livro Approximation Algorithms, de Vijay Vazirani (disponibilizado pelo autor)
  - Livro The Design of Approximation Algorithms, de David P. Williamson e David. B. Shmoys (disponibilizado pelos autores)
- Outros:
  - `http://ic.unicamp.br/~fkm/lectures/intro-otimizacao.pdf`
  - Livro Combinatorial Optimization, Algorithms and Complexity, de Christos H. Papadimitriou e Kenneth Steiglitz