Celso Carneiro Ribeiro

Metaheurísticas e Aplicações

2007

Metaheurísticas

- Origens
- Motivação
- Algoritmos construtivos
- Métodos de melhoria ou de busca local
- Metaheurísticas:
  - Algoritmos genéticos
  - GRASP, ...
- Aplicações:
  - Programação de tabelas de campeonatos
  - Engenharia de tráfego e roteamento na Internet

Origens

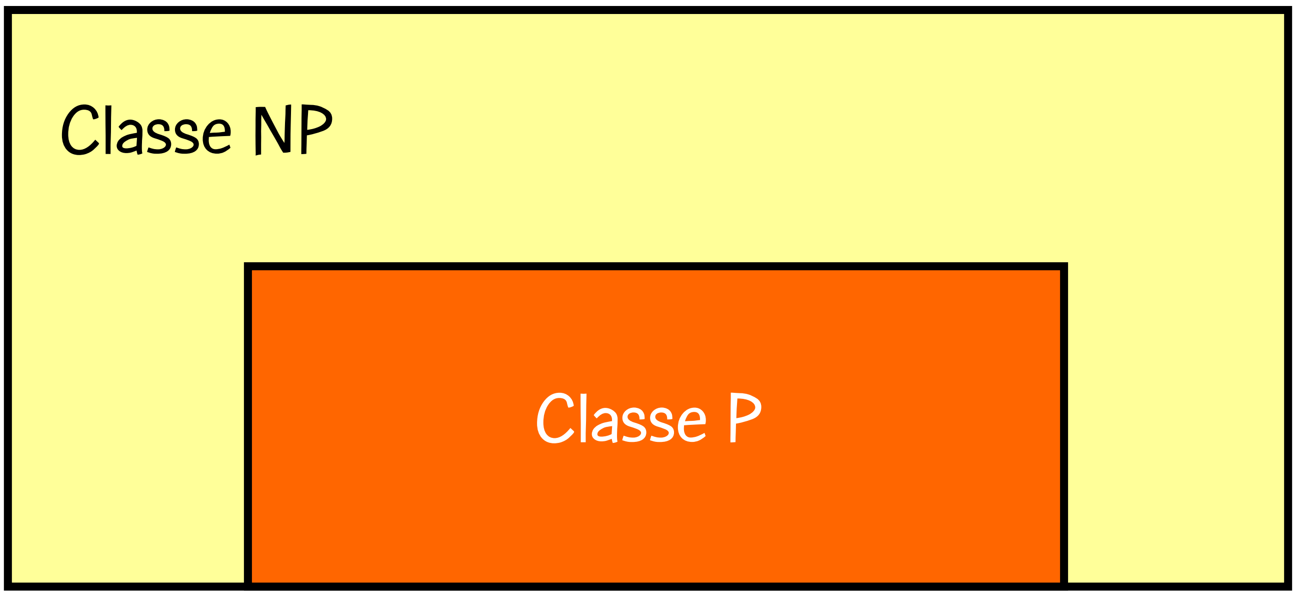
- Algoritmos aproximados para o problema do caixeiro viajante: métodos construtivos, busca local 2-opt, heurística de Lin-Kernighan (1973)
- Técnicas de inteligência artificial para problemas de busca em grafos:
  - Demonstração automática de teoremas, trajeto de robôs, problemas de otimização vistos como busca em grafos
  - Algoritmo A* ("versão" IA de B&B): Nilsson (1971)
  - Aplicações pioneiras no SE brasileiro (anos 70): modelos TANIA (expansão da transmissão) e VENUS (planejamento da expansão)

Origens

- Visão simplificada do “mundo” dos problemas de decisão:

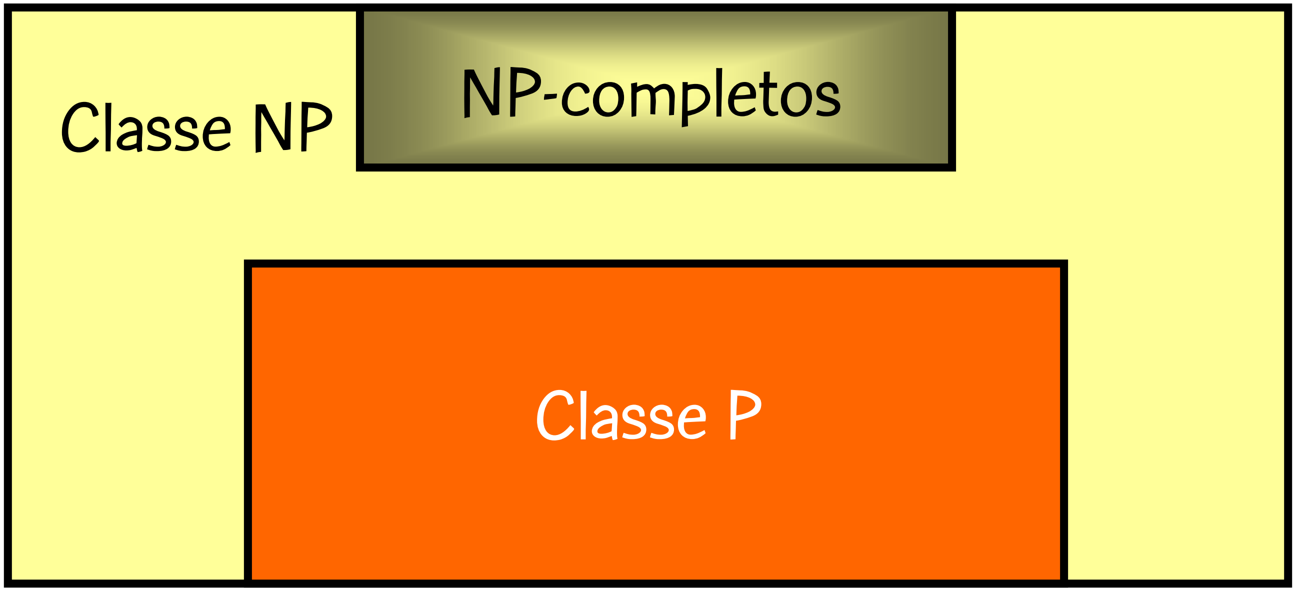
Origens

- Visão simplificada do "mundo" dos problemas de decisão:



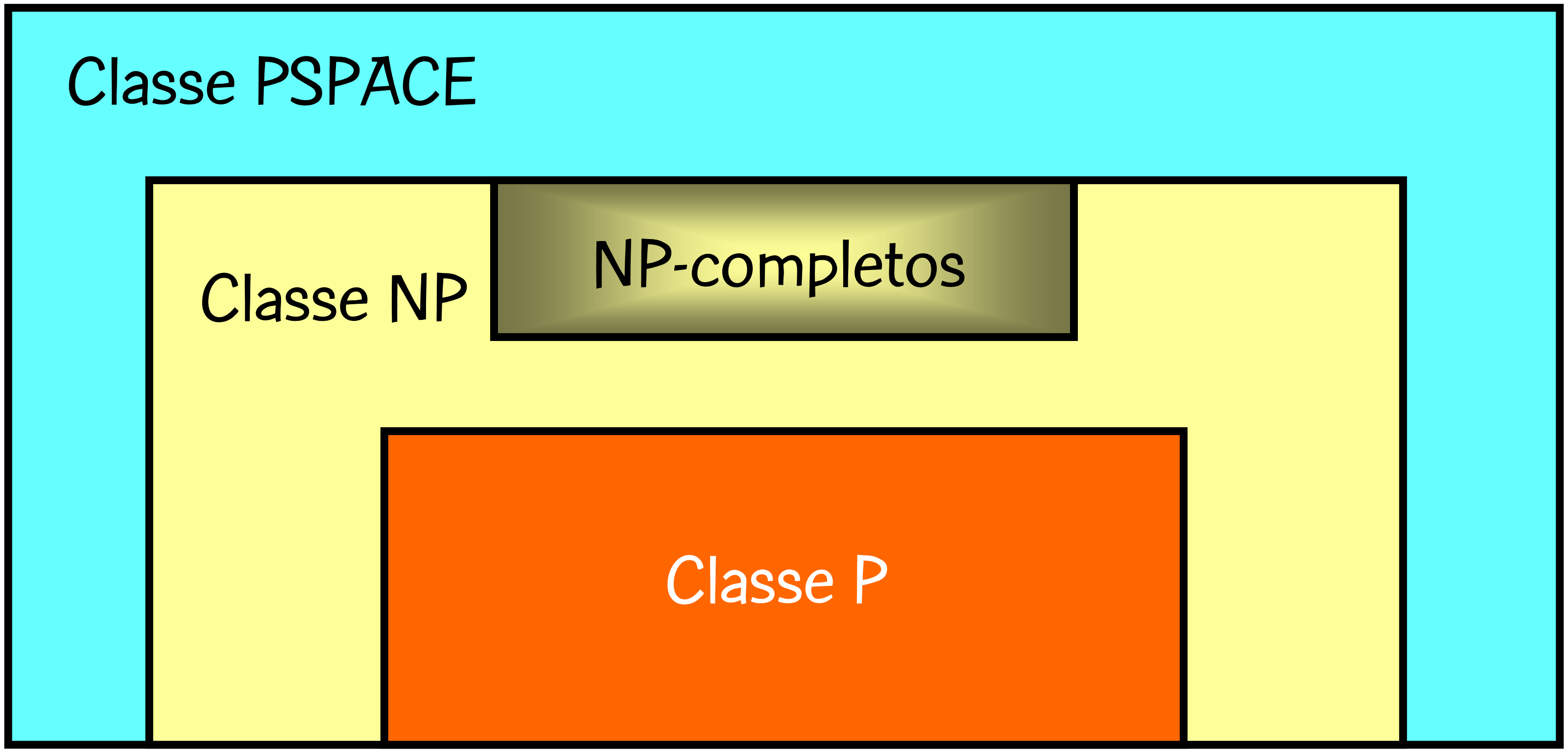
Origens

- Visão simplificada do "mundo" dos problemas de decisão:



Origens

- Visão simplificada do "mundo" dos problemas de decisão:



Origens

- Como tratar na prática os problemas NP-completos?

1. Métodos exatos de complexidade super-polinomial (branch-and-bound, cortes, enumeração, programação dinâmica): se for necessário e se o porte dos problemas assim o permitir.
2. Processamento paralelo: clusters e grids permitem acelerações significativas na prática, utilizando recursos computacionais básicos e com custo reduzido.

Origens

- Como tratar na prática os problemas NP-completos?

3. Heurísticas: qualquer método aproximado projetado com base nas propriedades estruturais ou nas características das soluções dos problemas, com complexidade reduzida em relação à dos algoritmos exatos e fornecendo, em geral, soluções viáveis de boa qualidade (sem garantia de qualidade)
   - Métodos construtivos
   - Busca local
   - Metaheurísticas

Origens

- Avanços no estudo e desenvolvimento de heurísticas buscam:
  - Resolver problemas maiores
  - Resolver problemas em tempos menores
  - Obter melhores soluções
- Heurísticas e metaheurísticas permitem resolver problemas de grande porte em tempos realistas, fornecendo sistematicamente soluções ótimas ou muito próximas da otimalidade:
  - Exemplo: problema do caixeiro viajante com milhões de cidades

Bibliografia

- N. Nilsson, "Problem-solving methods in artificial intelligence", 1971
- N. Nilsson, "Principles of artificial intelligence", 1982
- J. Pearl, "Heuristics: Intelligent search strategies for computer problem solving", 1985
- E. Lawler, J. Lenstra, A. Rinnooy Kan e D. Shmoys (eds.), "The traveling salesman problem", 1985
- C. Reeves (ed.), "Modern heuristic techniques for combinatorial problems", 1993

Bibliografia

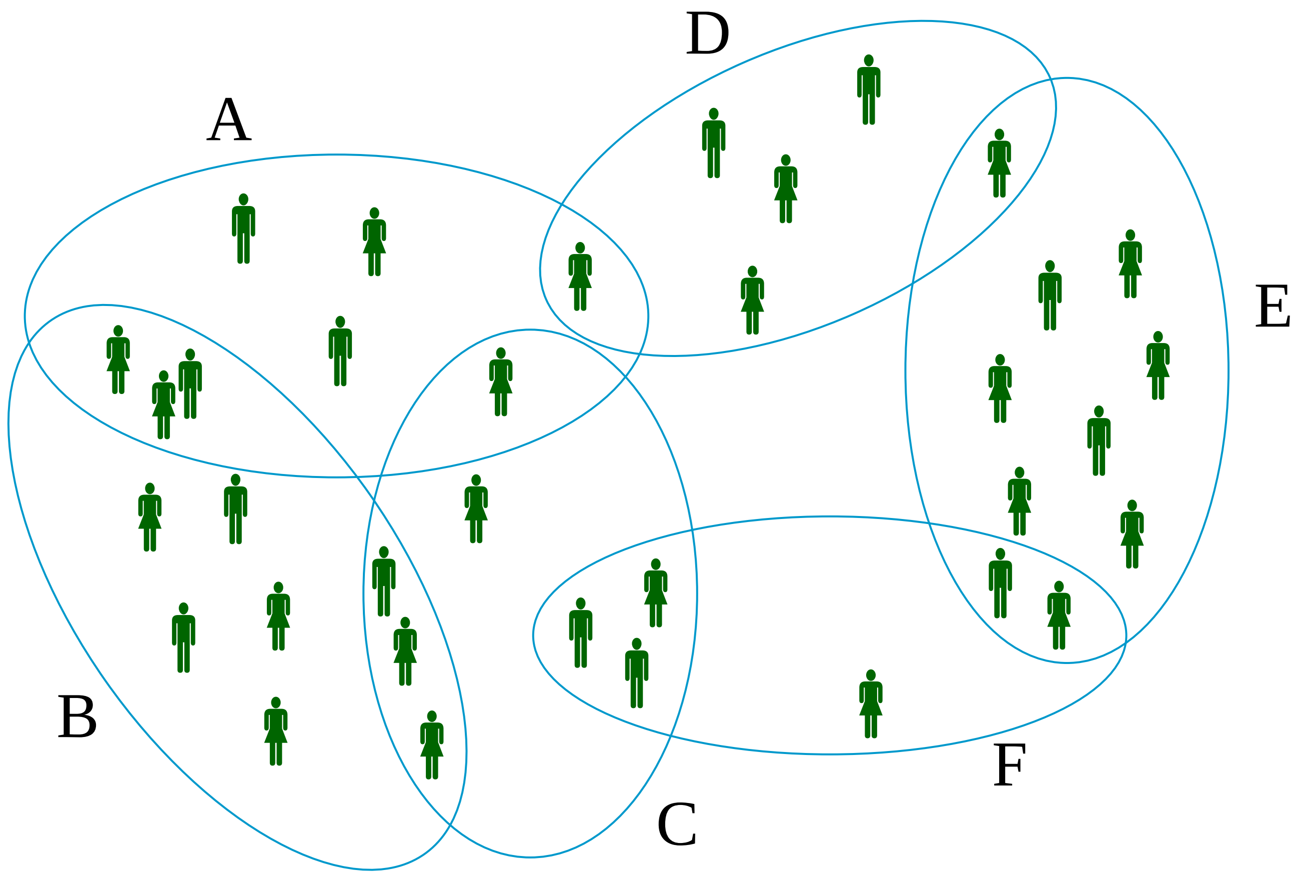
- J. Teghem e M. Pirlot (eds.), “Optimisation approchée en recherche opérationnelle”, 2002
- C. Ribeiro e P. Hansen (eds.), “Essays and surveys in metaheuristics”, 2002
- F. Glover e G. Kochenberger (eds.), “Handbook of metaheuristics”, 2003
- C. Ribeiro, “Notas de aula do curso de metaheurísticas”, http://www.inf.puc-rio.br/~celso/grupo_de_pesquisa.htm

Motivação

- Problema:
  - Alunos cursam disciplinas.
  - Cada disciplina tem uma prova.
  - Há um único horário em que são marcadas provas em cada dia.
  - Provas de duas disciplinas diferentes não podem ser marcadas para o mesmo dia se há alunos que cursam as duas disciplinas.
- Montar um calendário de provas:
  - satisfazendo as restrições de conflito e ...
  - ... minimizando o número de dias necessários para realizar todas as provas.

Motivação

- Modelagem por conjuntos:

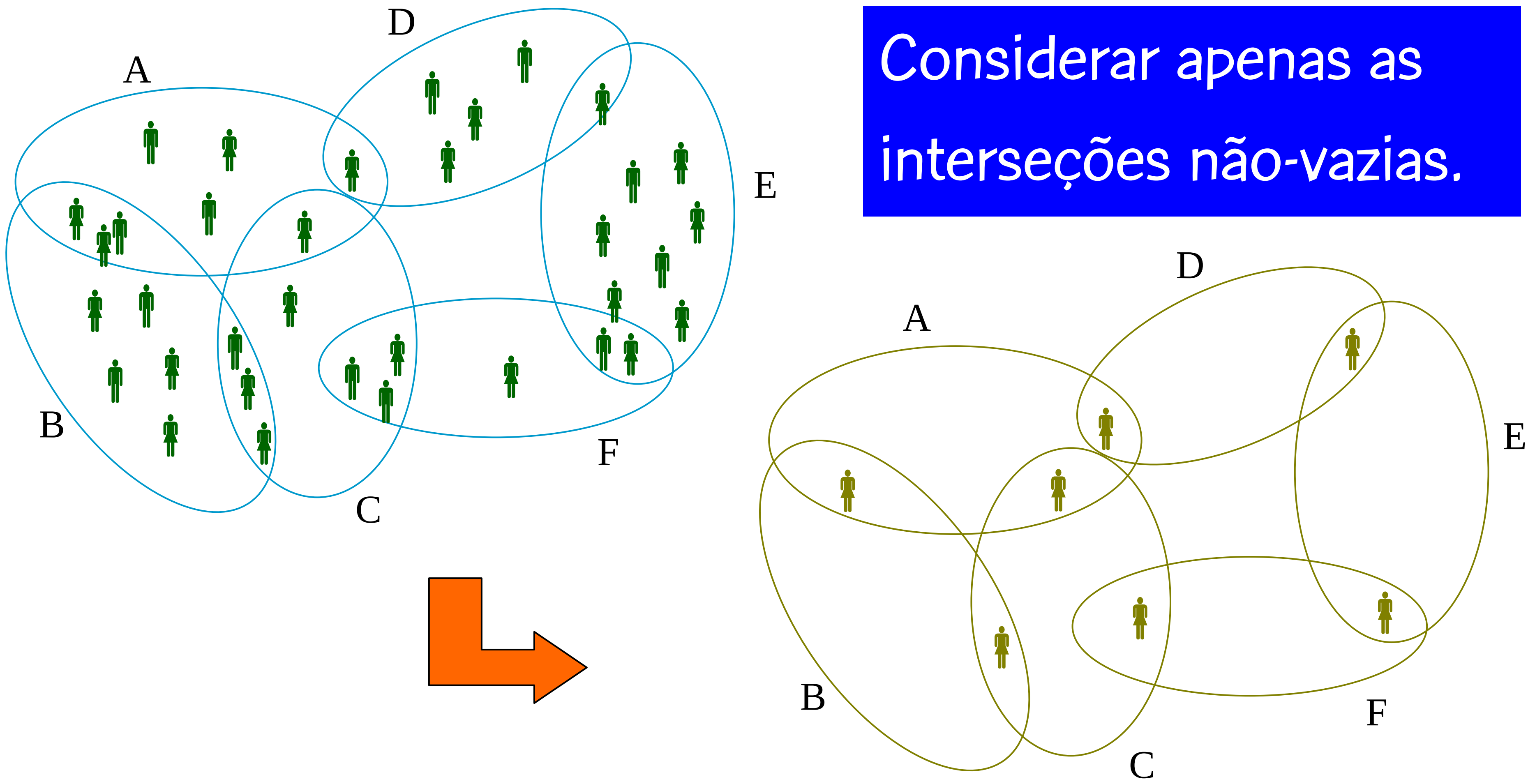


Motivação

- Os alunos que fazem apenas uma disciplina influem na modelagem?
  - Não!
- O número de alunos que cursam simultaneamente um par de disciplinas influi na modelagem?
  - Não!
  - E se o objetivo fosse minimizar o número de alunos "sacrificados"?
  - Neste caso, sim!
- Este modelo pode ser simplificado?

Motivação

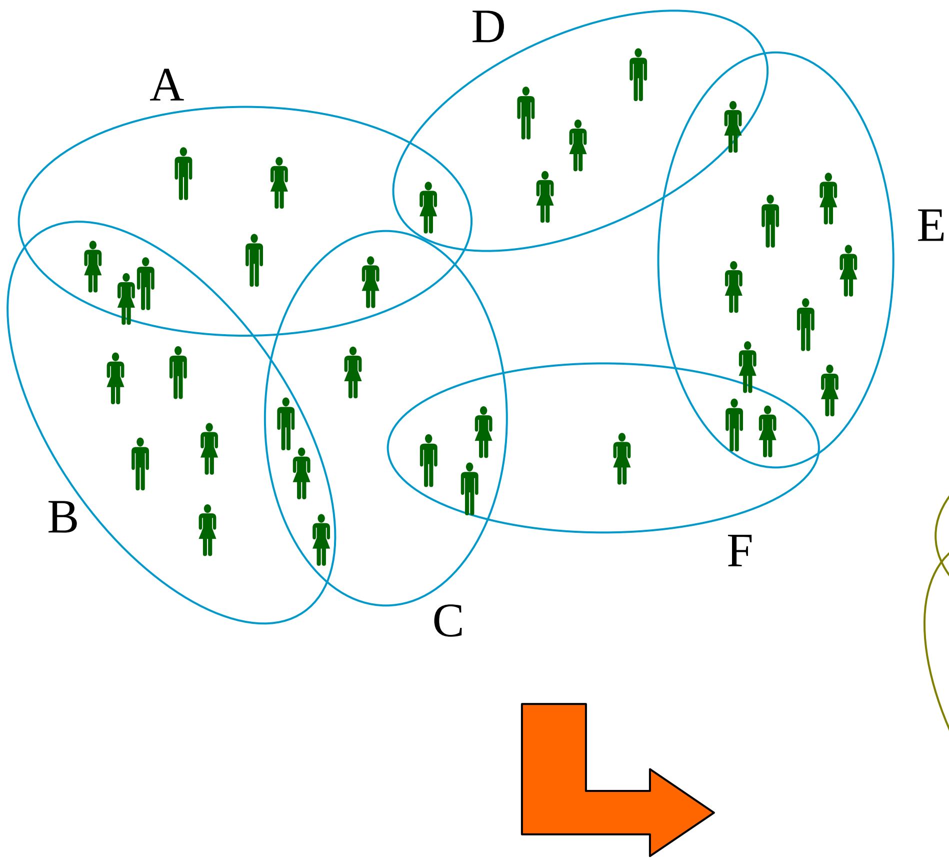
- Simplificação tratando-se apenas as interseções:



Considerar apenas as interseções não-vazias.

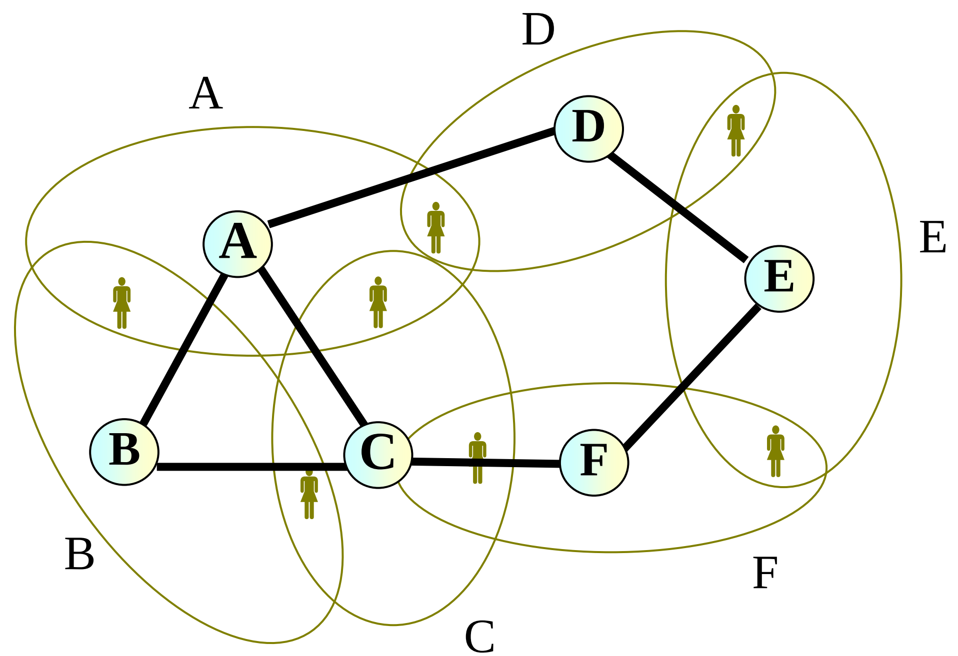
Motivação

- Simplificação com a utilização de um grafo de conflitos:



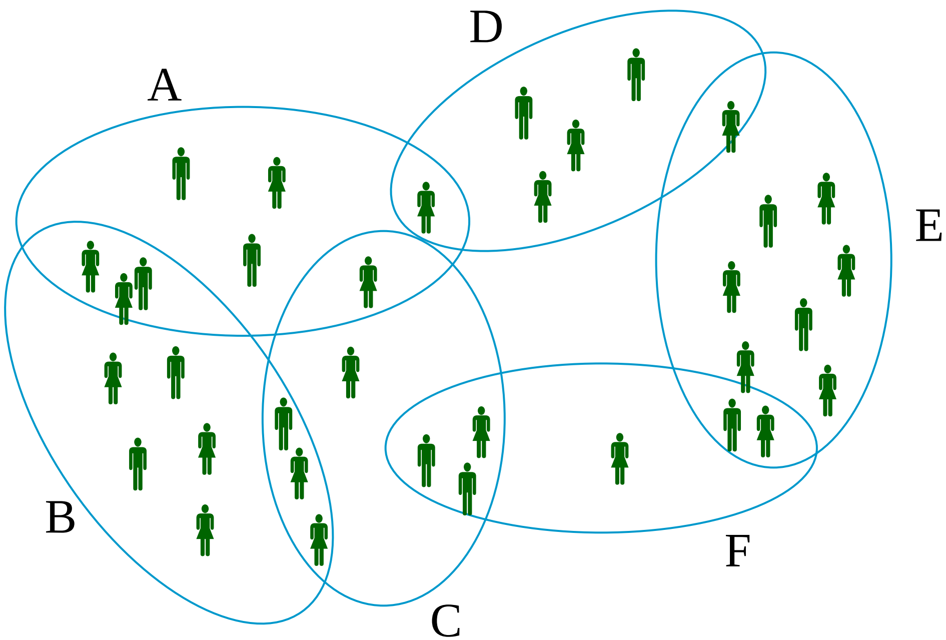
disciplinas ⟶ nós

conflitos ⟶ arestas



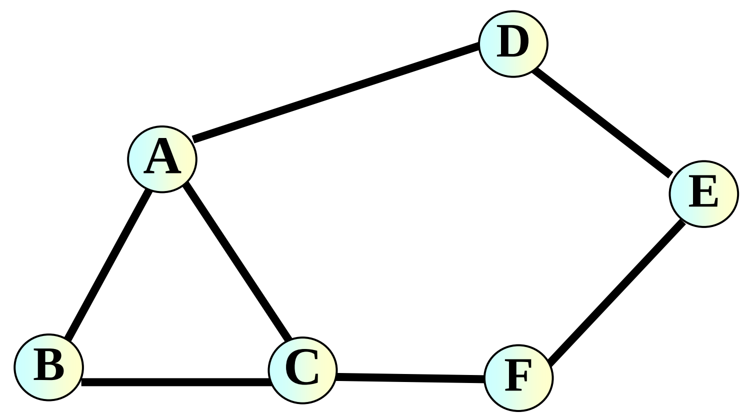
Motivação

- Simplificação com a utilização de um grafo de conflitos:



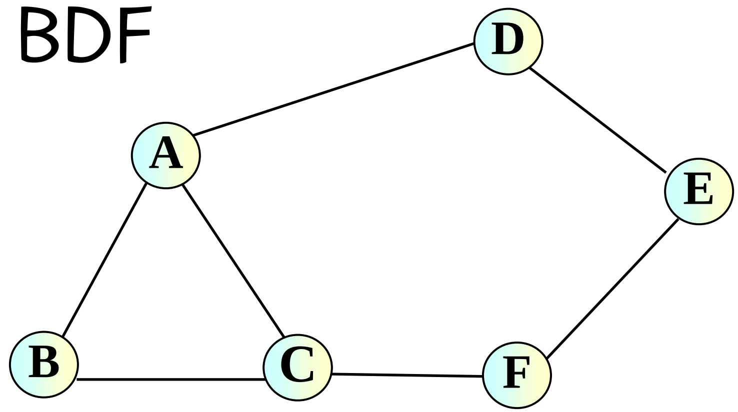
disciplinas ⟶ nós

conflitos ⟶ arestas



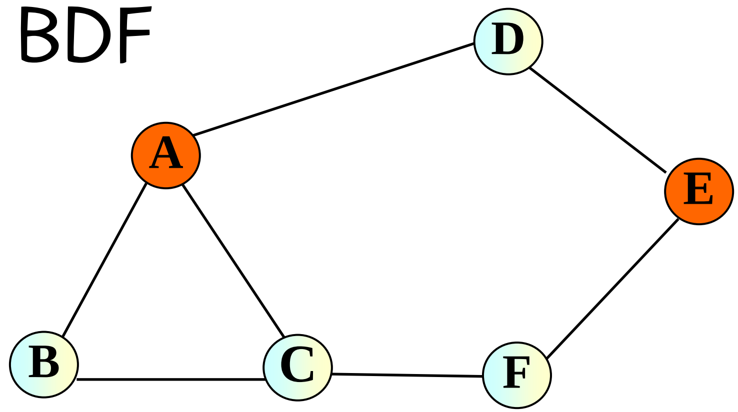
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



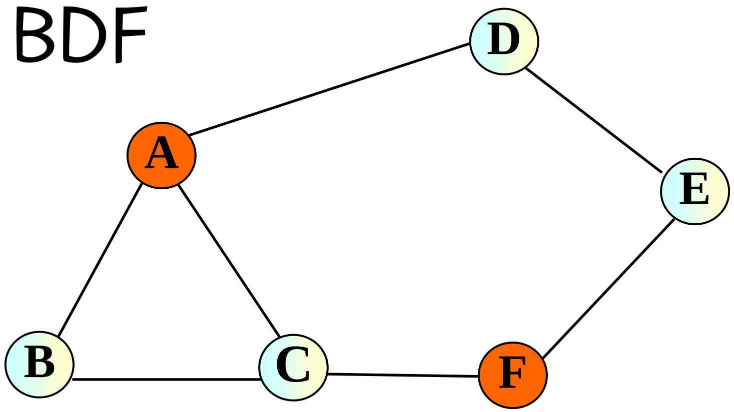
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



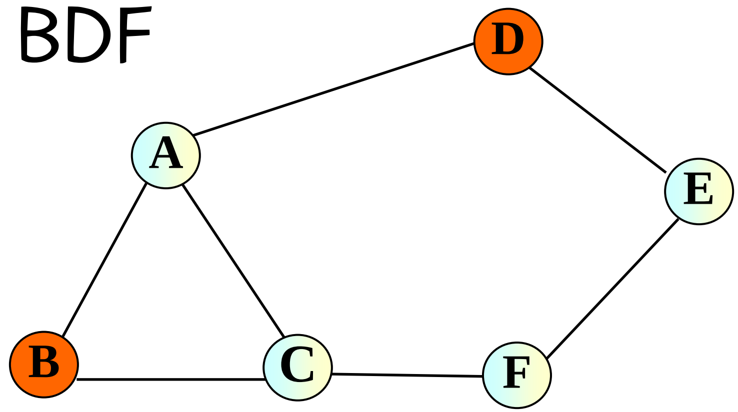
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



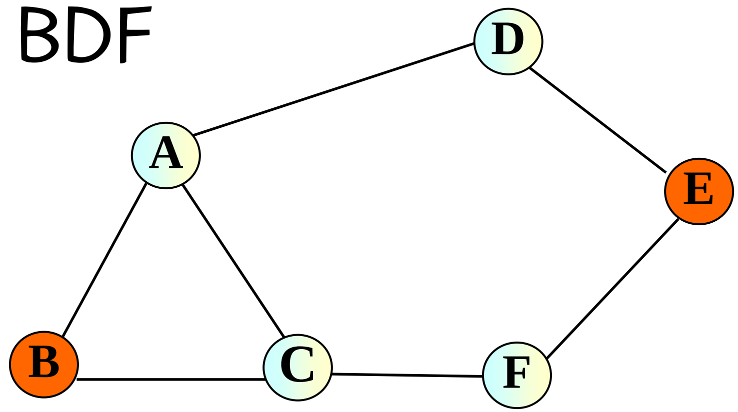
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



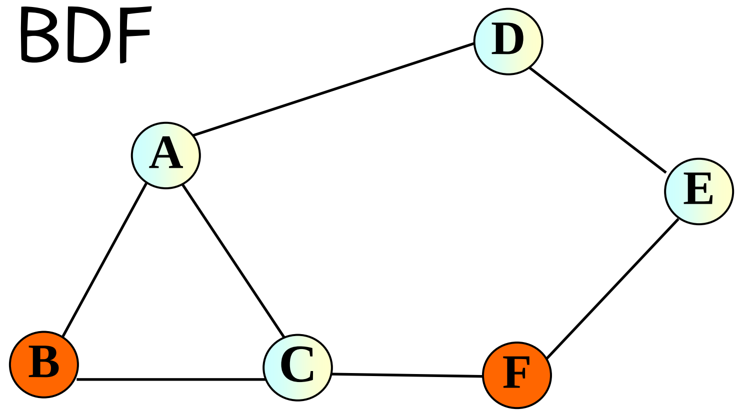
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



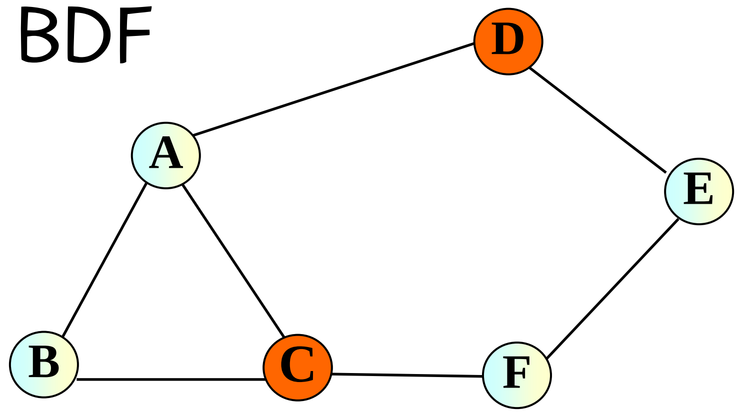
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



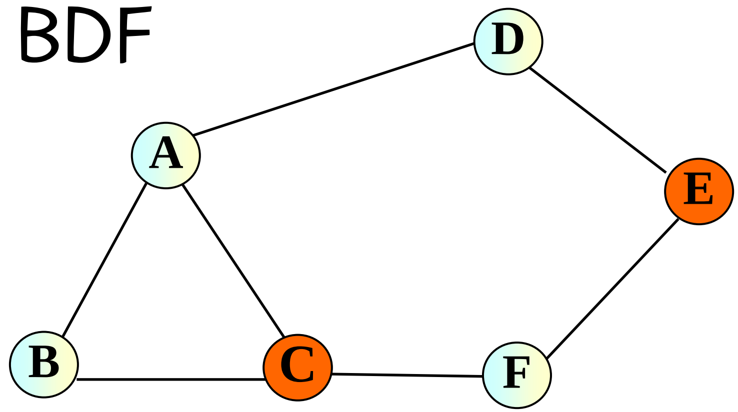
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



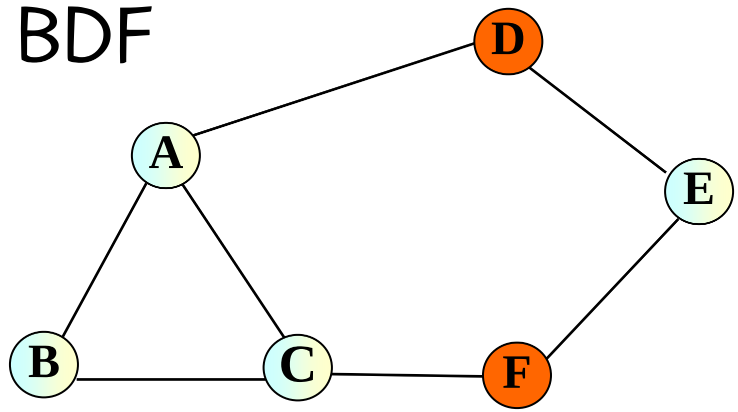
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



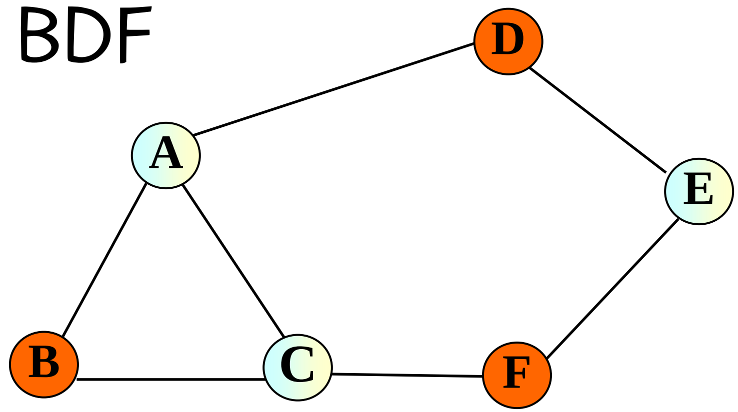
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



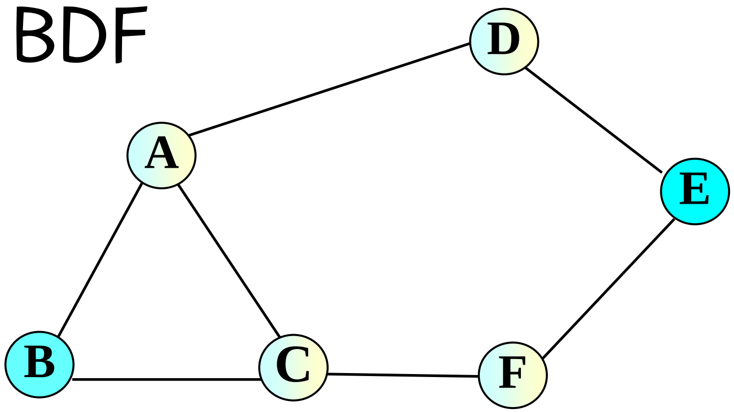
Motivação

- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



Motivação

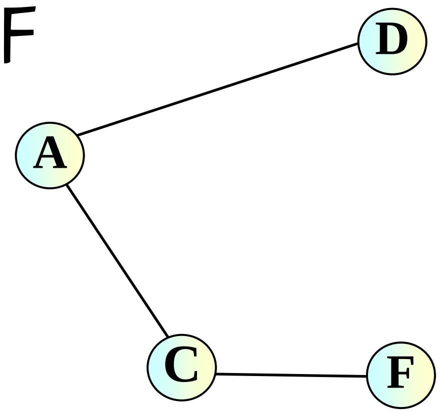
- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



- Escolher por exemplo o par de disciplinas B e E para o primeiro dia e retirá-las do grafo.

Motivação

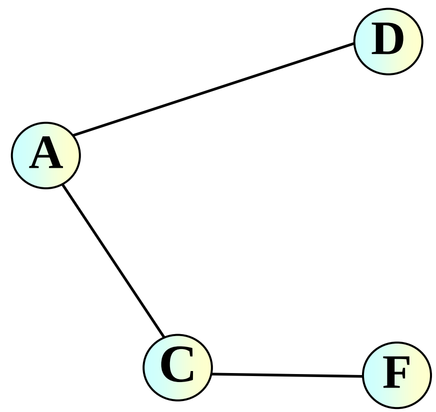
- Como montar um calendário satisfazendo as restrições de conflito?
- Marcar para o primeiro dia qualquer combinação de disciplinas sem conflitos: A, B, C, D, E, F, AE, AF, BD, BE, BF, CD, CE, DF, BDF



- Escolher por exemplo o par de disciplinas B e E para o primeiro dia e retirá-las do grafo.

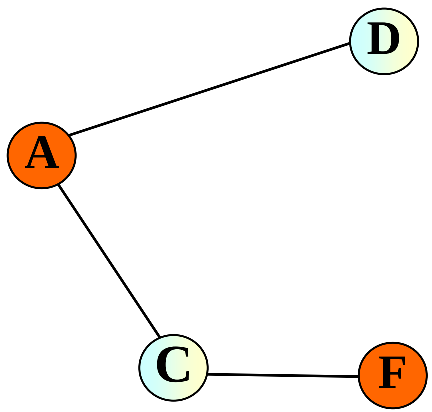
Motivação

- Marcar para o segundo dia qualquer combinação de disciplinas sem conflitos: A, C, D, F, AF, CD, DF



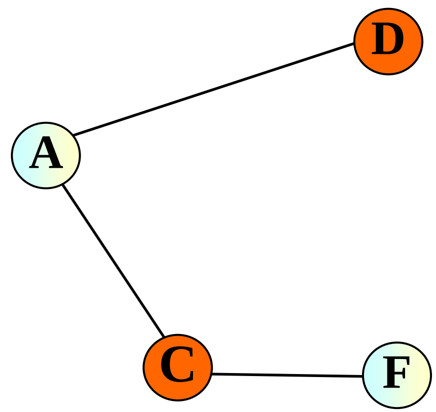
Motivação

- Marcar para o segundo dia qualquer combinação de disciplinas sem conflitos: A, C, D, F, AF, CD, DF



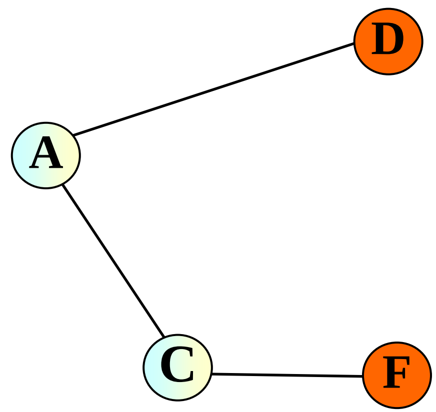
Motivação

- Marcar para o segundo dia qualquer combinação de disciplinas sem conflitos: A, C, D, F, AF, CD, DF



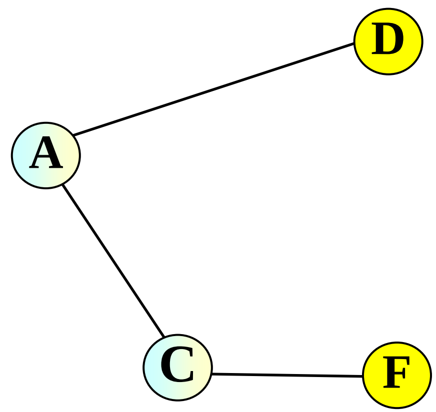
Motivação

- Marcar para o segundo dia qualquer combinação de disciplinas sem conflitos: A, C, D, F, AF, CD, DF



Motivação

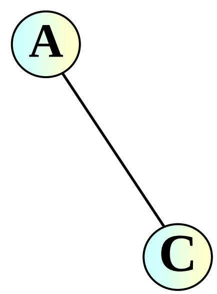
- Marcar para o segundo dia qualquer combinação de disciplinas sem conflitos: A, C, D, F, AF, CD, DF



- Escolher por exemplo o par de disciplinas D e F para o segundo dia e retirá-las do grafo.

Motivação

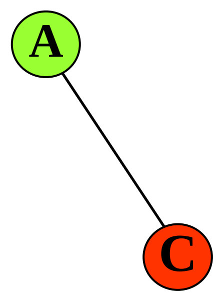
- Marcar para o segundo dia qualquer combinação de disciplinas sem conflitos: A, C, D, F, AF, CD, DF



- Escolher por exemplo o par de disciplinas D e F para o segundo dia e retirá-las do grafo.

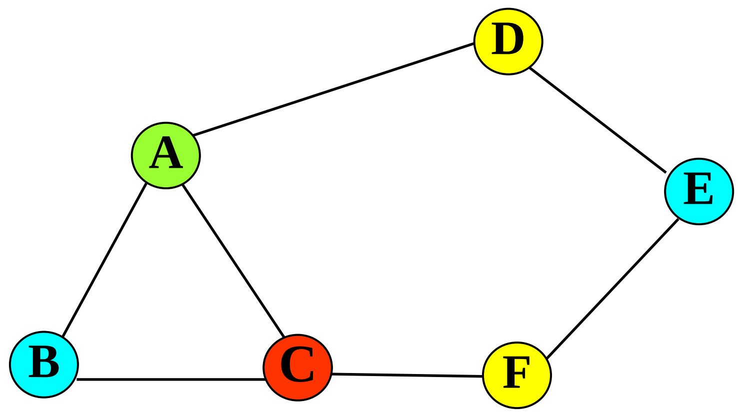
Motivação

- Marcar A ou C para o terceiro dia e a que sobrar para o quarto dia:



Motivação

- A solução assim construída utiliza quatro dias: B-E no primeiro dia, D-F no segundo dia, A no terceiro e C no quarto:

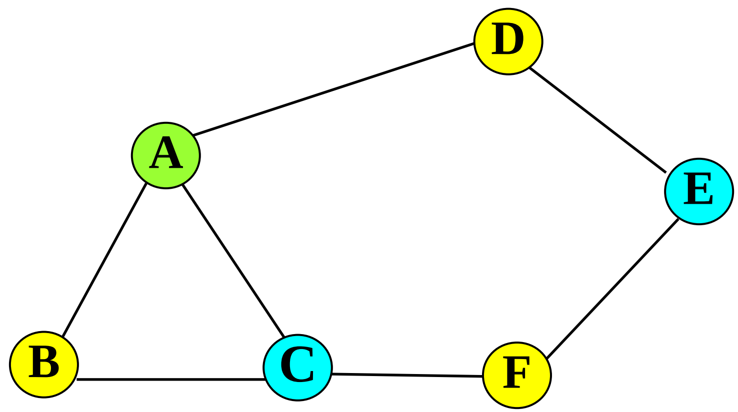


- Uma solução com quatro dias corresponde a colorir os nós do grafo de conflitos com quatro cores!

Motivação

- Esta solução utiliza o menor número possível de dias? ou ...
- É possível montar uma solução com menos dias? ou ...
- É possível colorir o grafo de conflitos com três cores?

Sim!



- É impossível colorir o grafo de conflitos com menos de três cores! (por que?)

Motivação

- Como montar um escalonamento com o menor número possível de dias?
- Recordando: um subconjunto independente de um grafo é um subconjunto de nós sem nenhuma aresta entre eles.
- Logo, um conjunto de disciplinas que forma um subconjunto independente dos nós do grafo de conflitos pode ter suas provas marcadas para o mesmo dia.
- Os nós deste subconjunto independente podem receber a mesma cor.

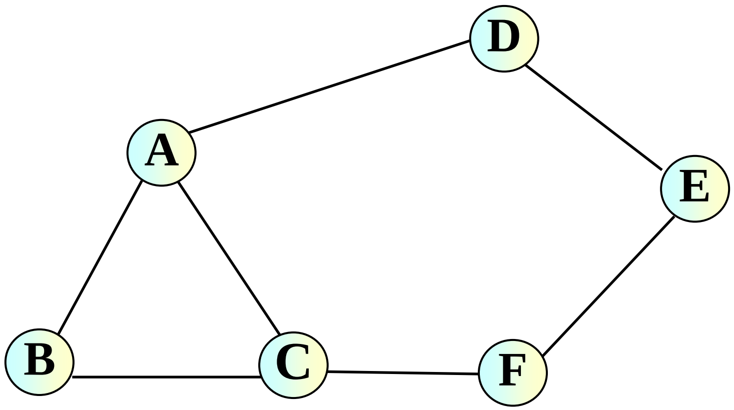
Motivação

- Quanto mais disciplinas forem marcadas para o primeiro dia, menos disciplinas sobram para os dias seguintes e, portanto, serão necessários menos dias para realizar as provas das disciplinas restantes.
- Então, marcar para o primeiro dia as provas das disciplinas correspondentes a um subconjunto independente máximo.
- Retirar os nós correspondentes a estas disciplinas do grafo de conflitos.
- Continuar procedendo da mesma maneira, até as provas de todas as disciplinas terem sido marcadas.

Motivação

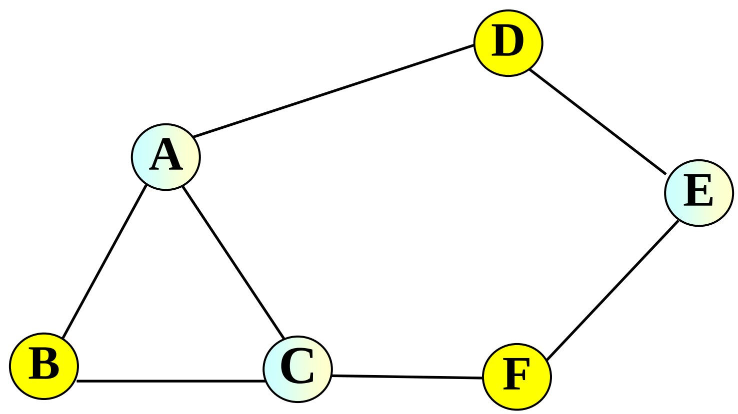
- Subconjunto independente máximo no grafo de conflito?

BDF



Motivação

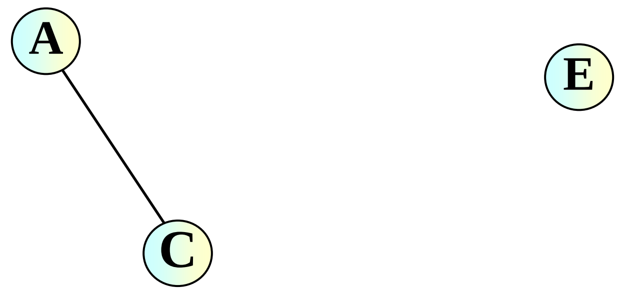
- Subconjunto independente máximo no grafo de conflito?
  BDF



- Eliminar os nós B, D e F do grafo de conflitos.

Motivação

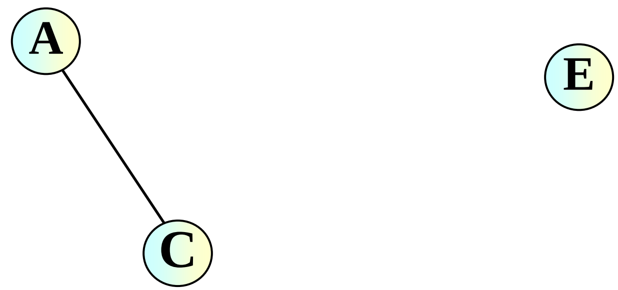
- Subconjunto independente máximo no grafo de conflito? BDF



- Eliminar os nós B, D e F do grafo de conflitos.

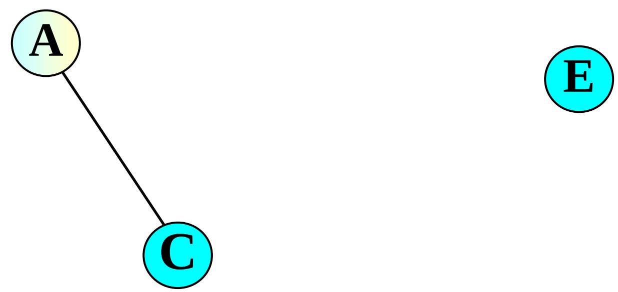
Motivação

- Subconjunto independente máximo no grafo de conflito?
  CE



Motivação

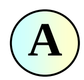
- Subconjunto independente máximo no grafo de conflito?
  CE



- Eliminar os nós C e E do grafo de conflitos.

Motivação

- Subconjunto independente máximo no grafo de conflito?
  CE

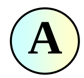


- Eliminar os nós C e E do grafo de conflitos.

Motivação

- Subconjunto independente máximo no grafo de conflito?

  A



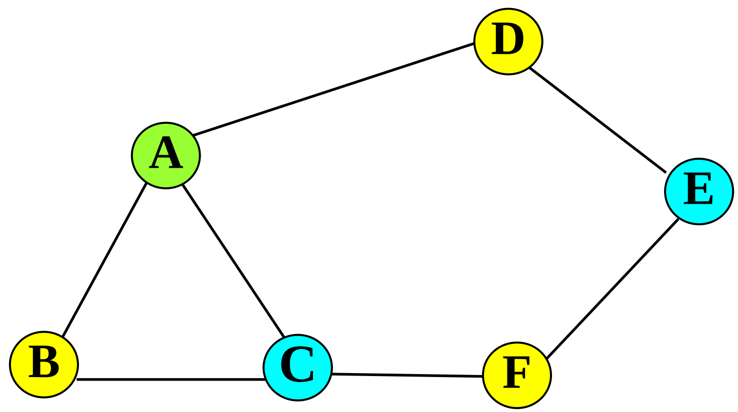
Motivação

- Subconjunto independente máximo no grafo de conflito?
  A

- Eliminar o nó A do grafo de conflitos.
- Solução completa (todos nós coloridos).

Motivação

- A solução encontrada usa o número mínimo de cores para colorir o grafo, logo o número de dias para aplicar todas as provas é mínimo:



- Foi então proposto um algoritmo para colorir um grafo com o menor número de cores.

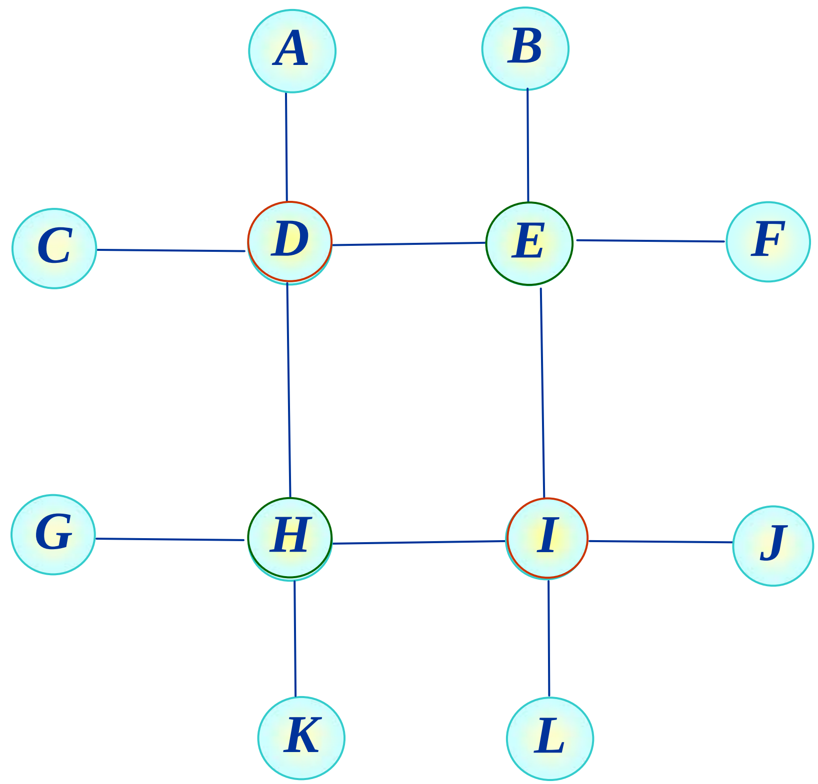
Motivação

- O passo básico deste algoritmo corresponde a determinar um subconjunto independente máximo.
- Entretanto, este subproblema a ser resolvido a cada iteração já é um problema NP-completo por si só, ou seja, cada subproblema é computacionalmente difícil.
- Além deste procedimento ser computacionalmente difícil, é possível garantir que este algoritmo sempre encontra a solução ótima com o número mínimo de cores?
  - Ou demonstrar que sim, ou dar um contra-exemplo.

Motivação

- Considerando-se o grafo abaixo, qual solução seria encontrada pelo algoritmo?

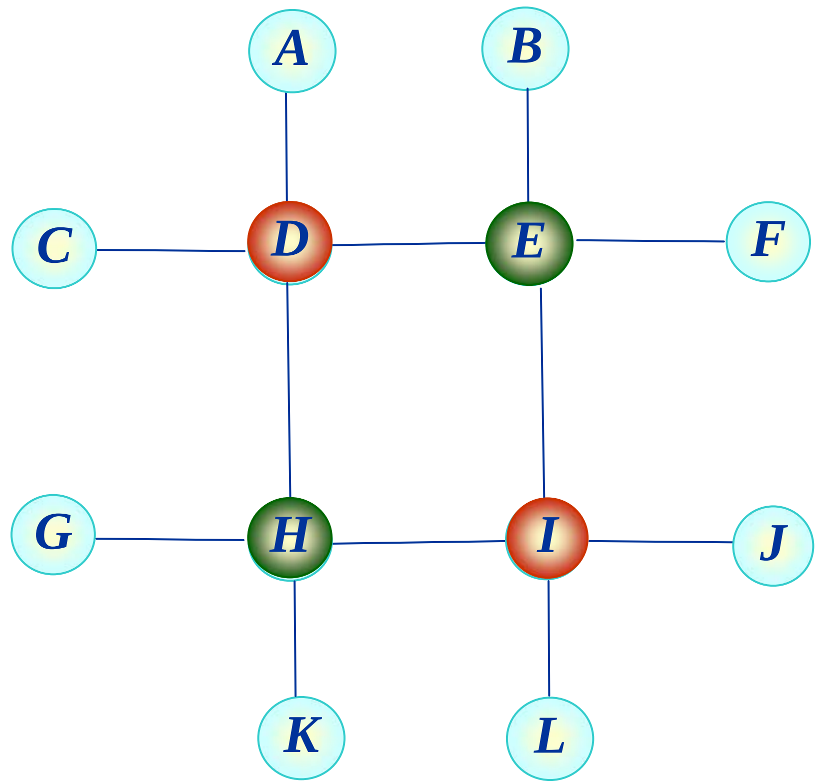
Os oito nós externos formam um subconjunto independente de cardinalidade máxima e podem todos ser coloridos com a mesma cor.



Motivação

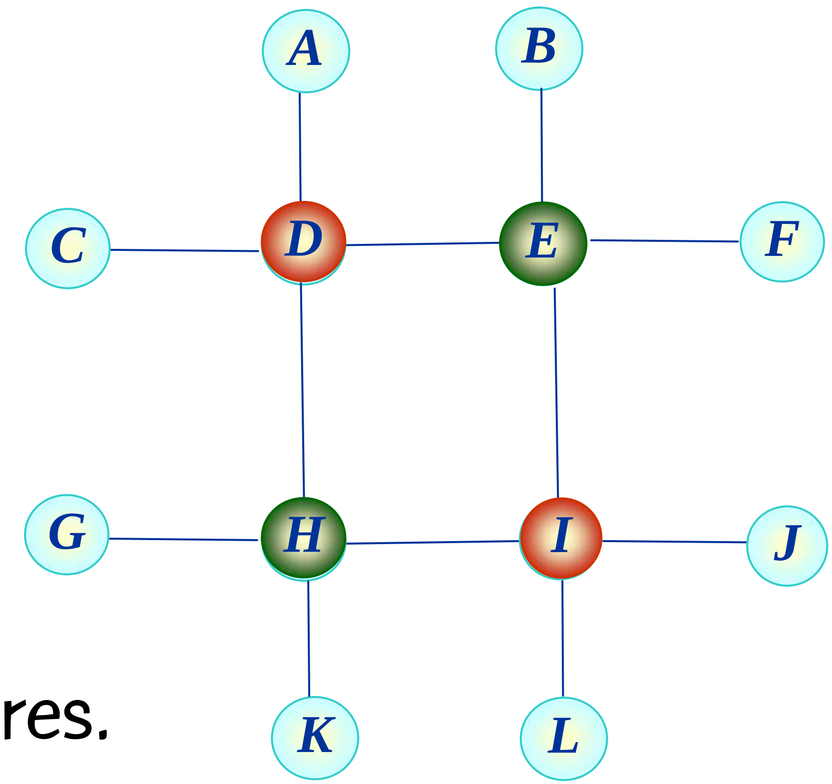
- Considerando-se o grafo abaixo, qual solução seria encontrada pelo algoritmo?

Os oito nós externos formam um subconjunto independente de cardinalidade máxima e podem todos ser coloridos com a mesma cor.



Motivação

- Considerando-se o grafo abaixo, qual solução seria encontrada pelo algoritmo?

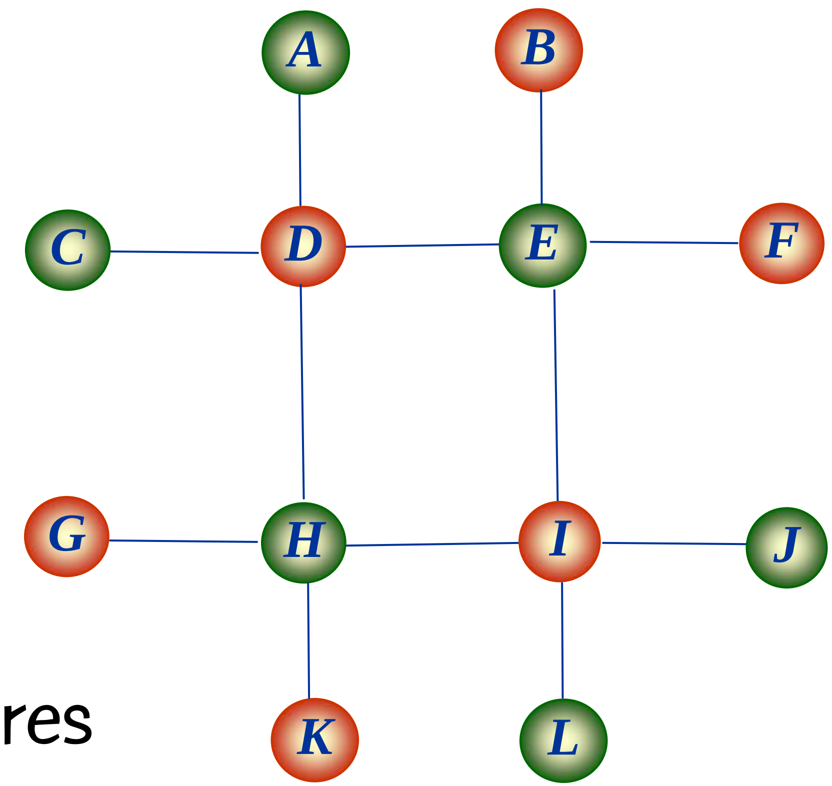


Esta solução é ótima, ou é possível colorir este grafo com menos cores?

Sim: o grafo pode ser colorido com apenas duas cores.

Motivação

- Considerando-se o grafo abaixo, qual solução seria encontrada pelo algoritmo?



Esta solução é ótima, ou é possível colorir este grafo com menos cores?

<u>Sim</u>: o grafo pode ser colorido com apenas duas cores

Motivação

- O algoritmo proposto nem sempre encontra a solução ótima (isto é, uma solução com o número mínimo de cores).
- Este algoritmo é então um algoritmo aproximado ou uma heurística para o problema de coloração de grafos.
- Algoritmos exatos vs. algoritmos aproximados:
  - qualidade da solução
  - tempo de processamento

Algoritmos construtivos

- Problema de otimização combinatória: Dado um conjunto finito $E = \{1,2, \ldots,n\}$ e uma função de custo $c: 2^E \rightarrow R$, encontrar $S^* \in F$ tal que $c(S^*) \leq c(S)\ \forall S \in F$, onde $F \in 2^E$ é o conjunto de soluções viáveis do problema (minimização).
- Conjunto discreto de soluções (vetores de variáveis 0-1, caminhos, ciclos, ...) com um número finito de elementos.
- Construção de uma solução: selecionar seqüencialmente elementos de E, eventualmente descartando alguns já selecionados, de tal forma que ao final se obtenha uma solução viável, i.e. pertencente a F.

Algoritmos construtivos

- Exemplo: problema da mochila

  E: conjunto de itens

  F: subconjuntos de E que satisfazem à restrição $\sum_{e \in S} a_e \leq b$

  $c(S) = \sum_{e \in S} c_e$

  c_e: lucro do item e

  a_e: peso do item e

  b: capacidade da mochila

Algoritmos construtivos

- Exemplo: problema do caixeiro viajante

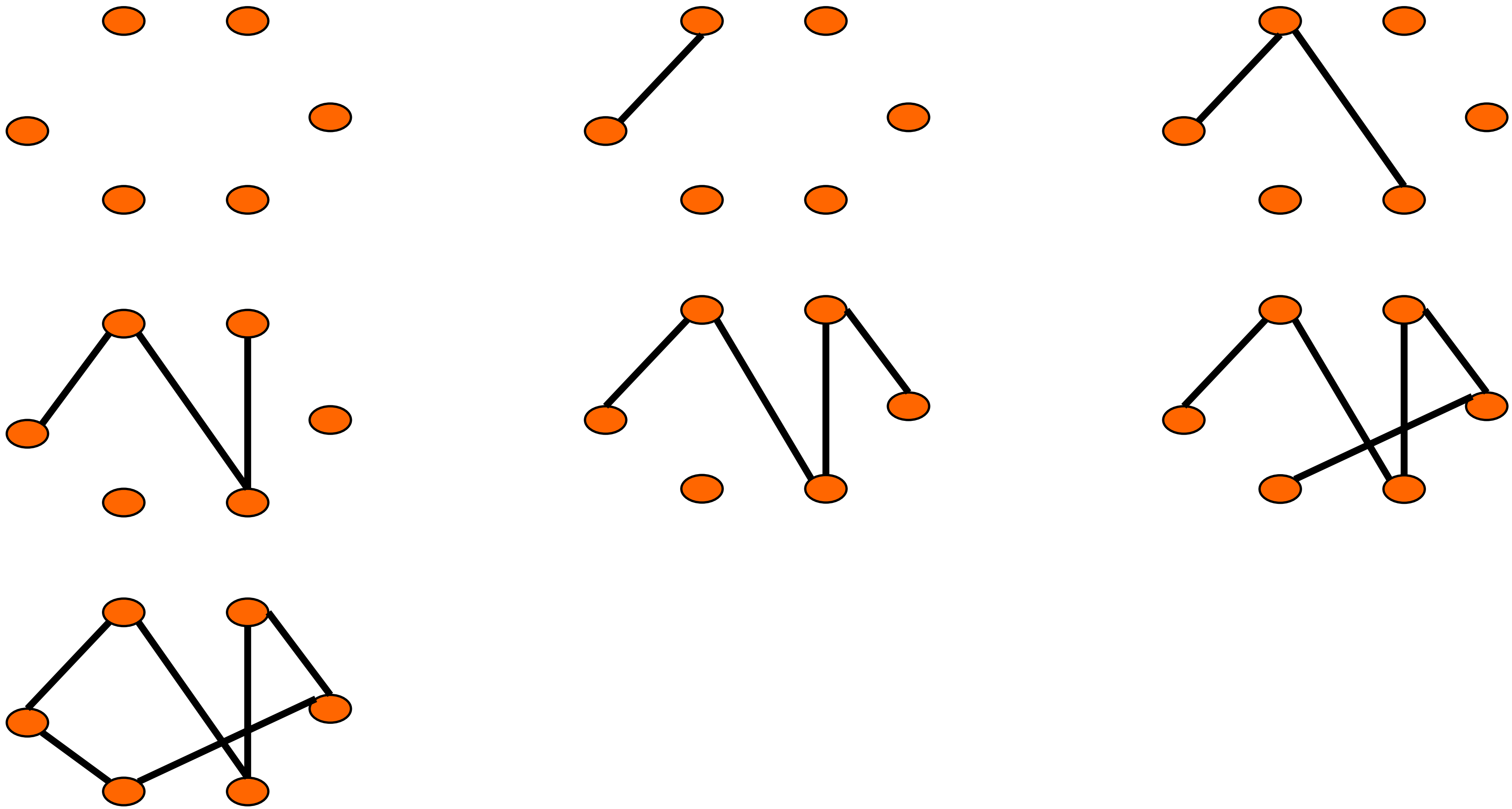
E: conjunto de arestas

F: subconjuntos de arestas que formam um circuito hamiltoniano (isto é, que visita cada cidade exatamente uma única vez)

$c(S) = \sum_{e \in S} c_e$

c_e: custo (comprimento) da aresta $e = (i,j)$

Algoritmos construtivos

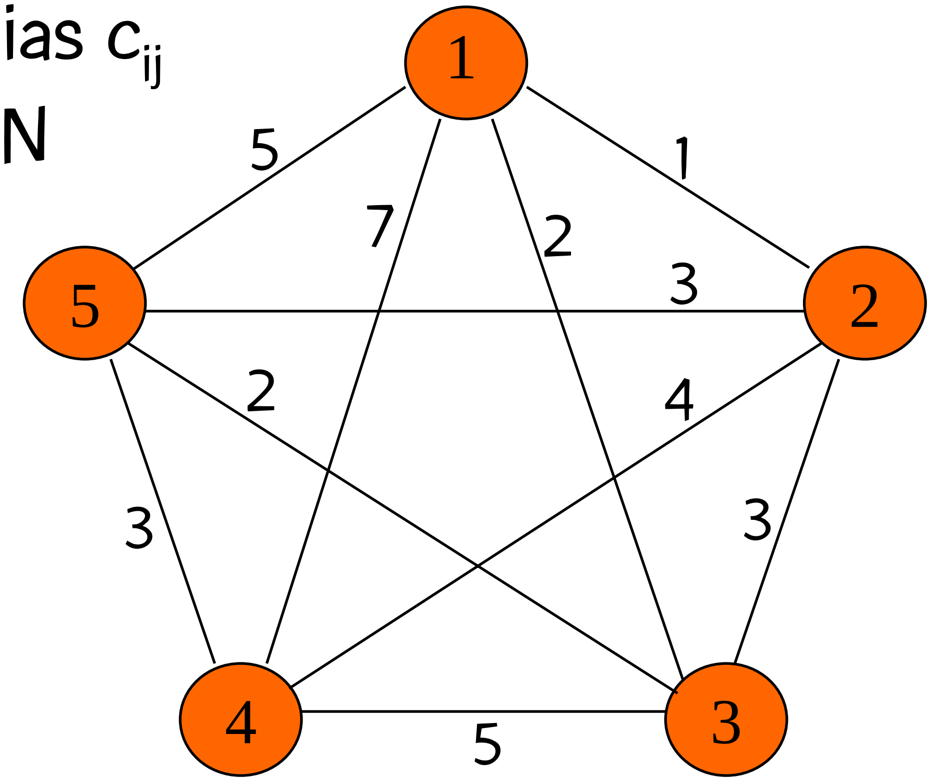


Problema do caixeiro viajante

Matrix de distâncias c_{ij}

Conjunto de nós N

$|N|=5$



Problema do caixeiro viajante

- Algoritmo do vizinho mais próximo:

Escolher o nó inicial i e fazer $N \leftarrow N\text{-}\{i\}$.

Enquanto $N \neq \emptyset$ fazer:

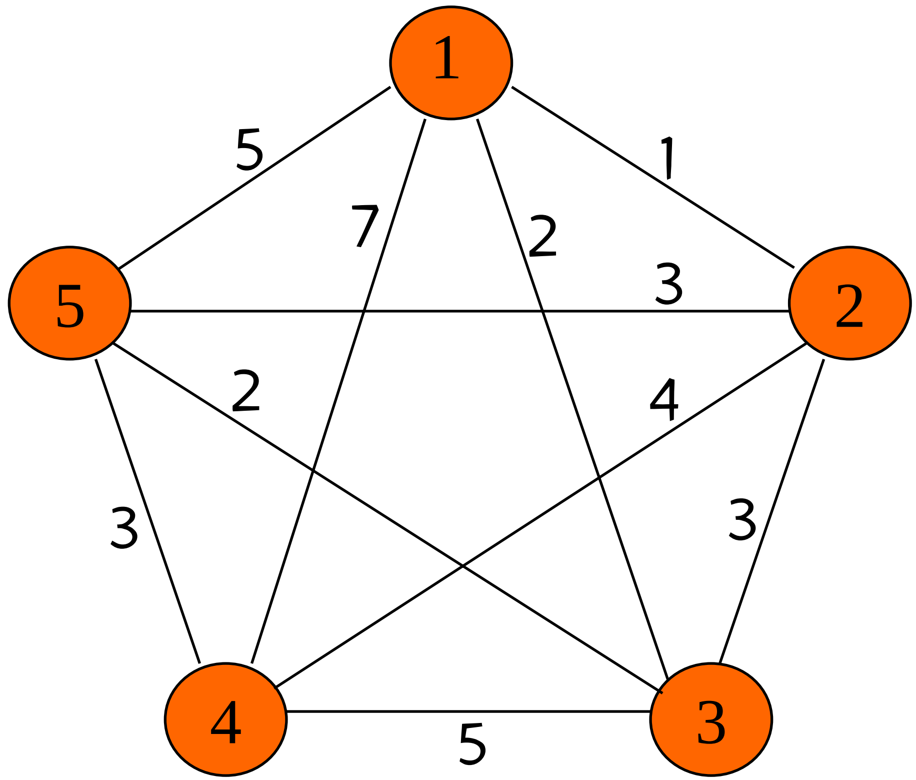
  Obter $j \in N$ tal que $c_{i,j} = \min_{k \in N} \{c_{i,k}\}$.

  $N \leftarrow N\text{-}\{j\}$

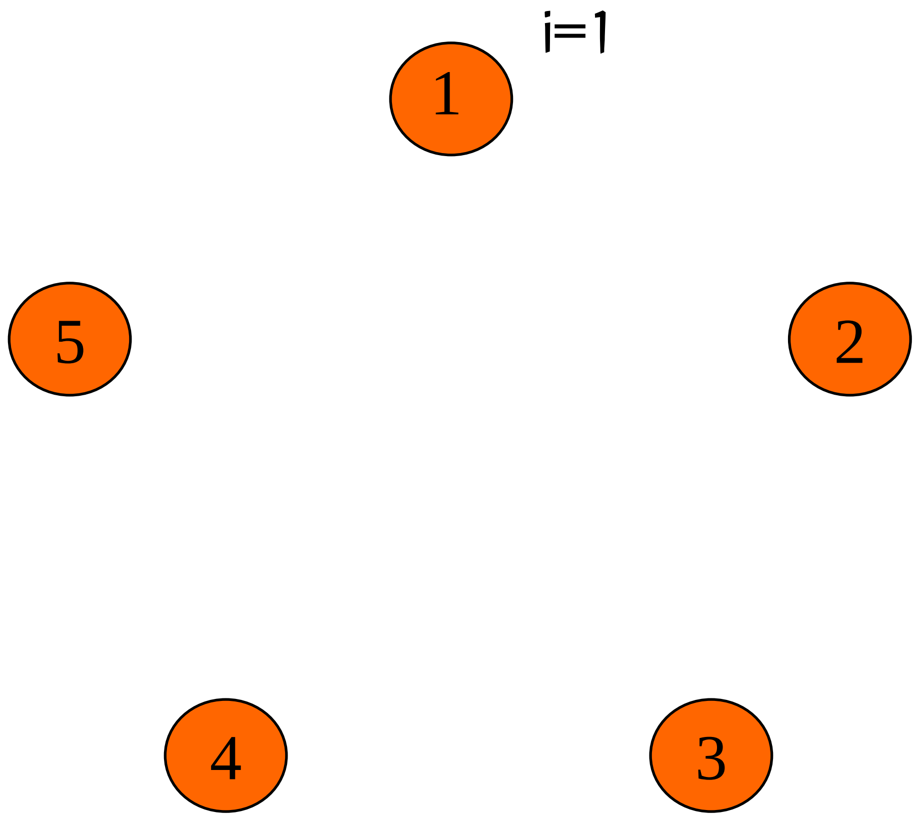
  $i \leftarrow j$

Fim-enquanto

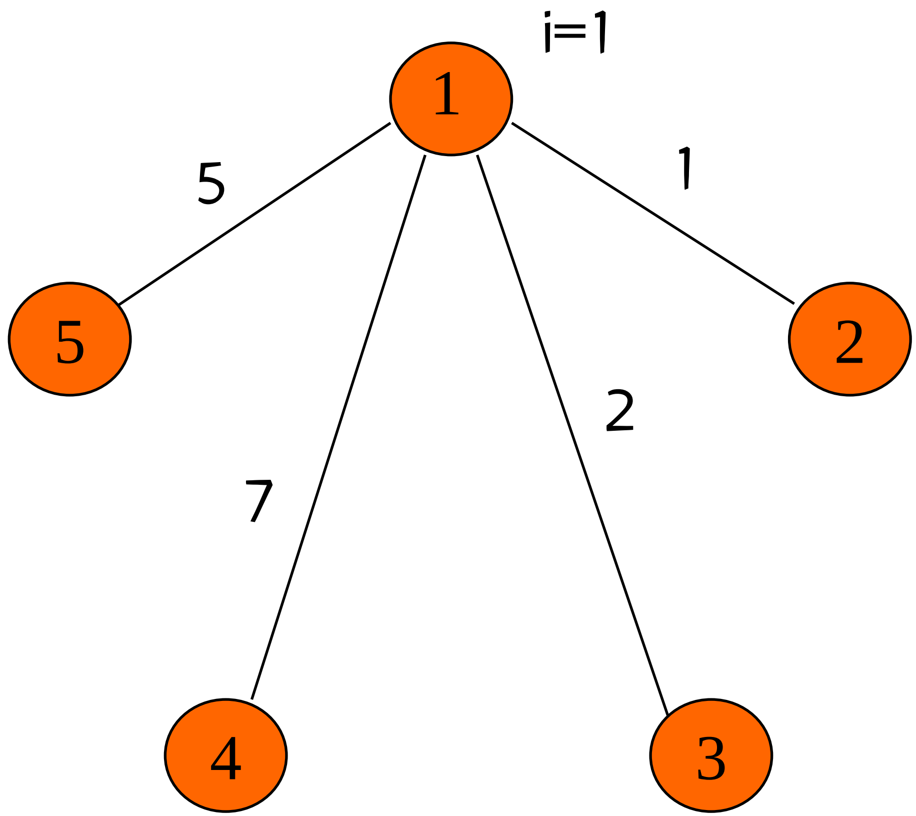
Problema do caixeiro viajante



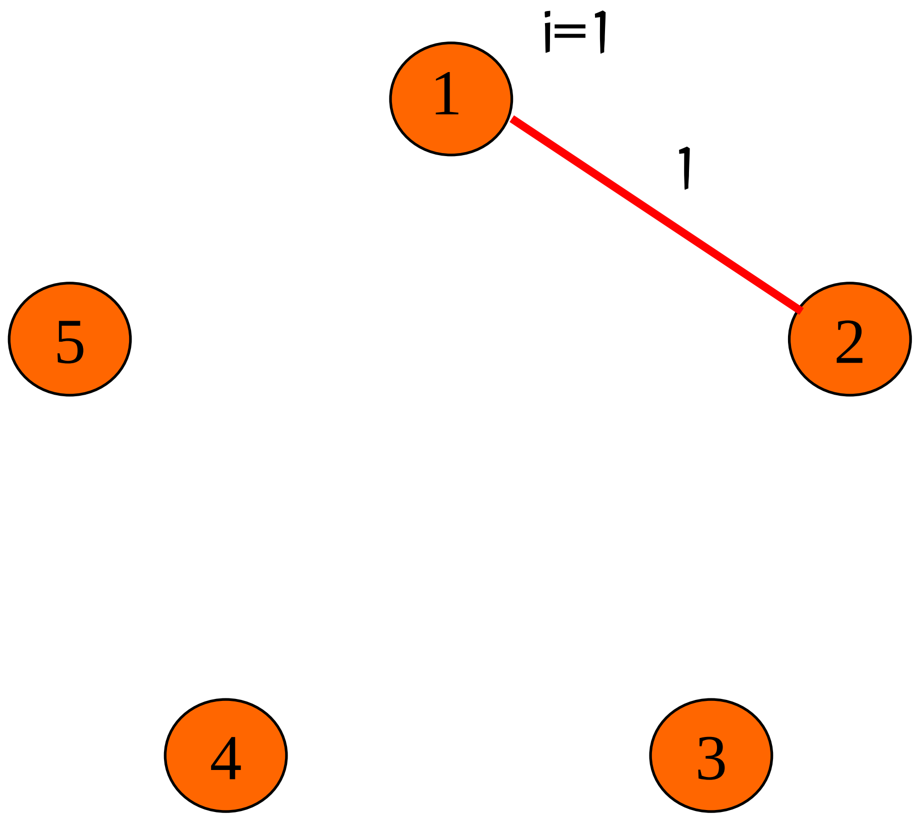
Problema do caixeiro viajante



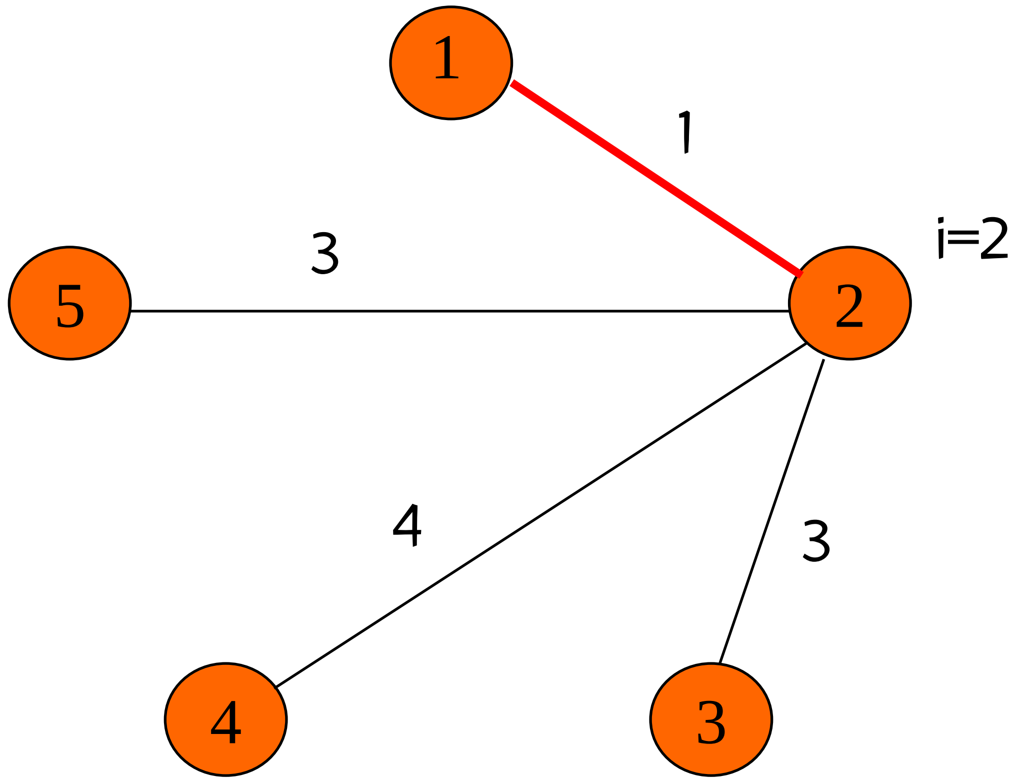
Problema do caixeiro viajante



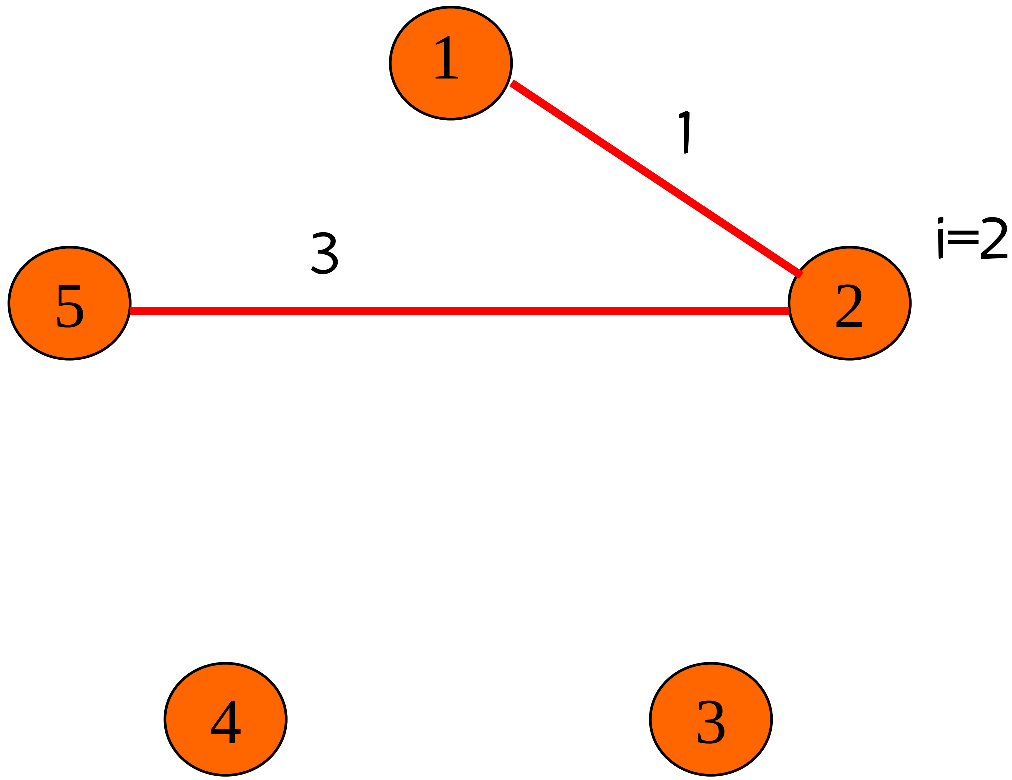
Problema do caixeiro viajante



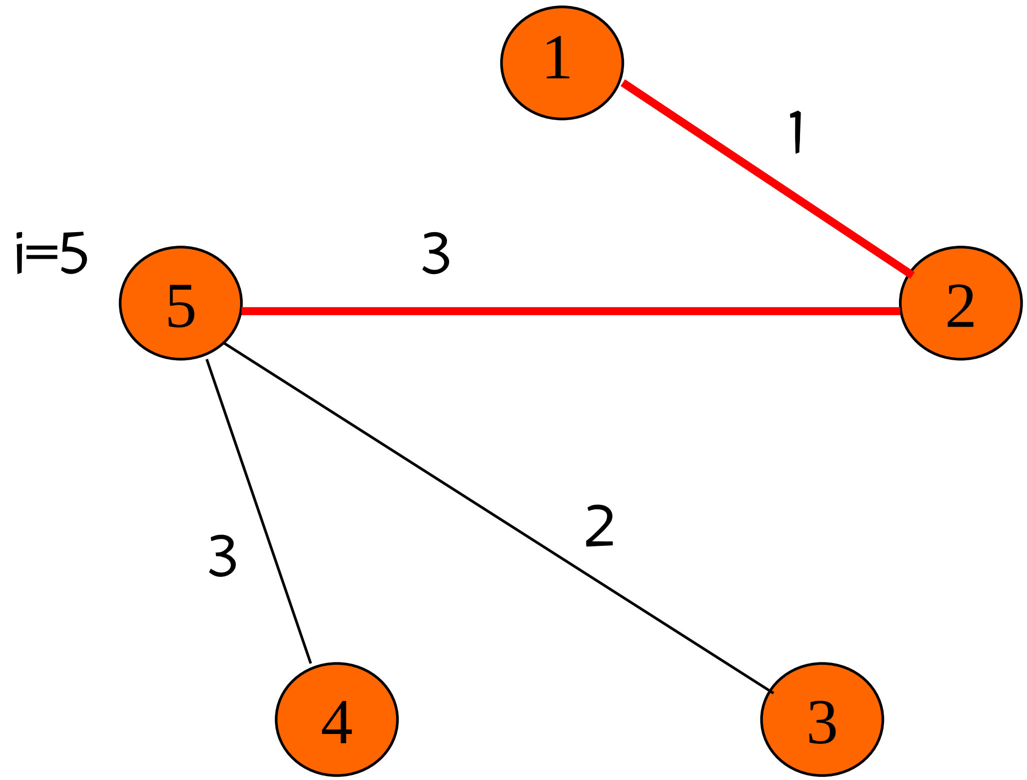
Problema do caixeiro viajante



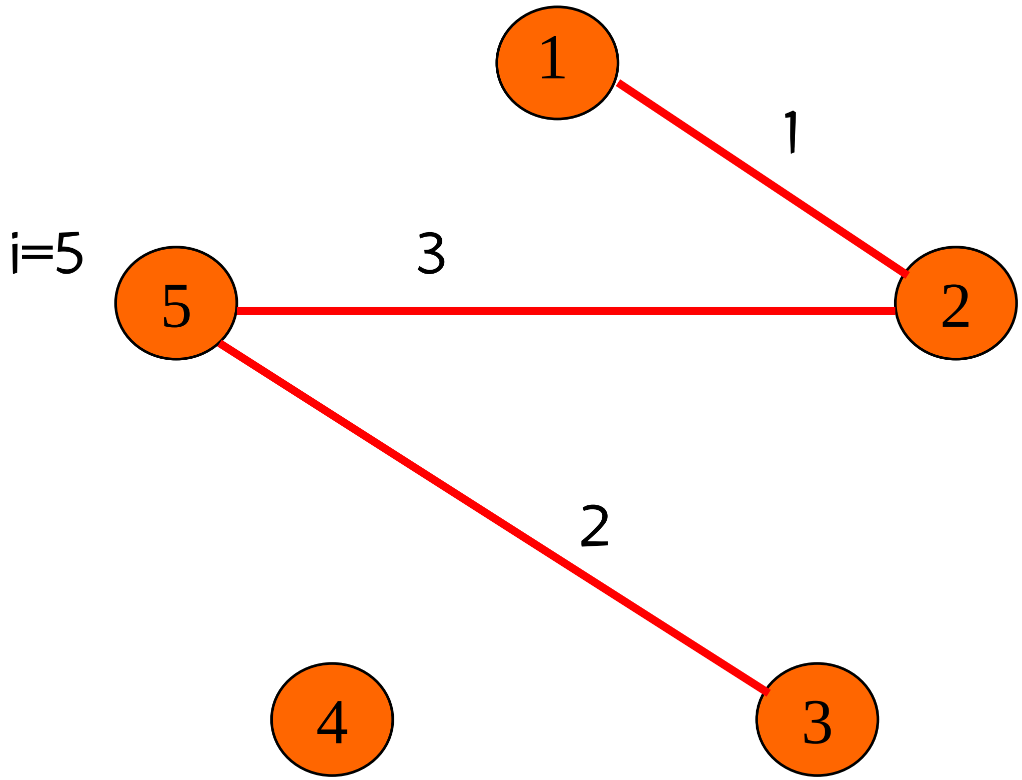
Problema do caixeiro viajante



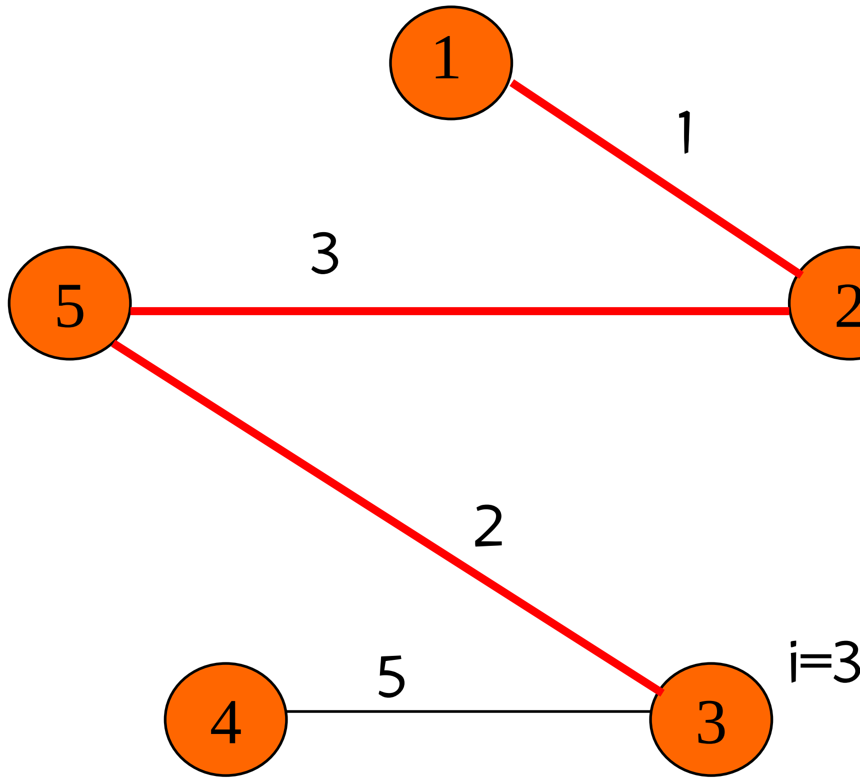
Problema do caixeiro viajante



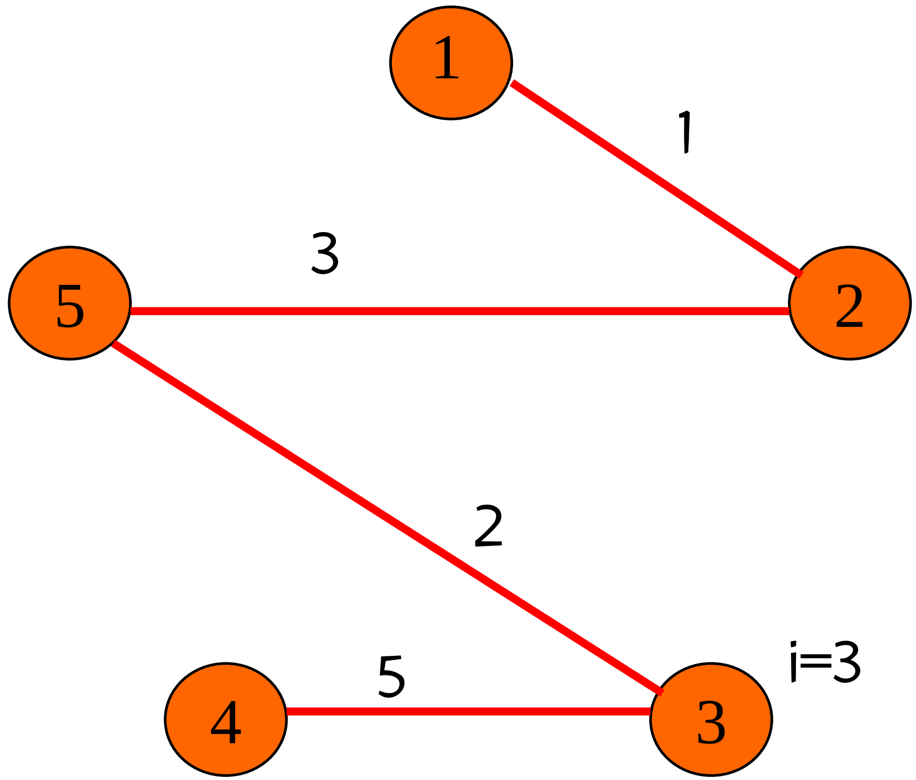
Problema do caixeiro viajante



Problema do caixeiro viajante

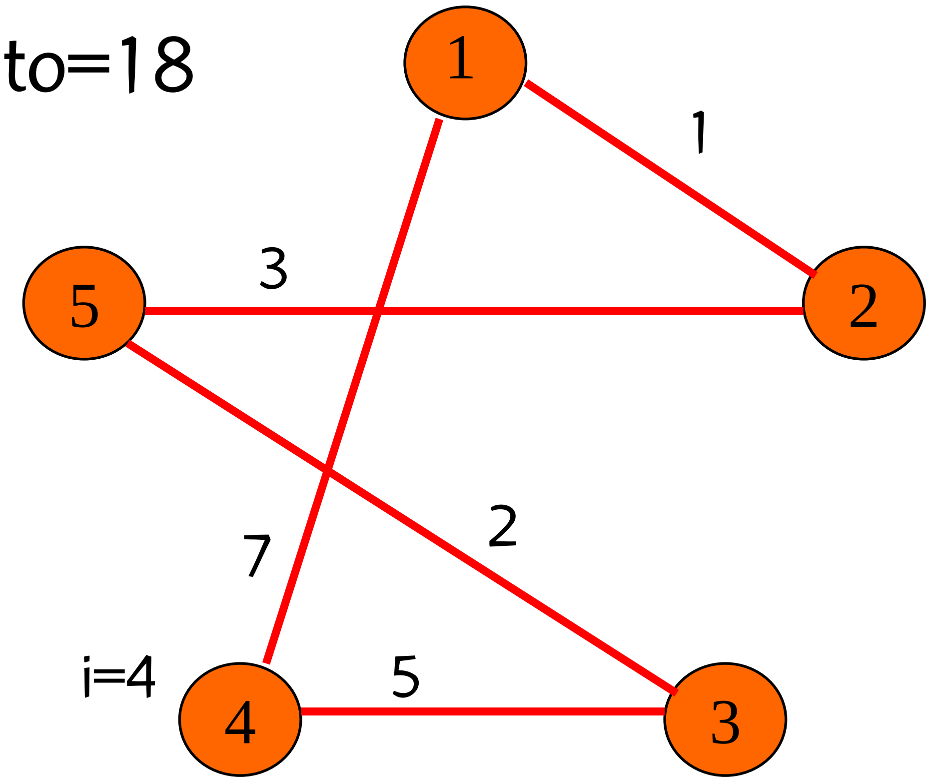


Problema do caixeiro viajante



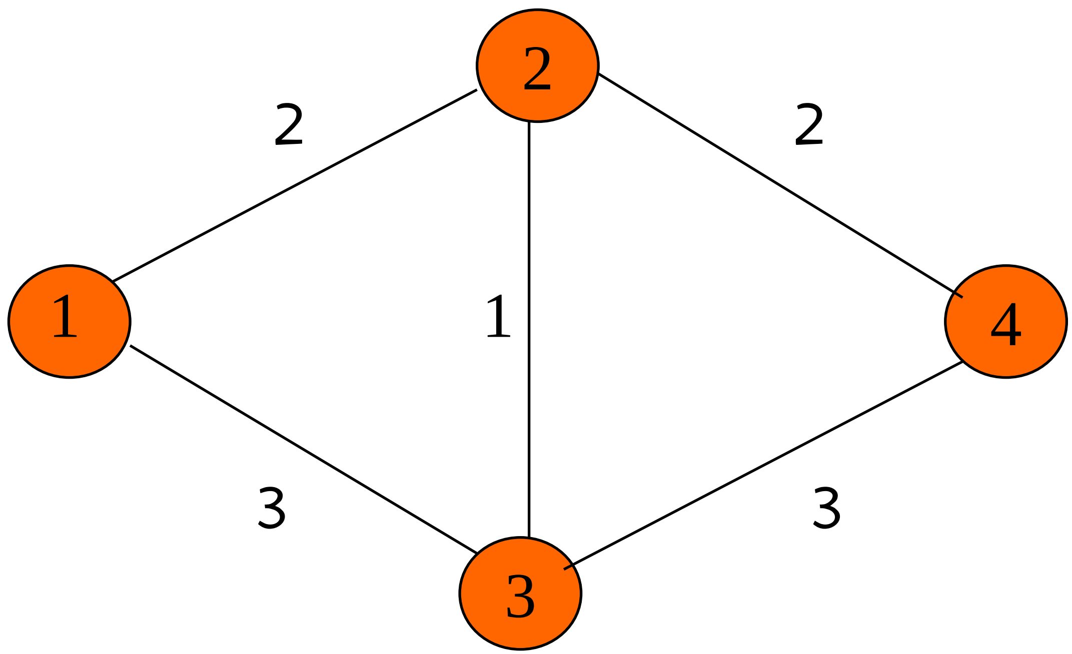
Problema do caixeiro viajante

comprimento=18



Problema do caixeiro viajante

- Algoritmos construtivos simples podem falhar mesmo para casos muito simples:



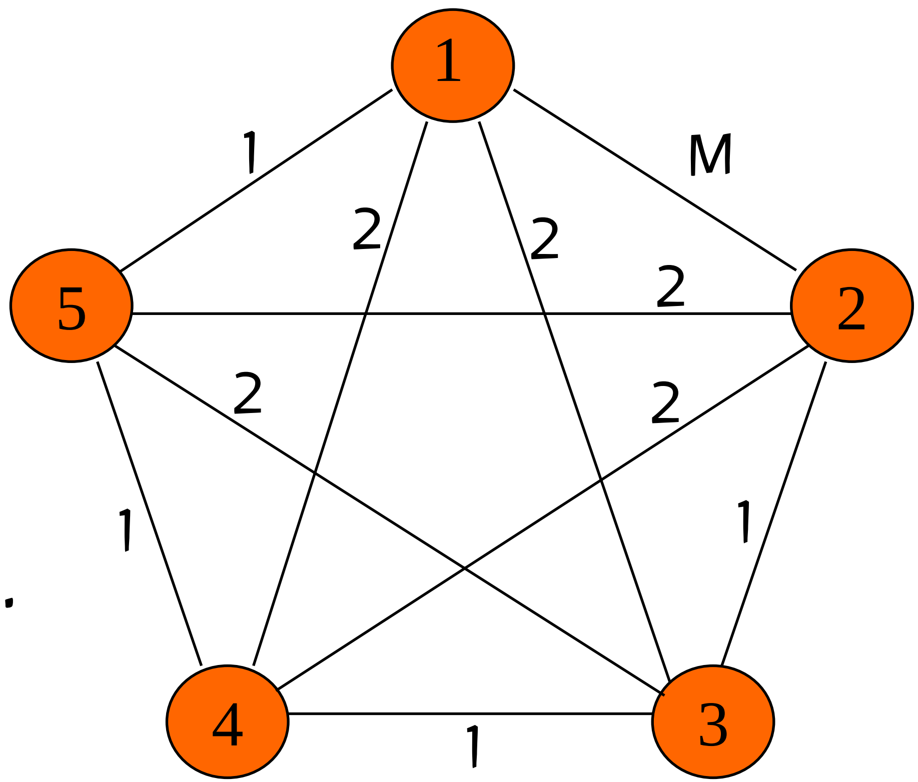
Problema do caixeiro viajante

- Podem encontrar soluções arbitrariamente ruins:

Heurística (1-5-4-3-2-1): M+4

Ótimo (1-5-2-3-4-1): 7

A solução ótima é encontrada se o algoritmo começa do nó 5.



Problema do caixeiro viajante

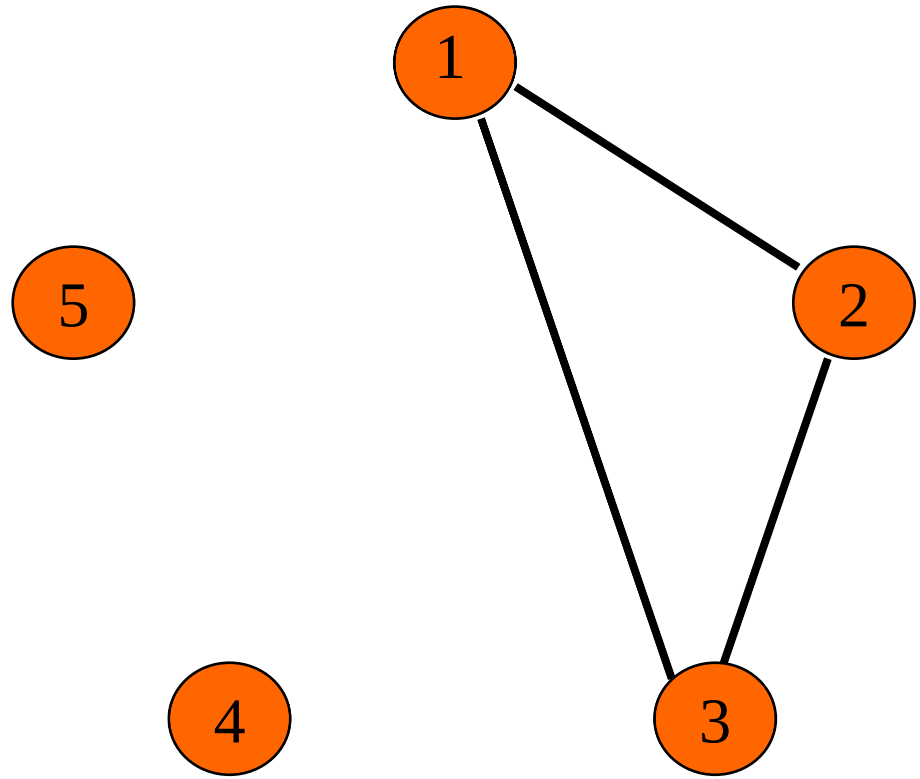
- Extensões e melhorias simples:
  - Repetir a aplicação do algoritmo a partir de cada nó do grafo e selecionar a melhor solução obtida.
    - Melhores soluções, mas tempo de processamento multiplicado por n.
  - A cada iteração selecionar a aresta mais curta a partir de alguma das extremidades em aberto do circuito, e não apenas a partir da última extremidade inserida: solução de comprimento 15 (tempos multiplicados por dois).
  - Critérios mais elaborados para (1) seleção do novo nó incorporado ao circuito a cada iteração e para (2) seleção da posição onde ele entra no circuito: algoritmo baseado no crescimento de um circuito até completá-lo.

Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração:
  - Selecionar o nó k fora do circuito parcial corrente, cuja aresta de menor comprimento que o liga a este circuito parcial corrente é mínima $\rightarrow$ algoritmo de inserção mais próxima
  - Selecionar o nó k fora do circuito parcial corrente, cuja aresta de menor comprimento que o liga a este circuito parcial corrente é máxima $\rightarrow$ algoritmo de inserção mais afastada

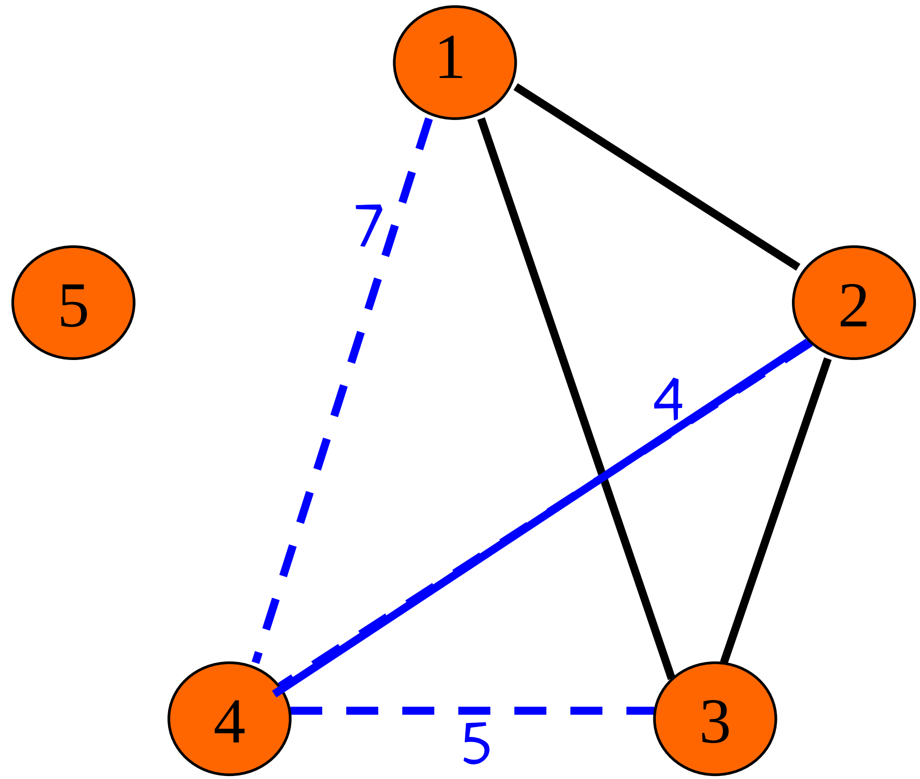
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais próxima
- Seleção do nó:



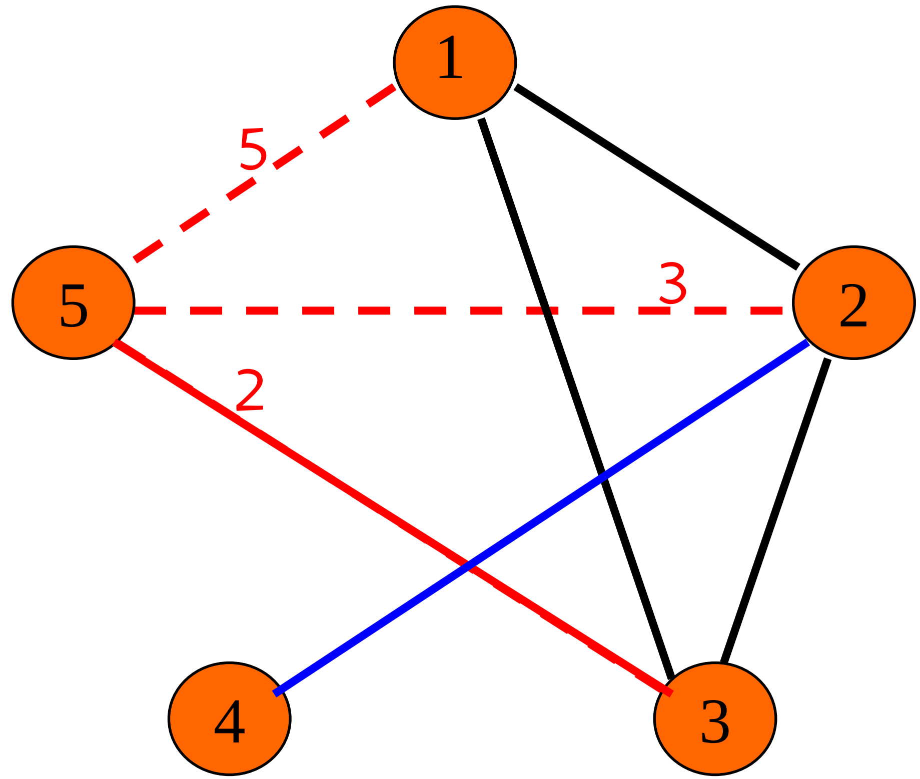
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais próxima
- Seleção do nó:
  - Distância mínima do nó 4: 4



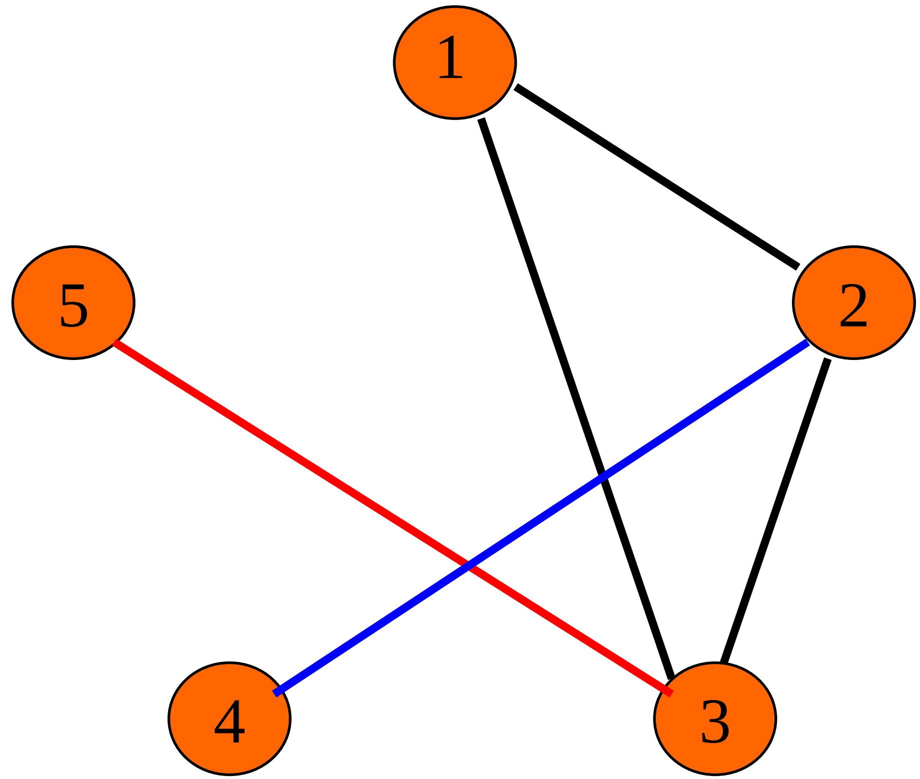
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais próxima
- Seleção do nó:
  - Distância mínima do nó 4: 4
  - Distância mínima do nó 5: 2



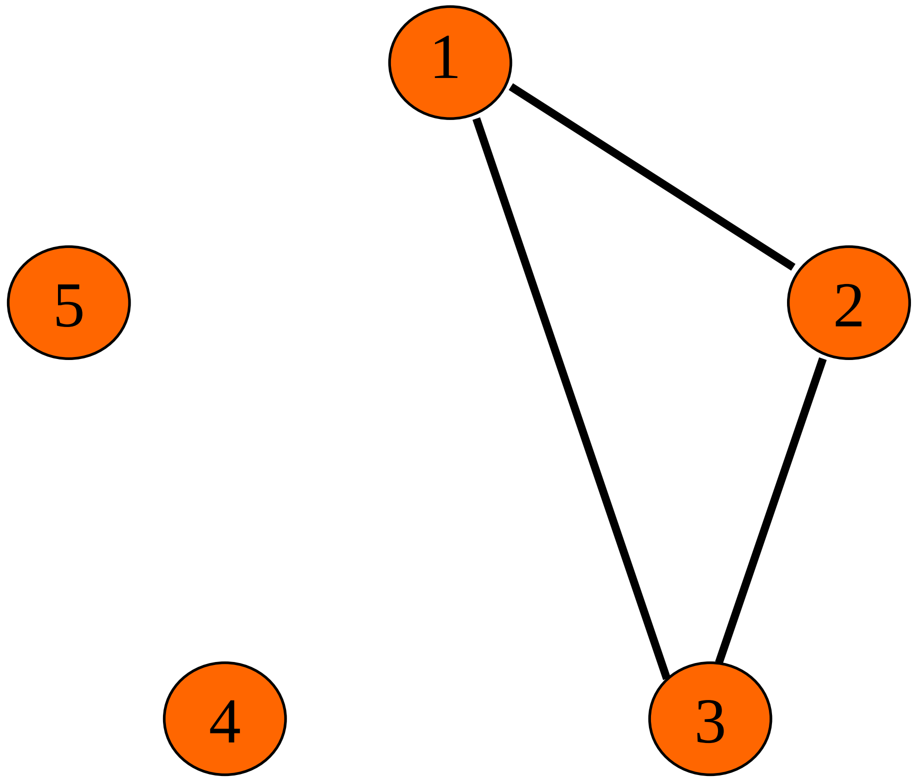
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais próxima
- Seleção do nó:
  - Distância mínima do nó 4: 4
  - Distância mínima do nó 5: 2
  - Nó selecionado: 5



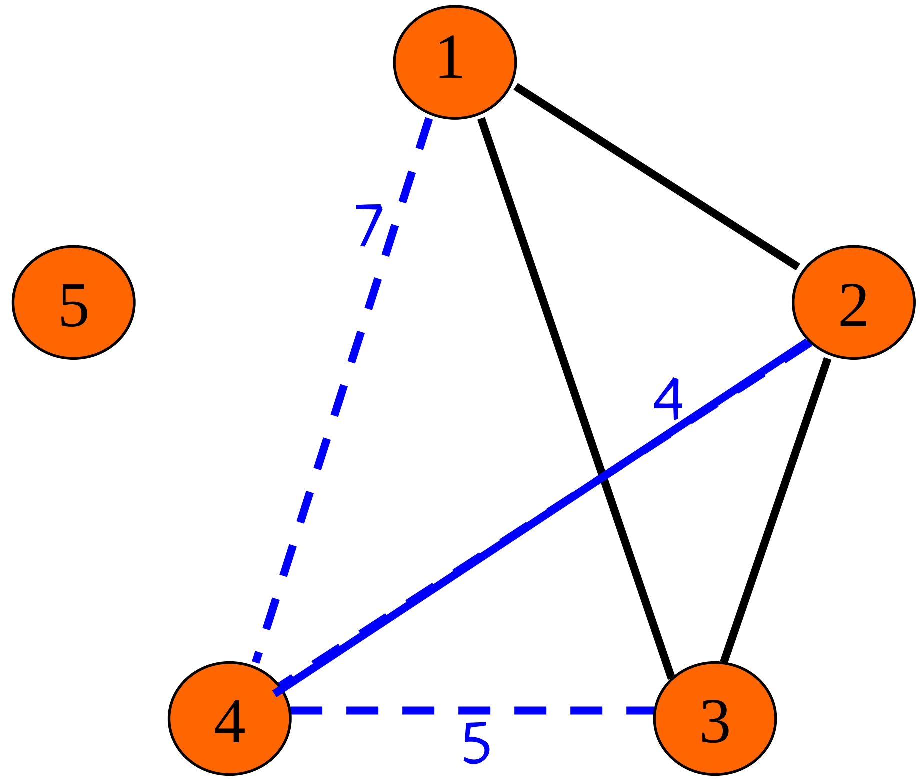
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais afastada
- Seleção do nó:



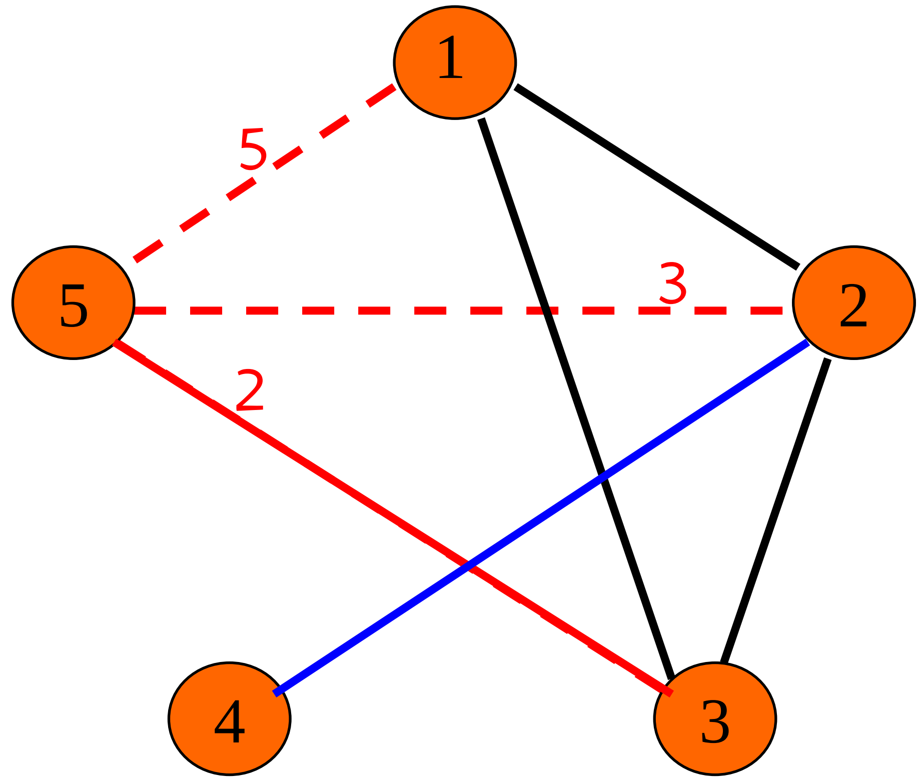
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais afastada
- Seleção do nó:
  - Distância mínima do nó 4: 4



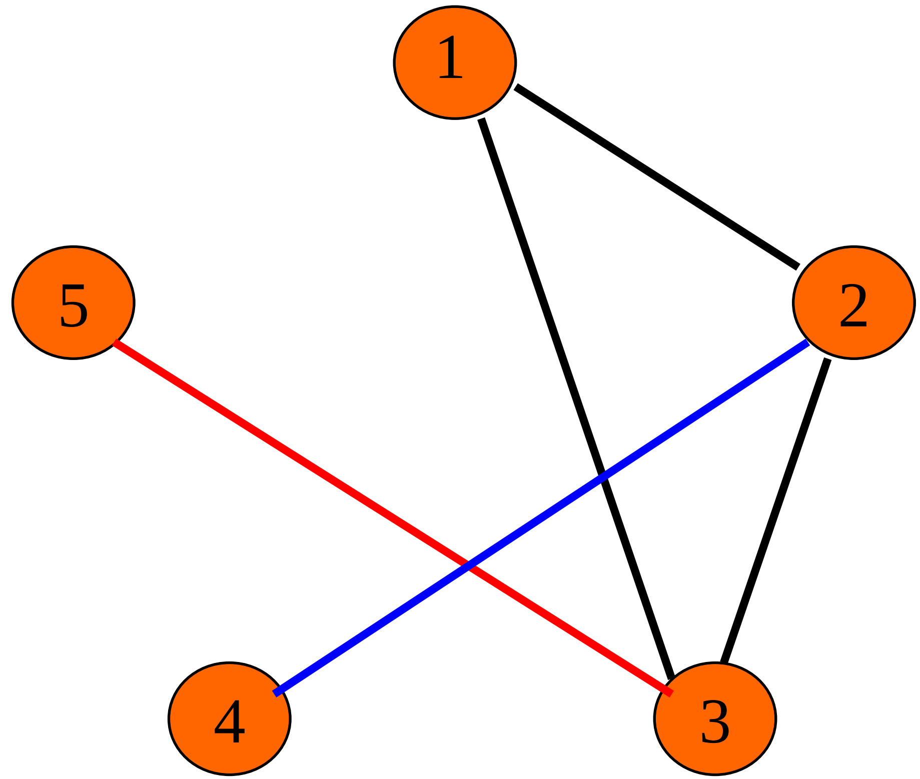
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais afastada
- Seleção do nó:
  - Distância mínima do nó 4: 4
  - Distância mínima do nó 5: 2



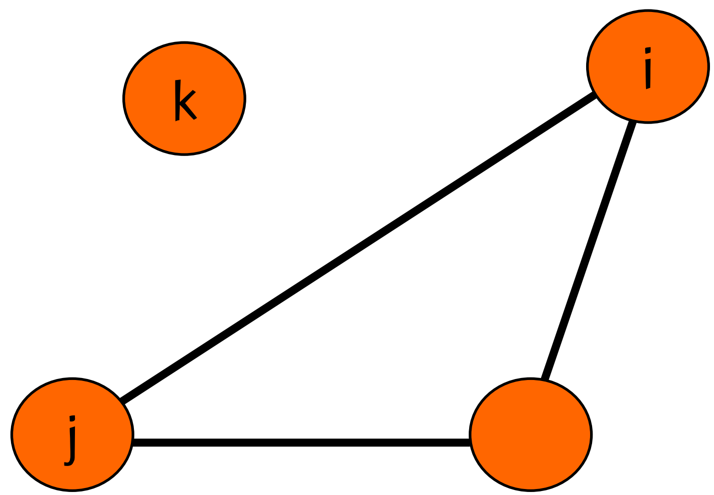
Problema do caixeiro viajante

- Escolha do novo nó a ser incorporado ao circuito a cada iteração pela inserção mais afastada
- Seleção do nó:
  - Distância mínima do nó 4: 4
  - Distância mínima do nó 5: 2
  - Nó selecionado: 4



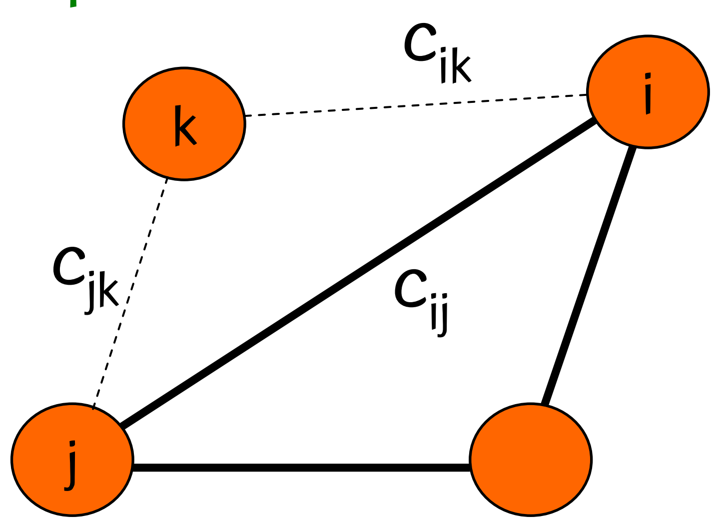
Problema do caixeiro viajante

- Escolha da posição onde o nó selecionado entra no circuito:
  - Suponha a entrada do nó k entre o nó i e o nó j.
  - Devem ser inseridas as areastas (i,k) e (j,k) no circuito parcial corrente, substituindo a aresta (i,j).
  - A variação no comprimento do circuito parcial corrente é dada por $\Delta_{ij}(k) = c_{ik} + c_{jk} - c_{ij}$.



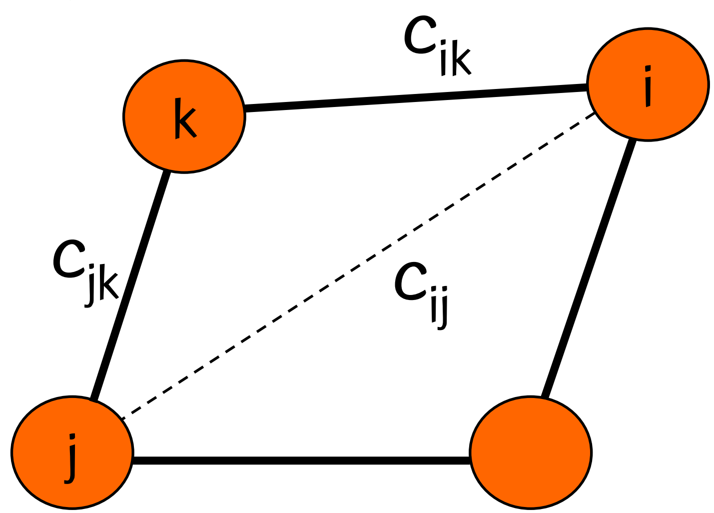
Problema do caixeiro viajante

- Escolha da posição onde o nó selecionado entra no circuito:
  - Suponha a entrada do nó k entre o nó i e o nó j.
  - Devem ser inseridas as areastas (i,k) e (j,k) no circuito parcial corrente, substituindo a aresta (i,j).
  - A variação no comprimento do circuito parcial corrente é dada por $\Delta_{ij}(k) = c_{ik} + c_{jk} - c_{ij}$.



Problema do caixeiro viajante

- Escolha da posição onde o nó selecionado entra no circuito:
  - Suponha a entrada do nó k entre o nó i e o nó j.
  - Devem ser inseridas as areastas (i,k) e (j,k) no circuito parcial corrente, substituindo a aresta (i,j).
  - A variação no comprimento do circuito parcial corrente é dada por $\Delta_{ij}(k) = c_{ik} + c_{jk} - c_{ij}$.



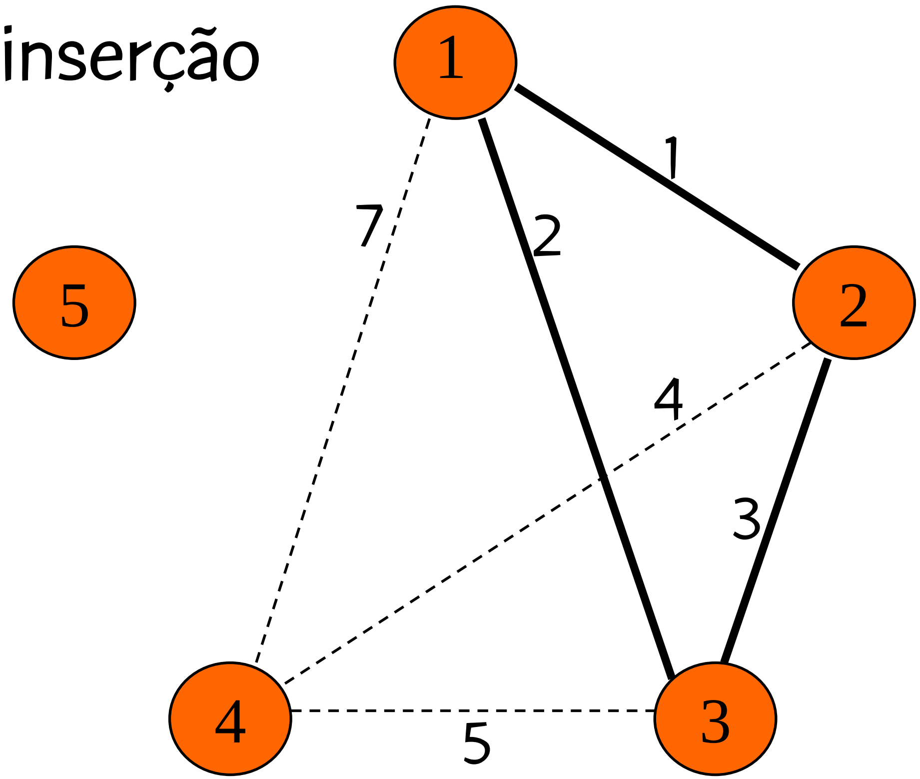
Problema do caixeiro viajante

- Escolha da posição onde o nó selecionado entra no circuito parcial corrente
- Exemplo: nó 4 escolhido para inserção

$\Delta_{12}(4) = 7+4-1 = 10$

$\Delta_{13}(4) = 7+5-2 = 10$

$\Delta_{23}(4) = 5+4-3 = 6$



Problema do caixeiro viajante

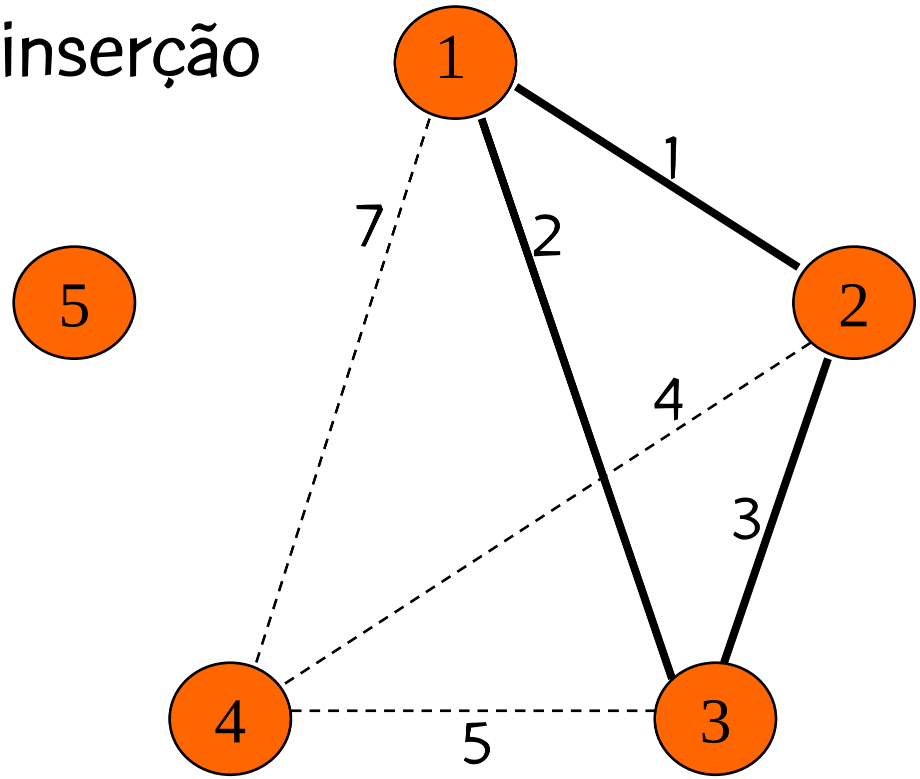
- Escolha da posição onde o nó selecionado entra no circuito parcial corrente
- Exemplo: nó 4 escolhido para inserção

$\Delta_{12}(4) = 7+4-1 = 10$

$\Delta_{13}(4) = 7+5-2 = 10$

$\Delta_{23}(4) = 5+4-3 = 6$

- Inserção mais próxima: entre os nós 2 e 3



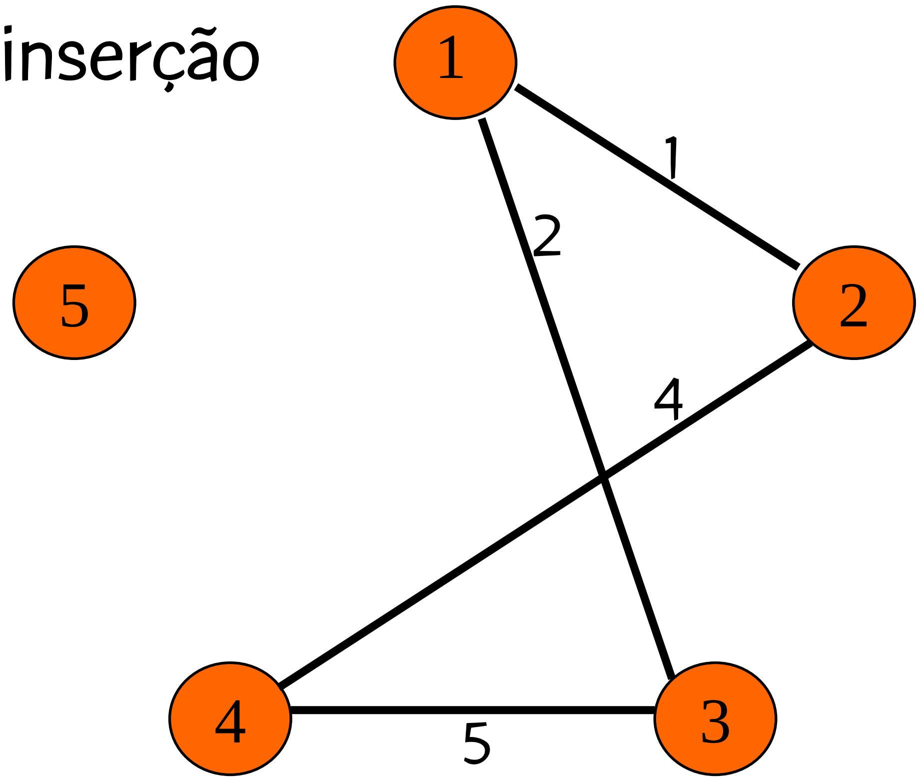
Problema do caixeiro viajante

- Escolha da posição onde o nó selecionado entra no circuito parcial corrente
- Exemplo: nó 4 escolhido para inserção

$\Delta_{12}(4) = 7+4-1 = 10$

$\Delta_{13}(4) = 7+5-2 = 10$

$\Delta_{23}(4) = 5+4-3 = 6$



- Inserção mais próxima: entre os nós 2 e 3

Problema do caixeiro viajante

- Algoritmo de inserção mais próxima:

Escolher o nó inicial i, inicializar um circuito apenas com o nó i e fazer $N \leftarrow N-\{i\}$.

Enquanto $N \neq \emptyset$ fazer:

Encontrar o vértice k fora do circuito corrente cuja aresta de menor comprimento que o liga a ele é mínima.

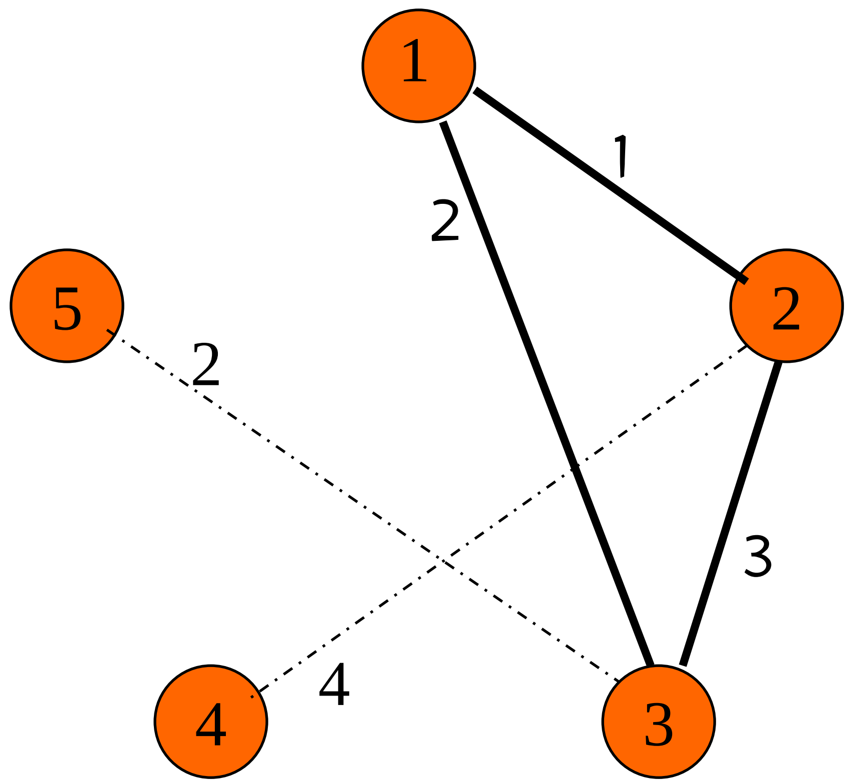
Encontrar o par de arestas (i,k) e (j,k) que ligam o vértice k ao ciclo minimizando $\Delta_{ij}(k) = c_{ik} + c_{jk} - c_{ij}$.

Inserir as arestas (i,k) e (j,k) e retirar a aresta (i,j).

Fazer $N \leftarrow N-\{k\}$.

Fim-enquanto

Problema do caixeiro viajante

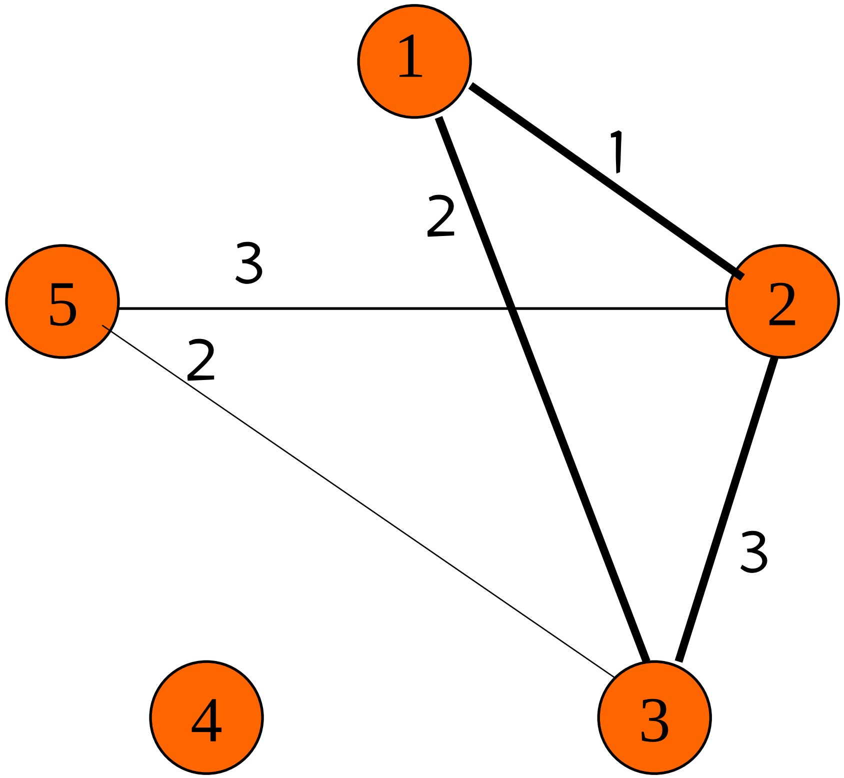


Problema do caixeiro viajante

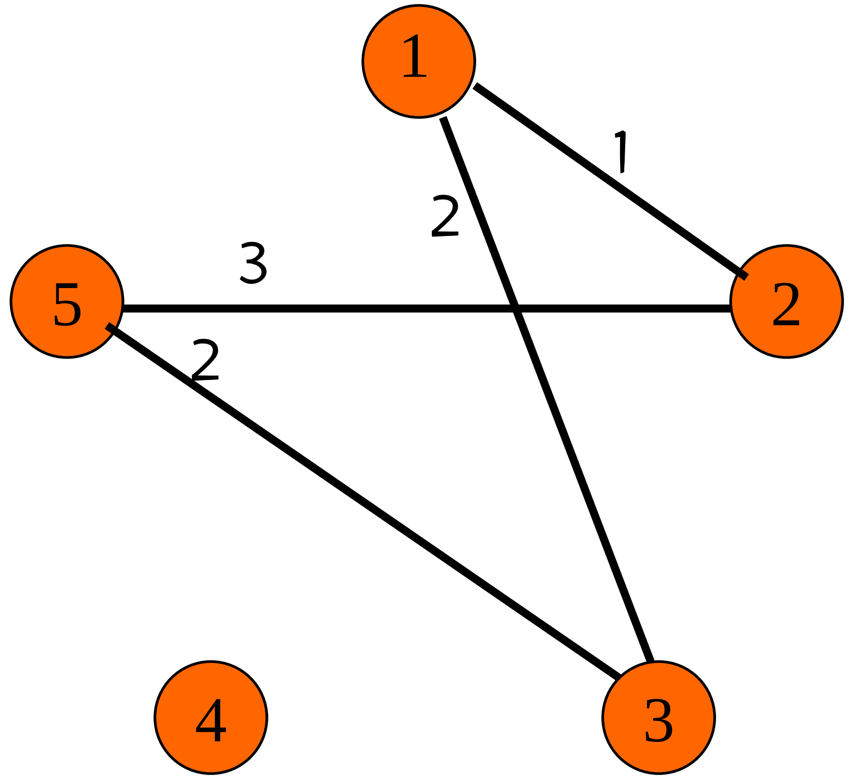
$\Delta_{12}(5) = 5+3-1 = 7$

$\Delta_{23}(5) = 2+3-3 = 2$

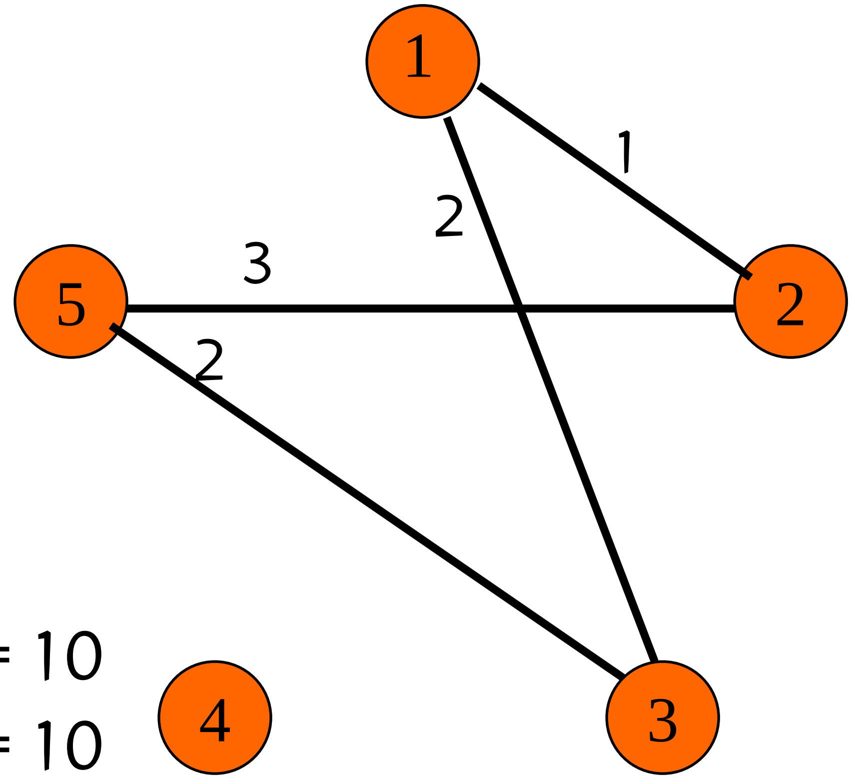
$\Delta_{13}(5) = 2+5-2 = 5$



Problema do caixeiro viajante



Problema do caixeiro viajante



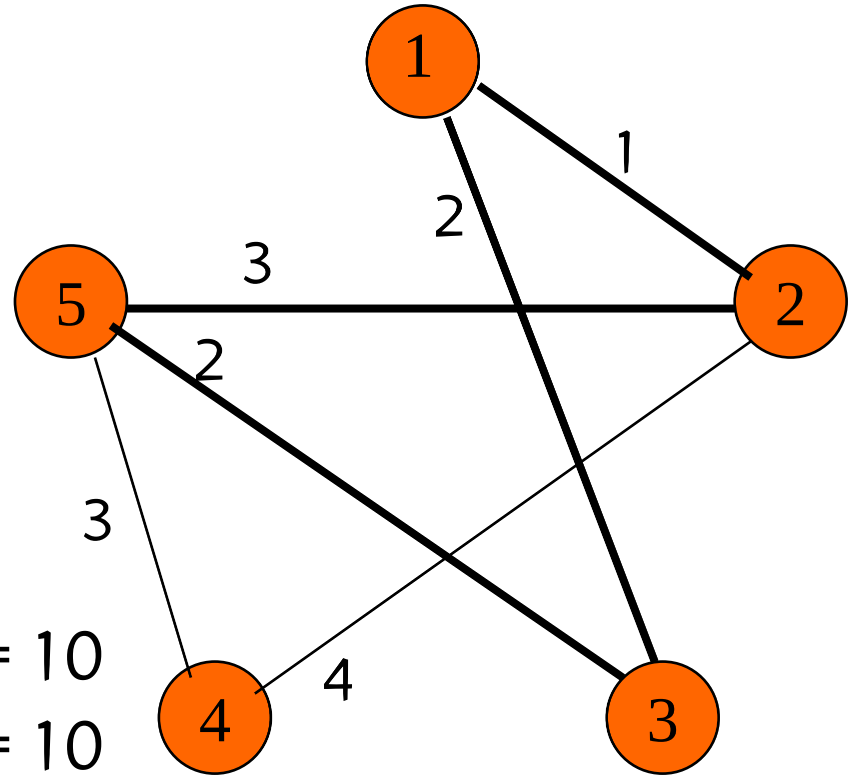
$\Delta_{12}(4) = 7+4-1 = 10$

$\Delta_{13}(4) = 7+5-2 = 10$

$\Delta_{35}(4) = 5+3-2 = 6$

$\color{red}{\Delta_{25}(4) = 4+3-3 = 4}$

Problema do caixeiro viajante



$\Delta_{12}(4) = 7+4-1 = 10$

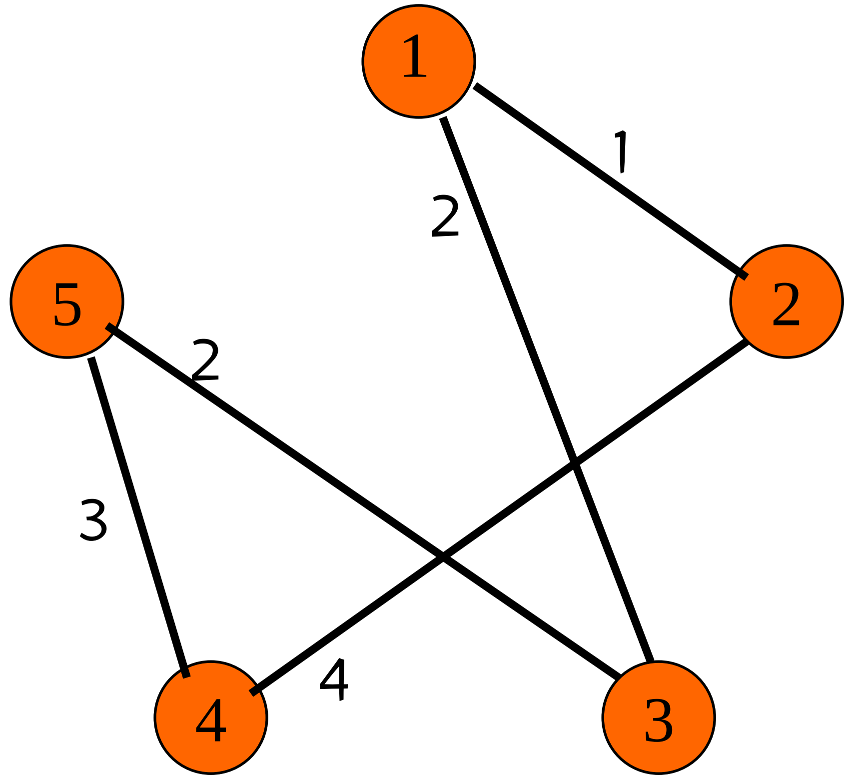
$\Delta_{13}(4) = 7+5-2 = 10$

$\Delta_{35}(4) = 5+3-2 = 6$

$\Delta_{25}(4) = 4+3-3 = 4$

Problema do caixeiro viajante

comprimento = 12



Problema do caixeiro viajante

- Comparação: na prática, o método de inserção mais afastada alcança melhores resultados do que o de inserção mais próxima.
- Melhoria simples: método de inserção mais barata
  - Por que separar em dois passos (1) a seleção do novo nó incorporado ao circuito a cada iteração e (2) a seleção da posição onde ele entra no circuito?
  - Fazer a escolha da melhor combinação em conjunto.
  - Melhores soluções, mas tempos de processamento maiores (cerca de n vezes maiores).

Problema do caixeiro viajante

- Algoritmo de inserção mais barata:

Escolher o nó inicial i, inicializar um circuito apenas com o nó i e fazer $N \leftarrow N-\{i\}$.

Enquanto $N \neq \emptyset$ fazer:

Encontrar o vértice k fora do circuito corrente e o par de arestas (i,k) e (j,k) que ligam o vértice k ao ciclo minimizando $\Delta_{ij}(k) = c_{ik} + c_{jk} - c_{ij}$.

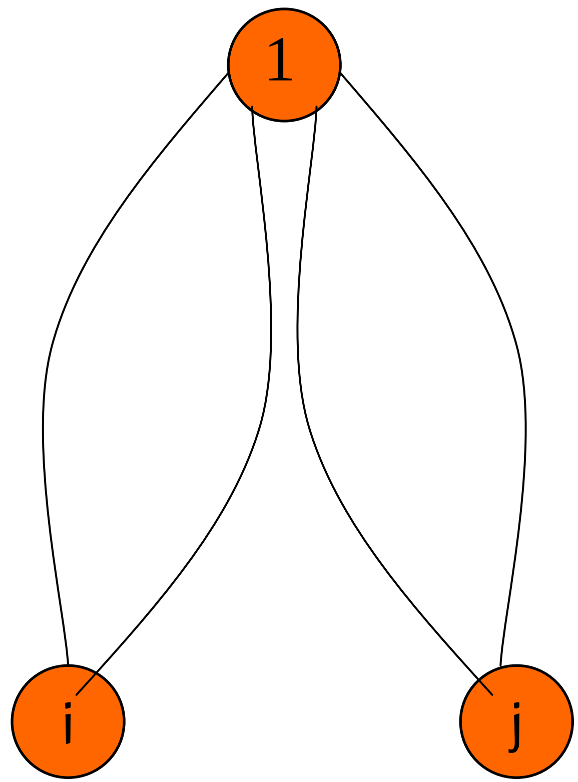
Inserir as arestas (i,k) e (j,k) e retirar a aresta (i,j).

Fazer $N \leftarrow N-\{k\}$.

Fim-enquanto

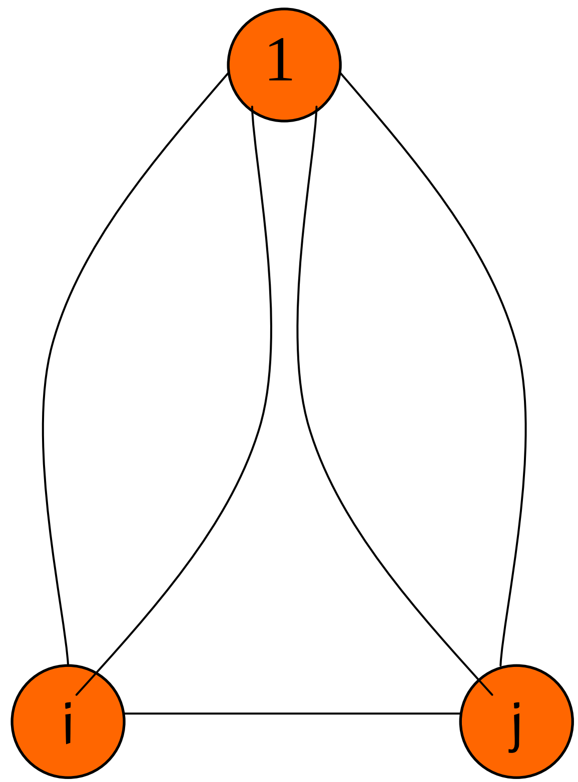
Problema do caixeiro viajante

- Outra idéia diferente: considerar a fusão de subcircuitos
- Considerar dois subcircuitos passando pelo nó 1 e pelos nós i e j.
- Conectá-los diretamente através da aresta (i,j).



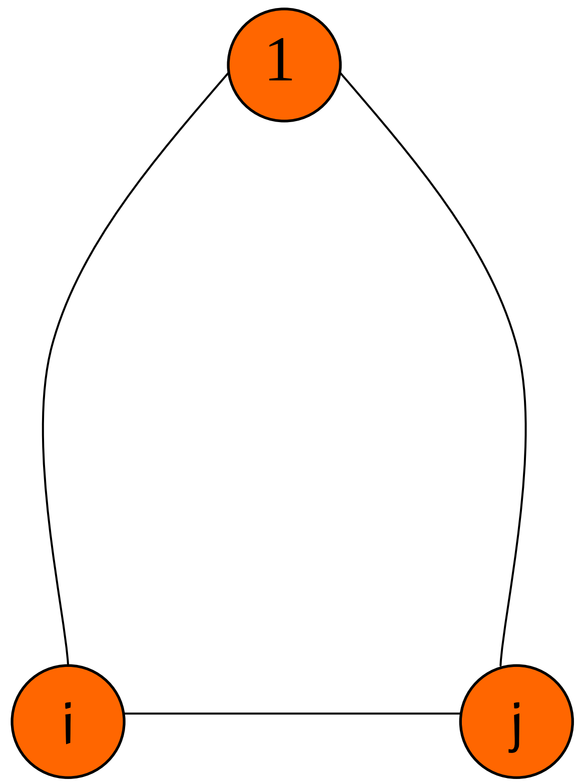
Problema do caixeiro viajante

- Outra idéia diferente: considerar a fusão de subcircuitos
- Considerar dois subcircuitos passando pelo nó 1 e pelos nós i e j.
- Conectá-los diretamente através da aresta (i,j).
- Remover as arestas (1,i) e (1,j).



Problema do caixeiro viajante

- Outra idéia diferente: considerar a fusão de subcircuitos
- Considerar dois subcircuitos passando pelo nó 1 e pelos nós i e j.
- Conectá-los diretamente através da aresta (i,j).
- Remover as arestas (1,i) e (1,j).
- Economia realizada:
  $s_{ij} = c_{1i} + c_{1j} - c_{ij}$



Problema do caixeiro viajante

- Algoritmo das economias:

Escolher um nó inicial i (e.g., i = 1).

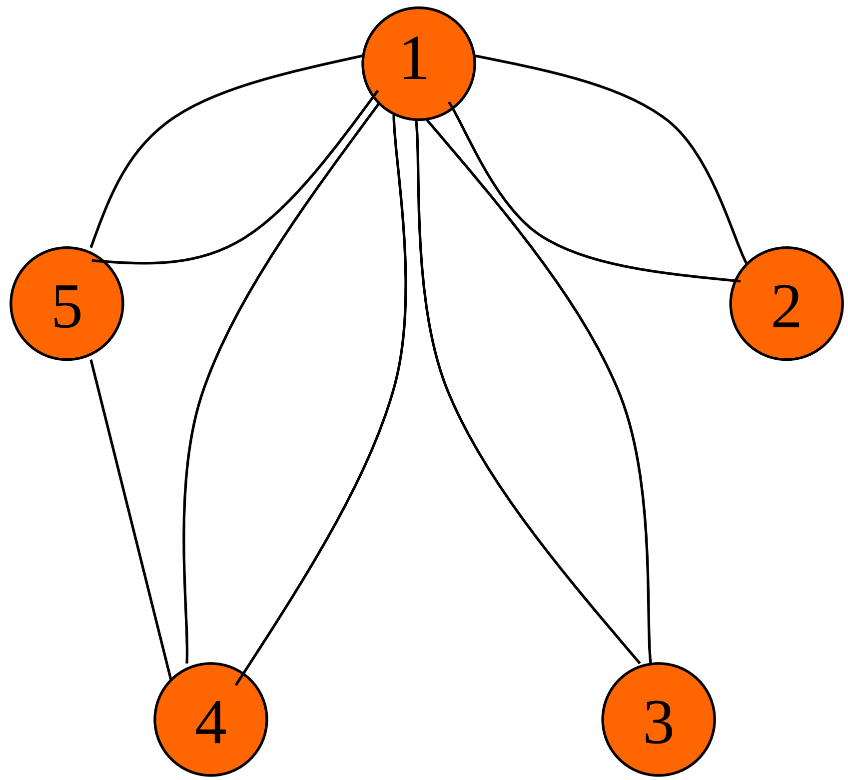
Construir subcircuitos de comprimento 2 envolvendo o nó inicial (e.g., i = 1) e cada um dos demais nós de N.

Calcular as economias $s_{ij} = c_{1i} + c_{1j} - c_{ij}$ obtidas pela fusão dos subcircuitos contendo i e j e ordená-las em ordem decrescente.

Percorrer a lista de economias e fundir os subcircuitos possíveis: a cada iteração, maximizar a distância economizada sobre a solução anterior, combinando-se dois subcircuitos e substituindo-os por uma nova aresta.

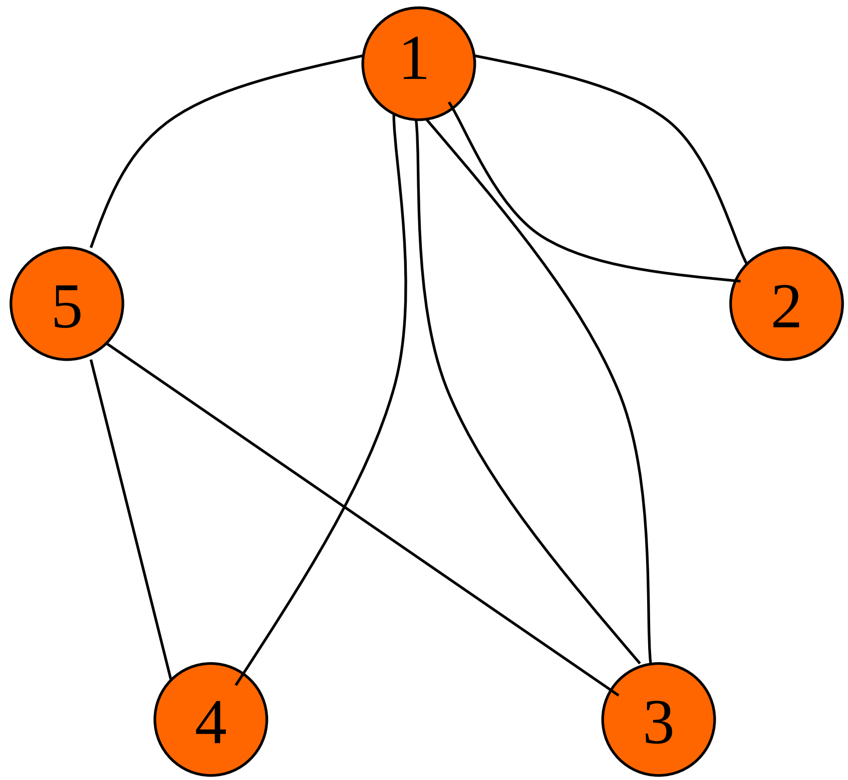
Problema do caixeiro viajante

$s_{45} = 9$
$s_{35} = 5$
$s_{34} = 4$
$s_{24} = 4$
$s_{25} = 3$
$s_{23} = 0$



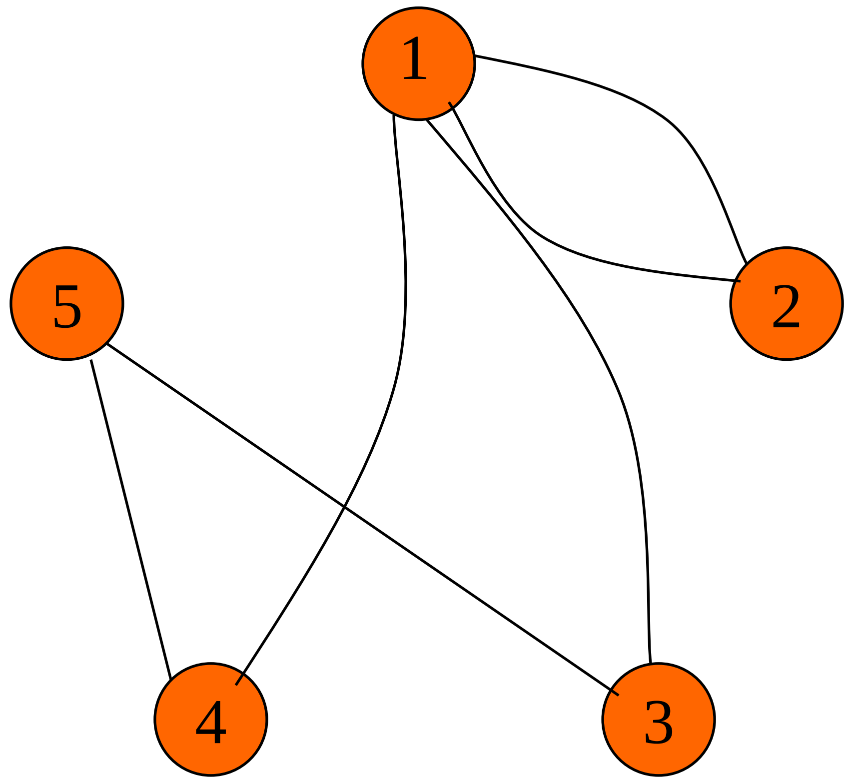
Problema do caixeiro viajante

$s_{45} = 9$
$s_{35} = 5$
$s_{34} = 4$
$s_{24} = 4$
$s_{25} = 3$
$s_{23} = 0$



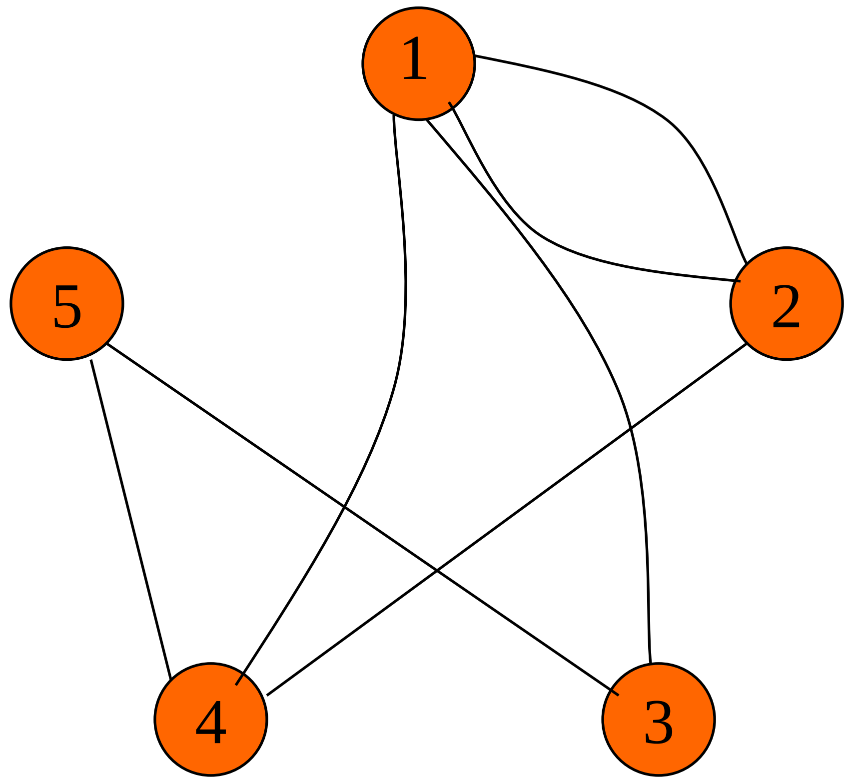
Problema do caixeiro viajante

$s_{45} = 9$
$s_{35} = 5$
$s_{34} = 4$
$s_{24} = 4$
$s_{25} = 3$
$s_{23} = 0$



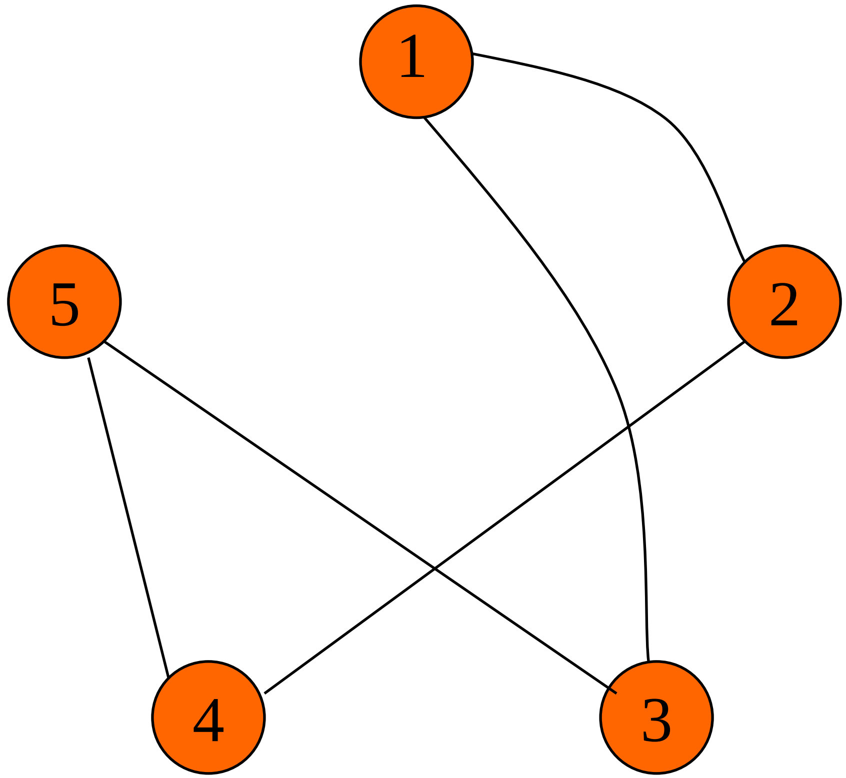
Problema do caixeiro viajante

$s_{45} = 9$
$s_{35} = 5$
$s_{34} = 4$
$s_{24} = 4$
$s_{25} = 3$
$s_{23} = 0$



Problema do caixeiro viajante

$s_{45} = 9$
$s_{35} = 5$
$s_{34} = 4$
$s_{24} = 4$
$s_{25} = 3$
$s_{23} = 0$



comprimento = 12

Problema de Steiner em grafos

- Grafo não-orientado $G=(V,E)$

  V: vértices

  E: arestas

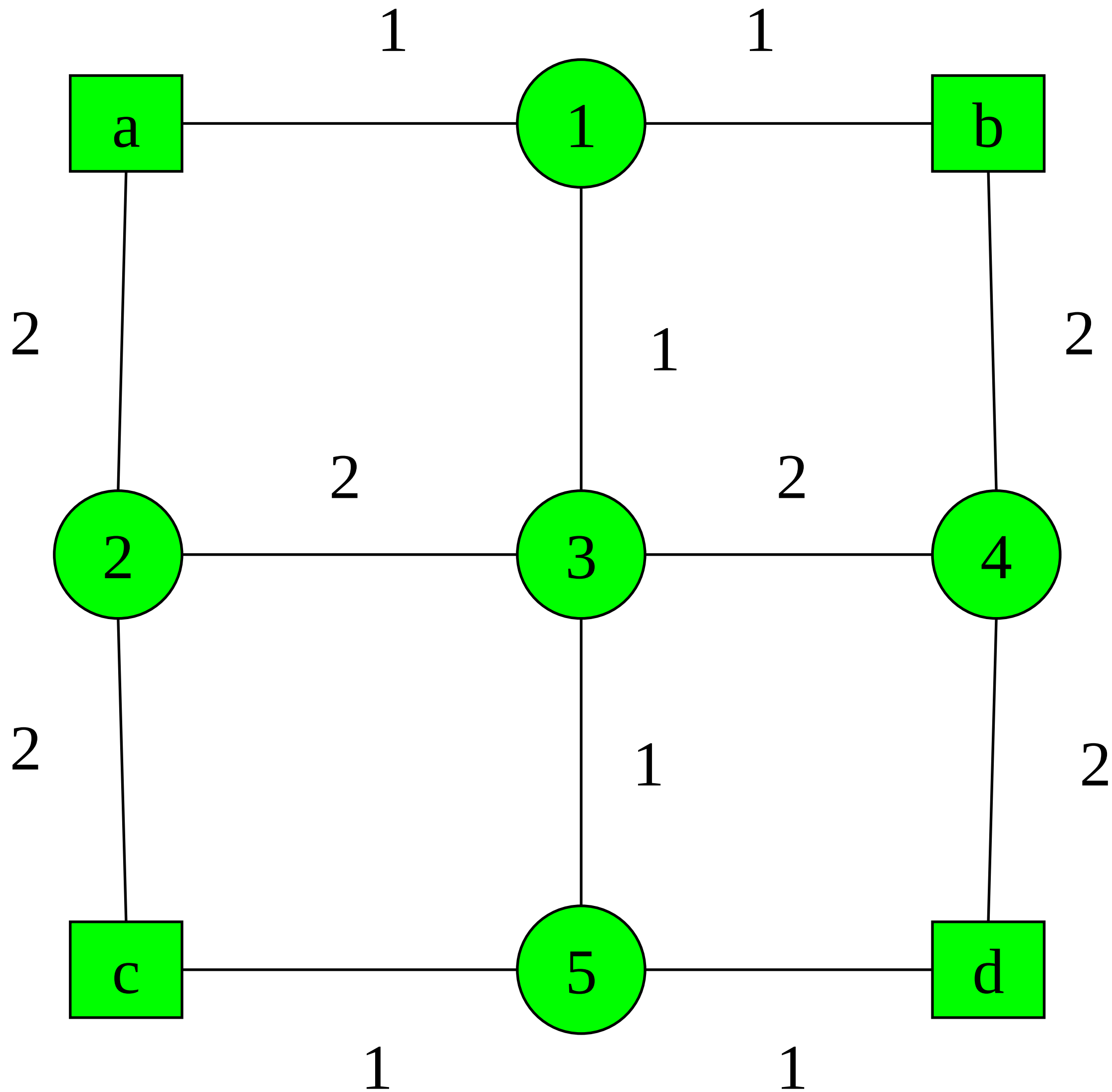
  T: vértices terminais (obrigatórios)

  c_e: peso (positivo) da aresta $e \in E$
- Problema: conectar os nós terminais com custo (peso) mínimo, eventualmente utilizando os demais nós como passagem.

Problema de Steiner em grafos

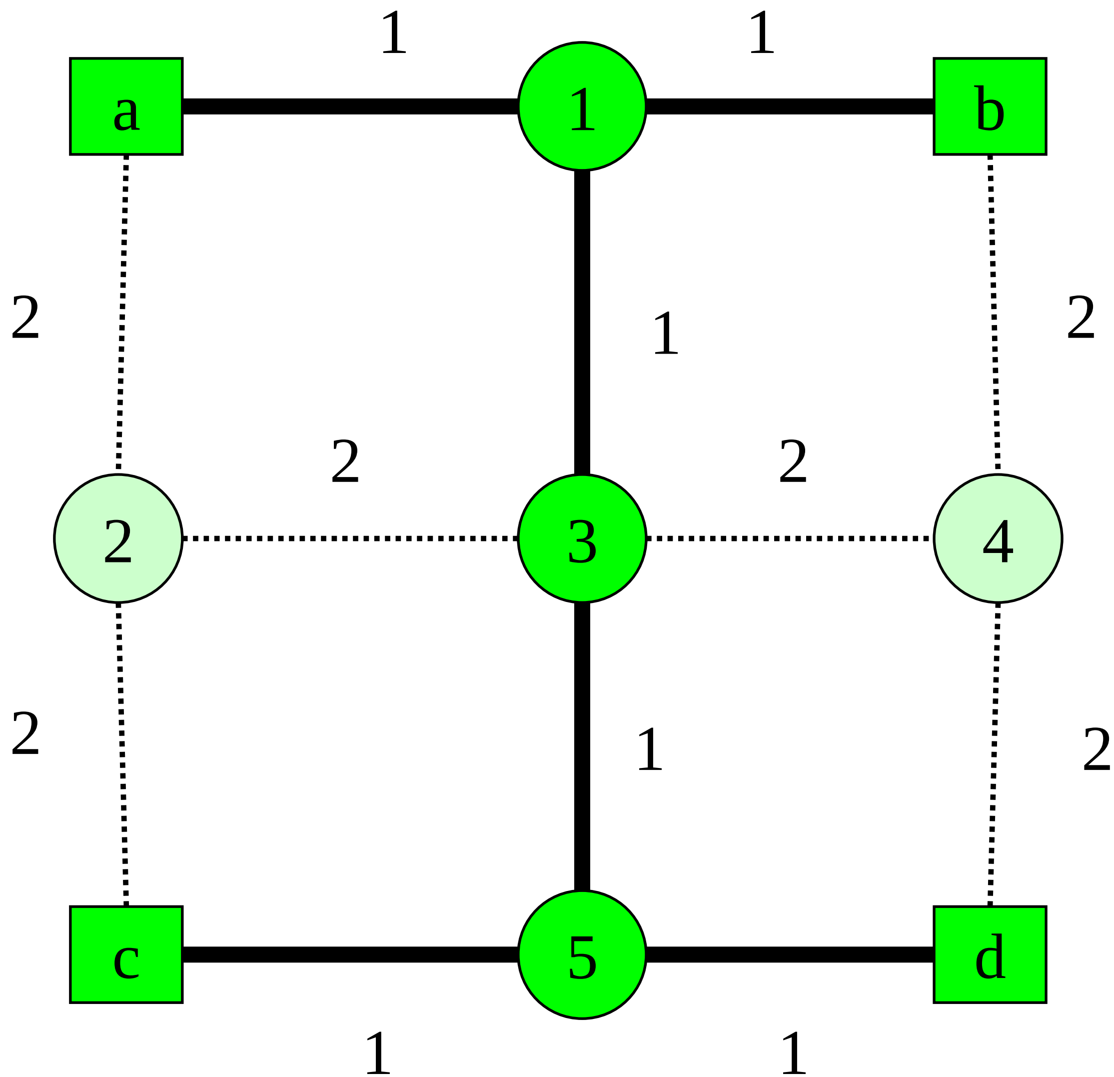
- Vértices de Steiner: vértices opcionais que fazem parte da solução ótima
- Aplicações: projeto de redes de computadores (conectar um conjunto de clientes através de concentradores, cujos locais devem ser determinados), redes de telecomunicações, problema da filogenia em biologia, etc.

Problema de Steiner em grafos



Problema de Steiner em grafos

Mínimo: 6



Problema de Steiner em grafos

- Heurística da rede de distâncias:

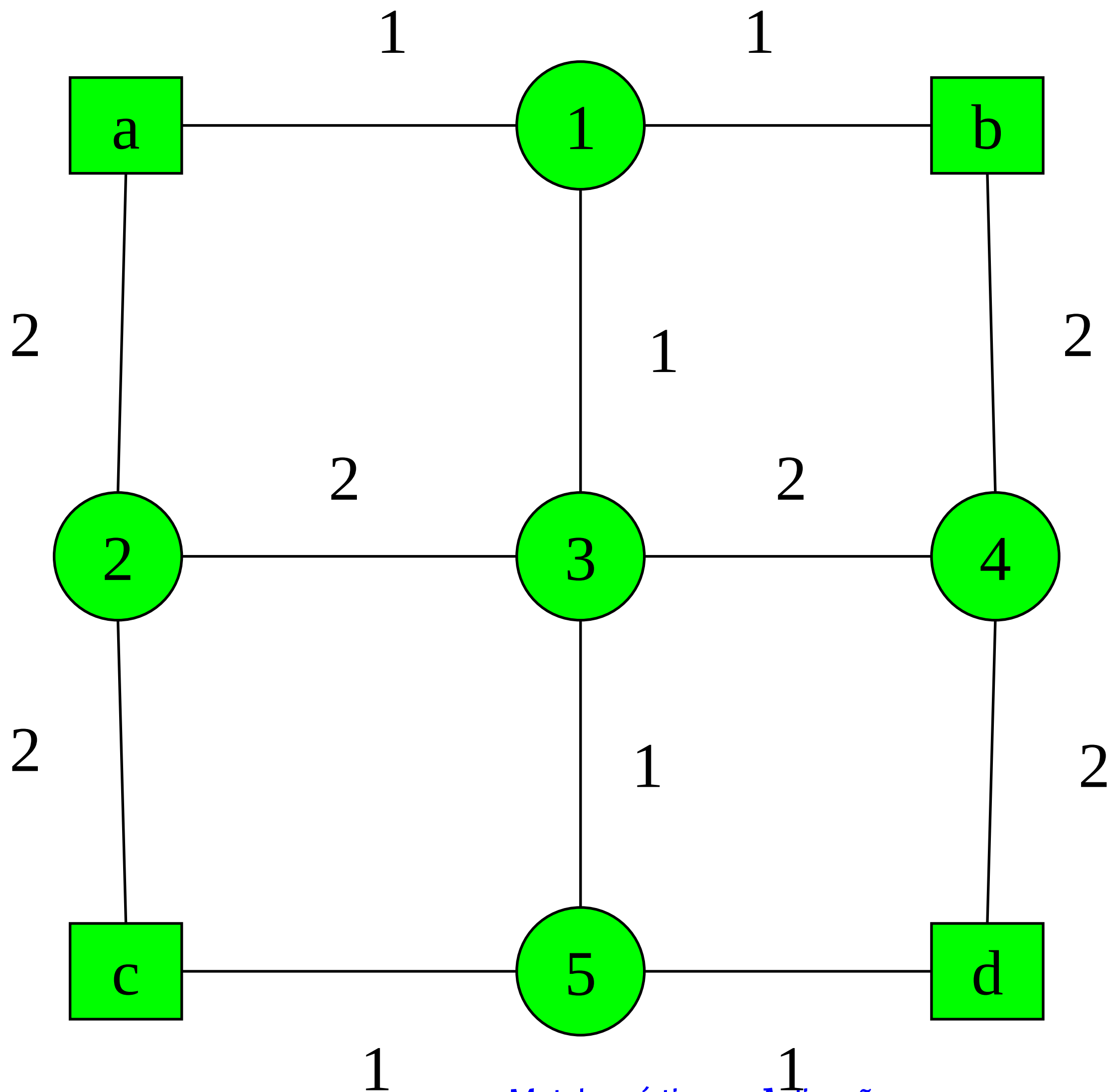
Calcular os caminhos mais curtos entre cada par de terminais do grafo.

Criar a rede de distâncias formada pelos nós obrigatórios e pelas arestas correspondentes aos caminhos mais curtos.

Obter a árvore geradora de peso mínimo dos nós da rede de distâncias.

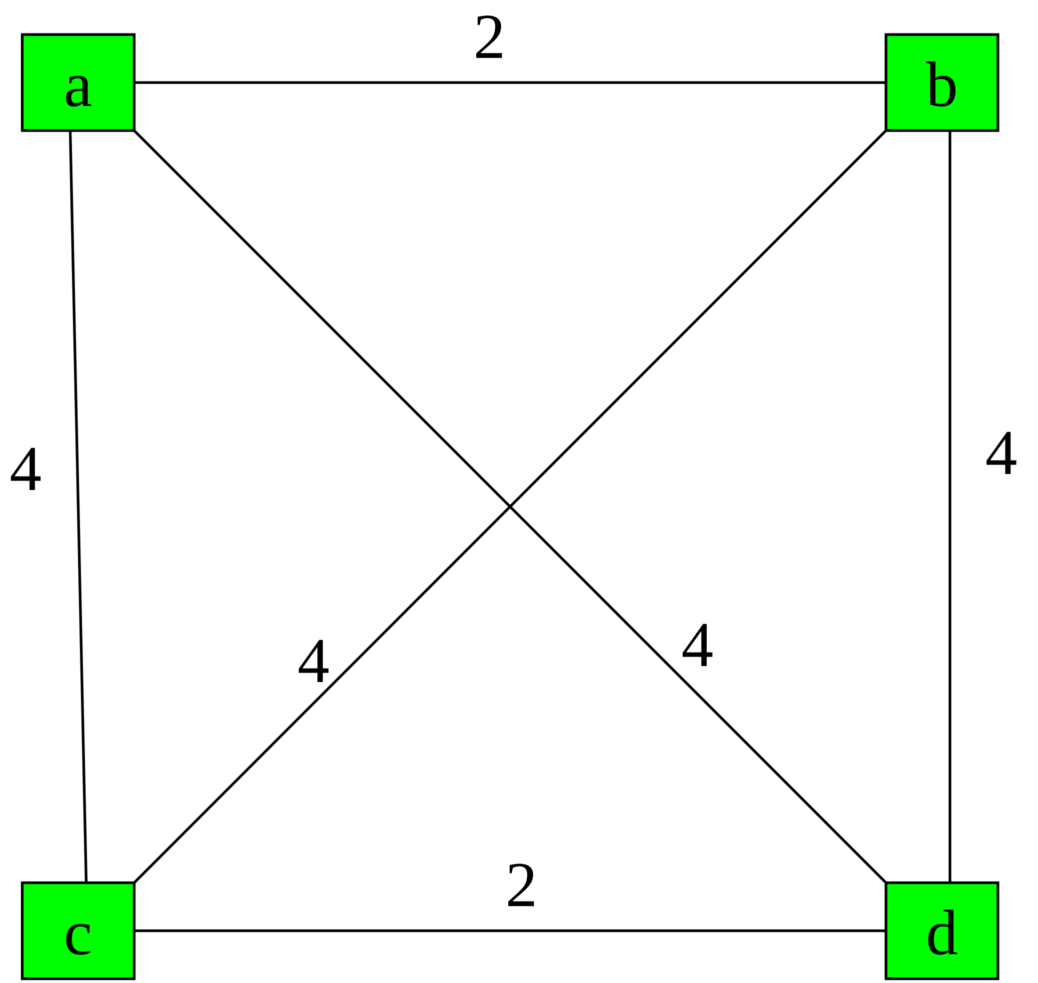
Expandir as arestas da árvore geradora.

Problema de Steiner em grafos



C_{ab}: a,1,b (2)
C_{ac}: a,2,c (4)
C_{ad}: a,1,3,5,d (4)
C_{bc}: b,1,3,5,c (4)
C_{bd}: b,4,d (4)
C_{cd}: c,5,d (2)

Problema de Steiner em grafos



C_{ab}: a,1,b (2)

C_{ac}: a,2,c (4)

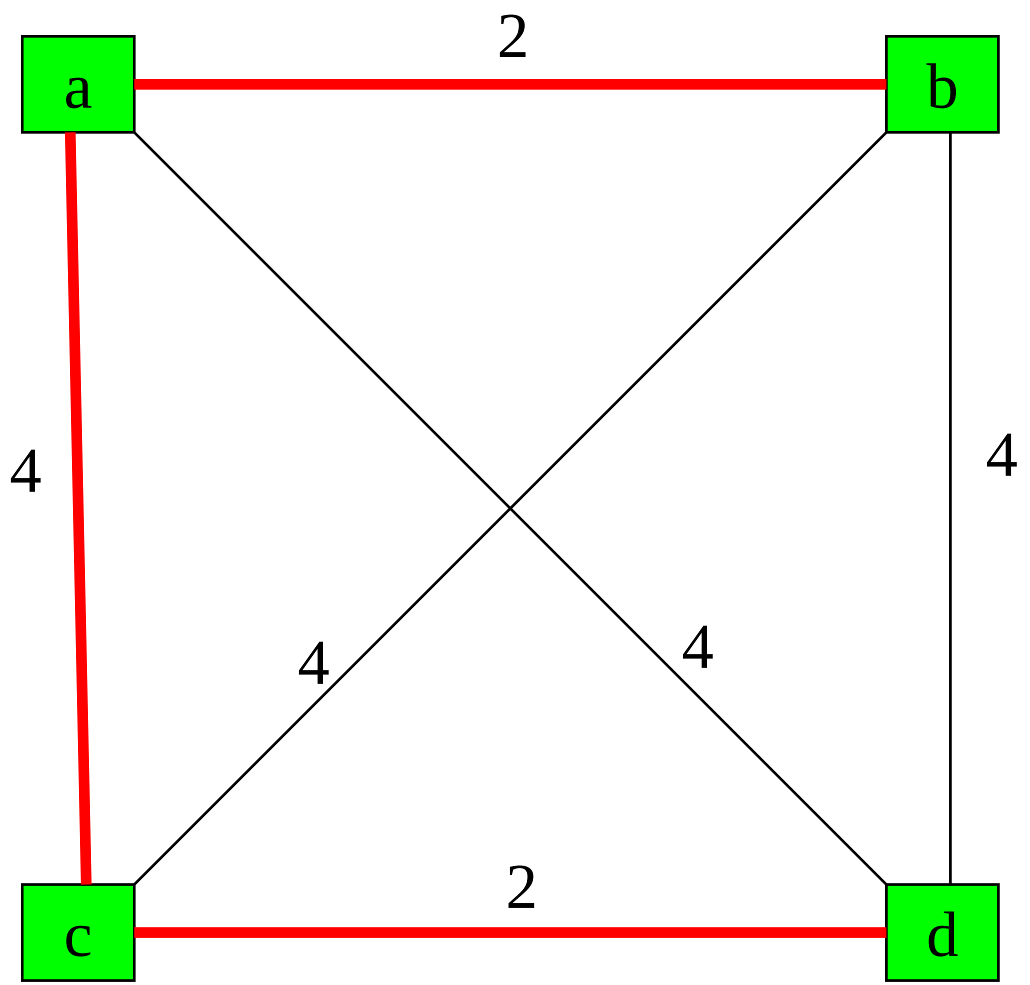
C_{ad}: a,1,3,5,d (4)

C_{bc}: b,1,3,5,c (4)

C_{bd}: b,4,d (4)

C_{cd}: c,5,d (2)

Problema de Steiner em grafos



C_{ab}: a,1,b (2)

C_{ac}: a,2,c (4)

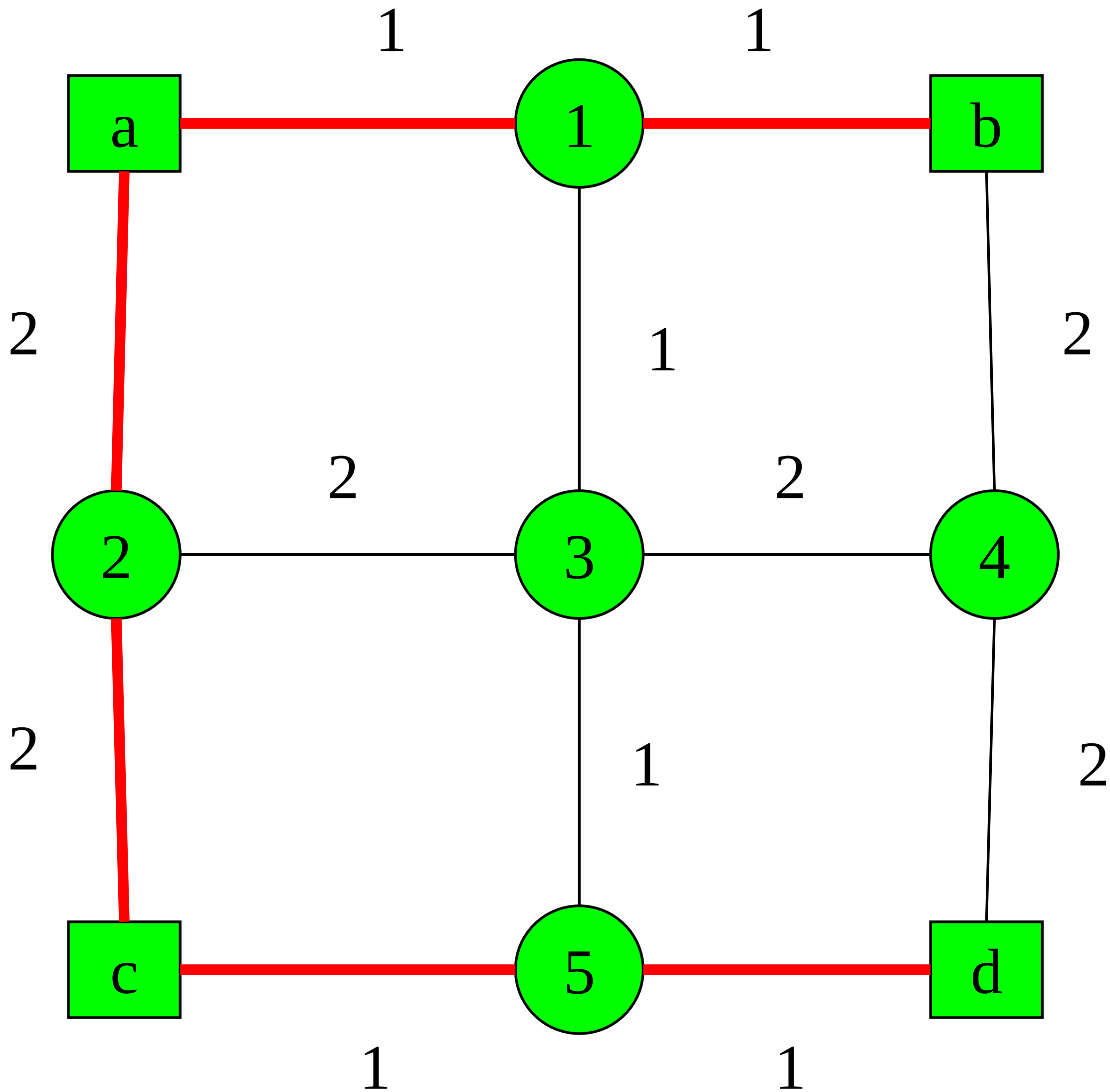
C_{ad}: a,1,3,5,d (4)

C_{bc}: b,1,3,5,c (4)

C_{bd}: b,4,d (4)

C_{cd}: c,5,d (2)

Problema de Steiner em grafos



C_{ab}: a,1,b (2)
C_{ac}: a,2,c (4)
C_{ad}: a,1,3,5,d (4)
C_{bc}: b,1,3,5,c (4)
C_{bd}: b,4,d (4)
C_{cd}: c,5,d (2)

Peso: 8

Problema de Steiner em grafos

- Heurística dos caminhos mais curtos:

Calcular o caminho mais curto de entre cada par de terminais.

Sejam s um nó terminal, Solução $\leftarrow \{s\}$, $S \leftarrow \{s\}$, $k \leftarrow 0$.

Enquanto $S \neq T$ fazer:

  Obter o terminal s mais próximo de Solução e o caminho correspondente C.

  Fazer $S \leftarrow S \cup \{s\}$ e Solução $\leftarrow$ Solução $\cup$ C.

Fim-enquanto

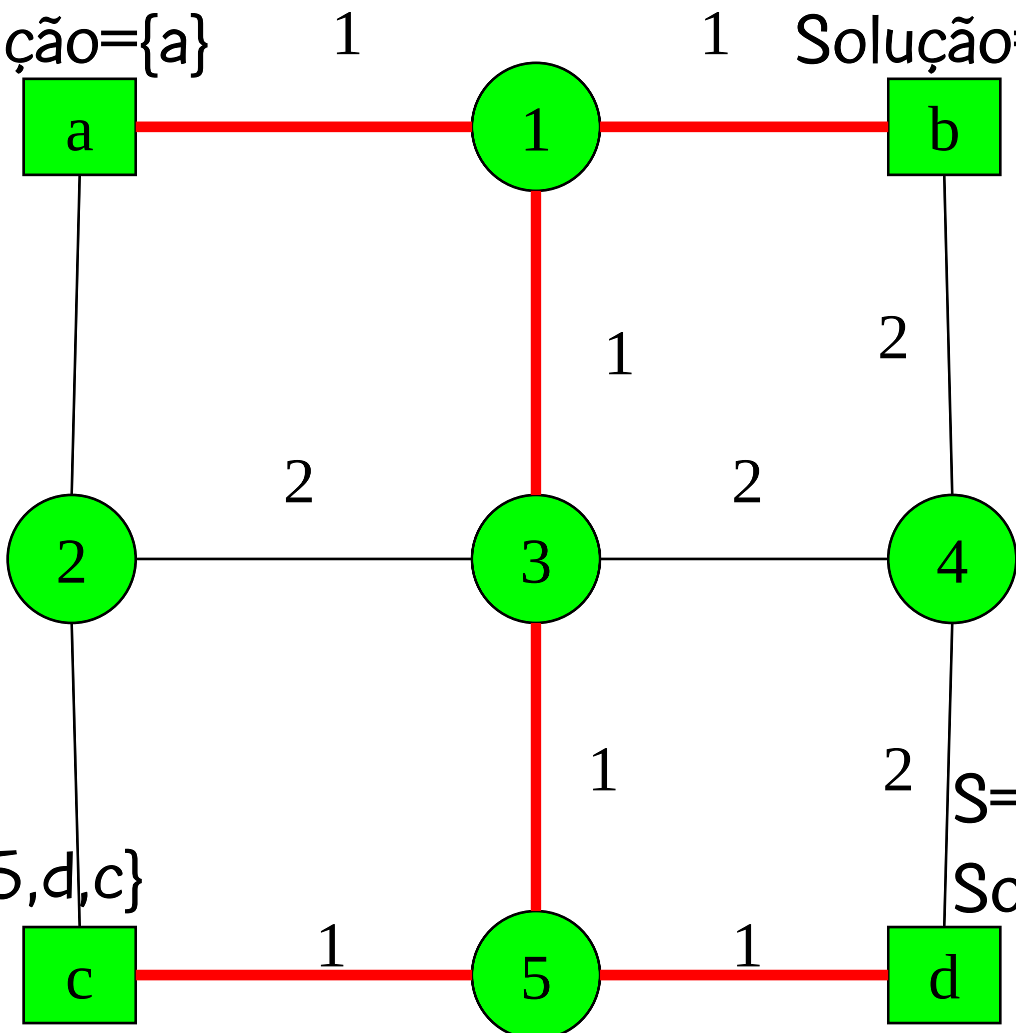
Problema de Steiner em grafos

S={a}

Solução={a}

S={a,b}

Solução={a,1,b}



Peso: 6

S={a,b,d,c}

Solução={a,1,b,3,5,d,c}

S={a,b,d}

Solução={a,1,b,3,5,d}

Algoritmos gulosos

- Algoritmos gulosos:

  A construção de uma solução gulosa consiste em selecionar seqüencialmente o elemento de E que minimiza o incremento no custo da solução parcial mantendo sua viabilidade, terminando quando se obtém uma solução viável (problema de minimização).

  O incremento no custo da solução parcial é chamado de função gulosa.

Algoritmos gulosos

- Algoritmos gulosos:

  Por escolher a cada passo considerando apenas a próxima decisão, chama-se também de algoritmo míope, pois enxerga somente o que está mais próximo.

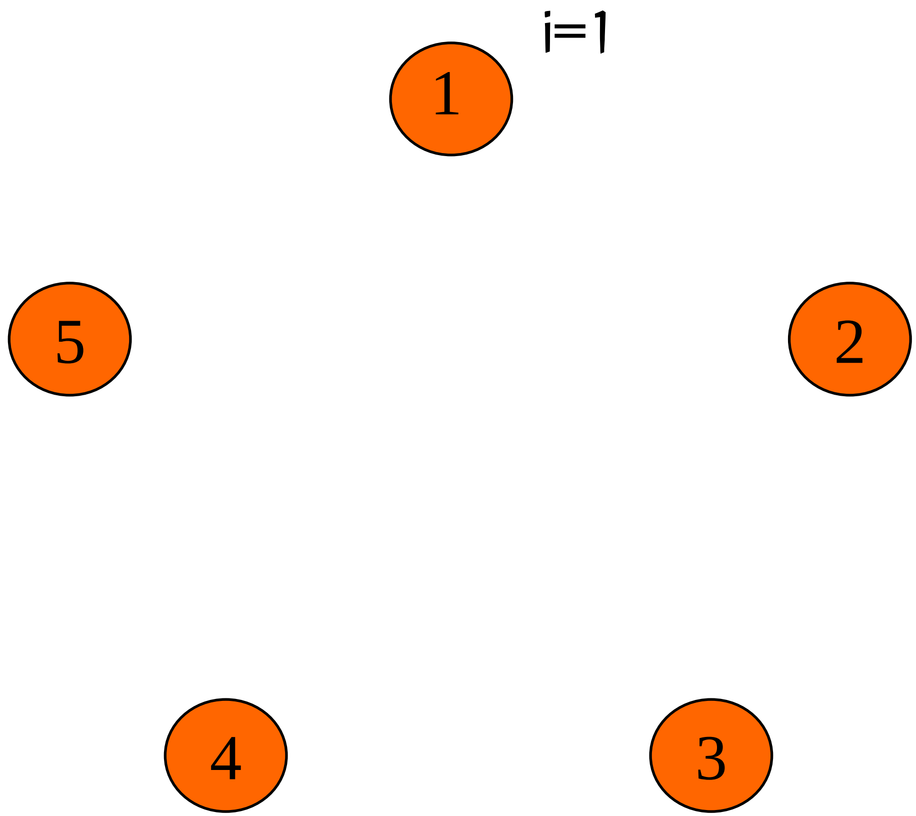
Algoritmos gulosos

- Cada elemento que entra na solução, nela permanece até o final.
- Algoritmo guloso para o problema da árvore geradora de peso mínimo: sempre determina a solução ótima
- Algoritmo guloso para o problema da mochila:
  - Ordenar os itens em ordem decrescente da razão c_j/a_j.
  - Selecionar os itens que cabem na mochila segundo esta ordem.
- Algoritmo do vizinho mais próximo para o PCV
- Cuidado: nem sempre encontram a solução ótima exata, são portanto heurísticas para estes problemas!

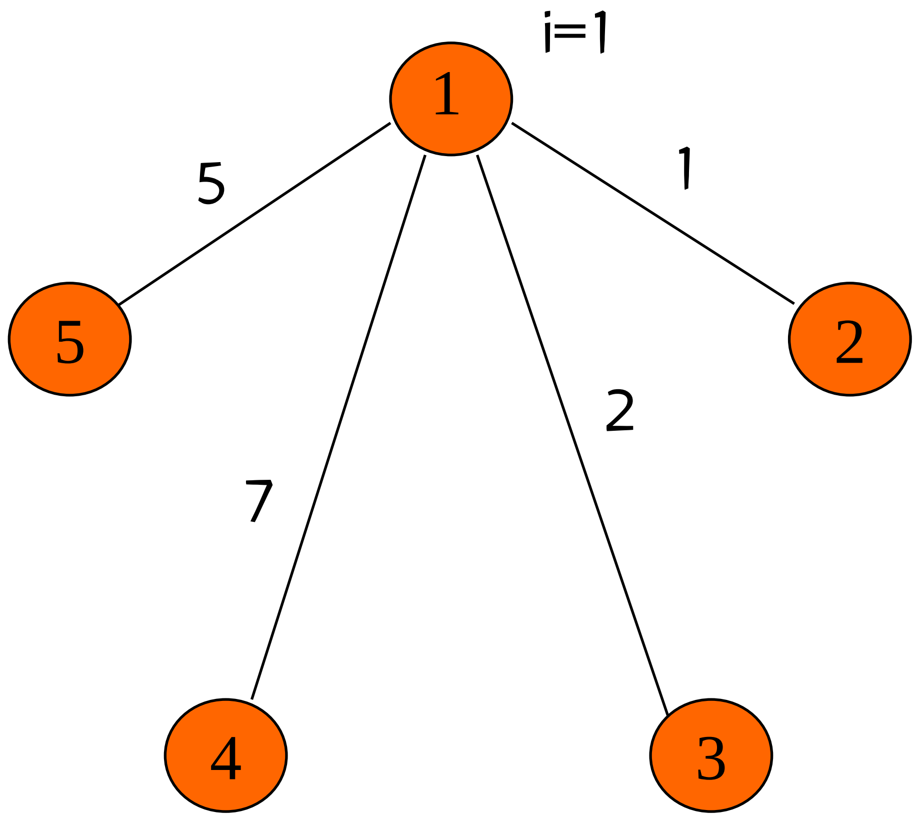
Algoritmos gulosos randomizados

- Um algoritmo guloso encontra sempre a mesma solução para um dado problema.
- Randomização das escolhas gulosas permite alcançar diversidade nas soluções encontradas, se o algoritmo for aplicado diversas vezes.
- Algoritmo guloso randomizado:
  - Criar uma lista de candidatos a cada iteração com os melhores elementos ainda não selecionados e fazer uma escolha aleatória entre eles.
- Aplicar o algoritmo repetidas vezes, obtendo soluções diferentes a cada aplicação e salvando a melhor.

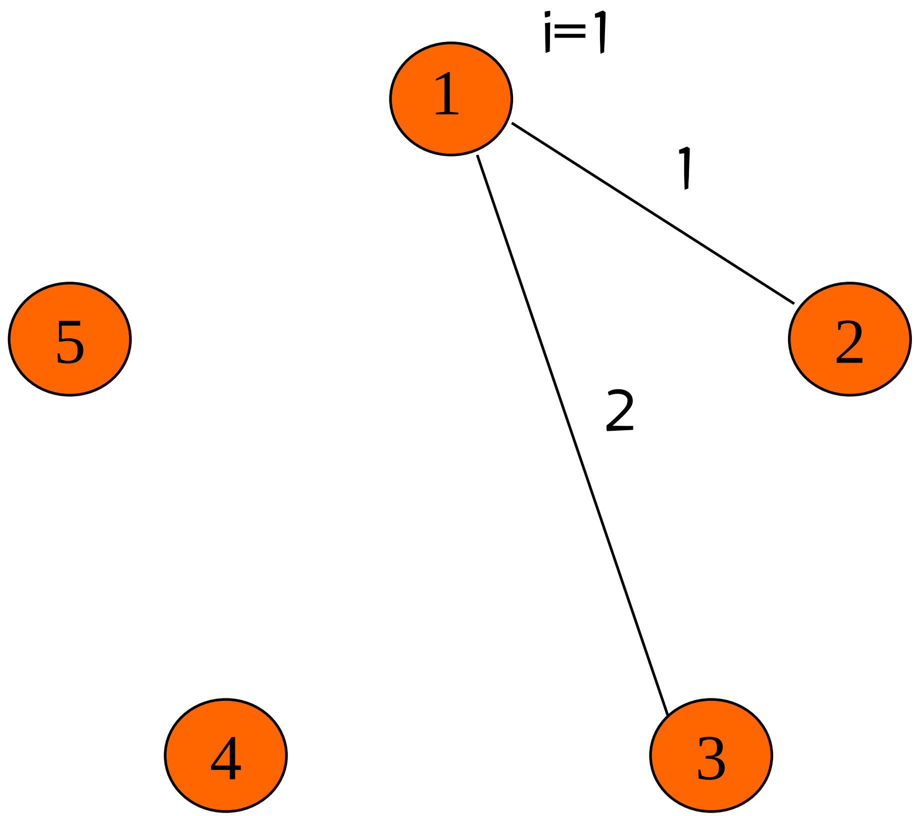
Problema do caixeiro viajante



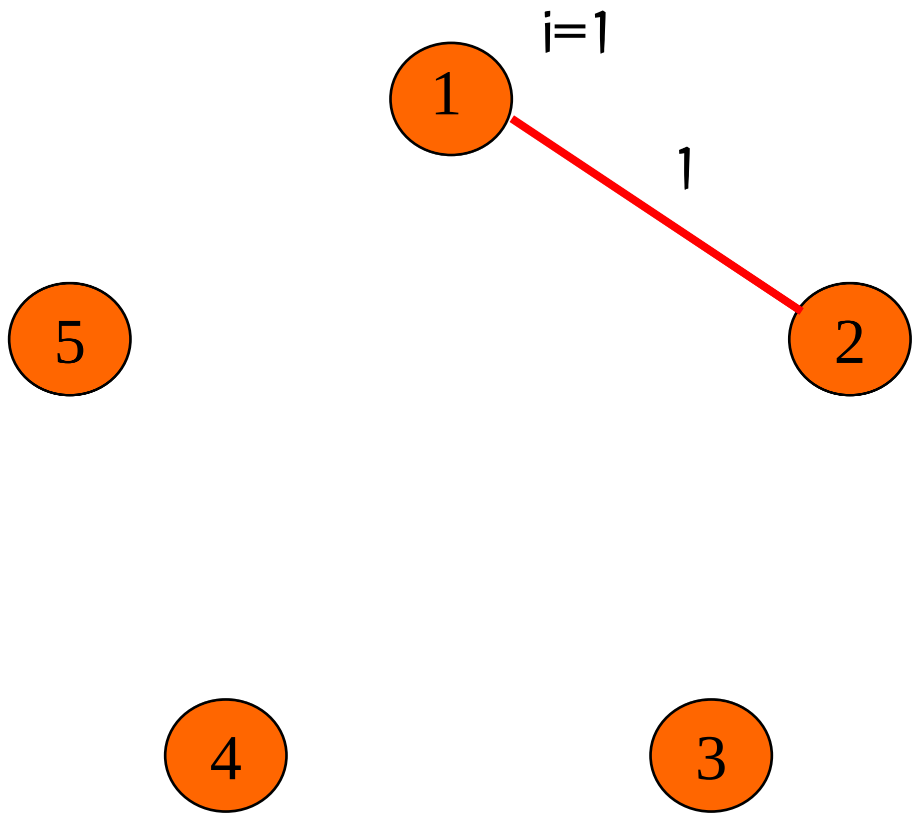
Problema do caixeiro viajante



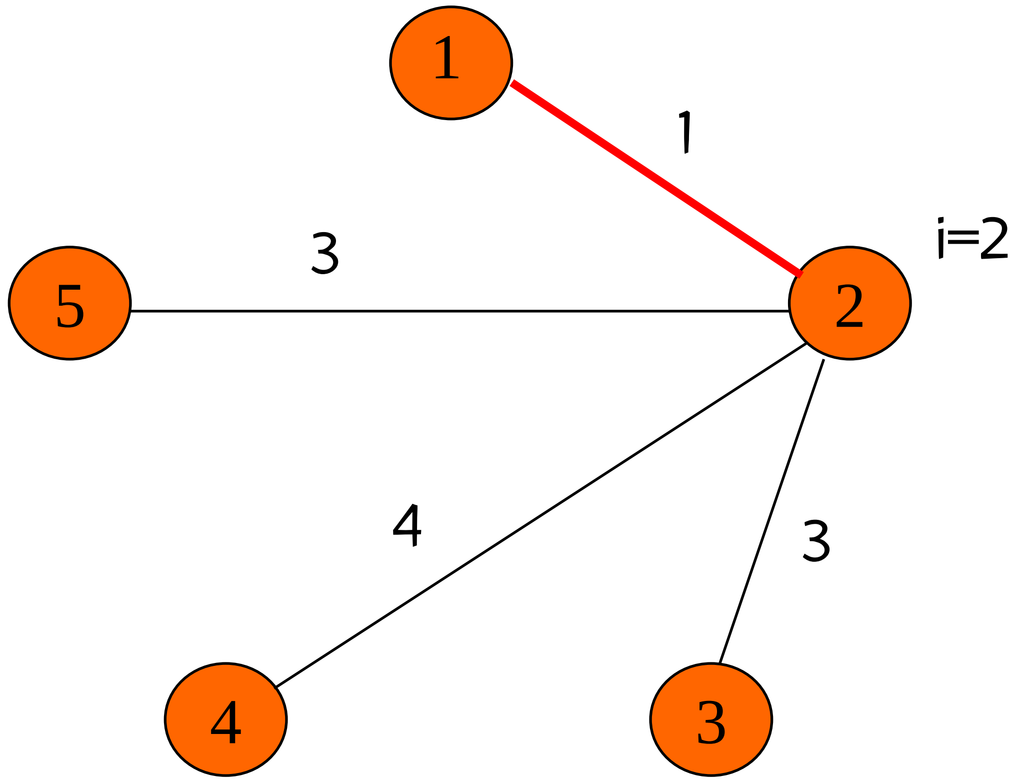
Problema do caixeiro viajante



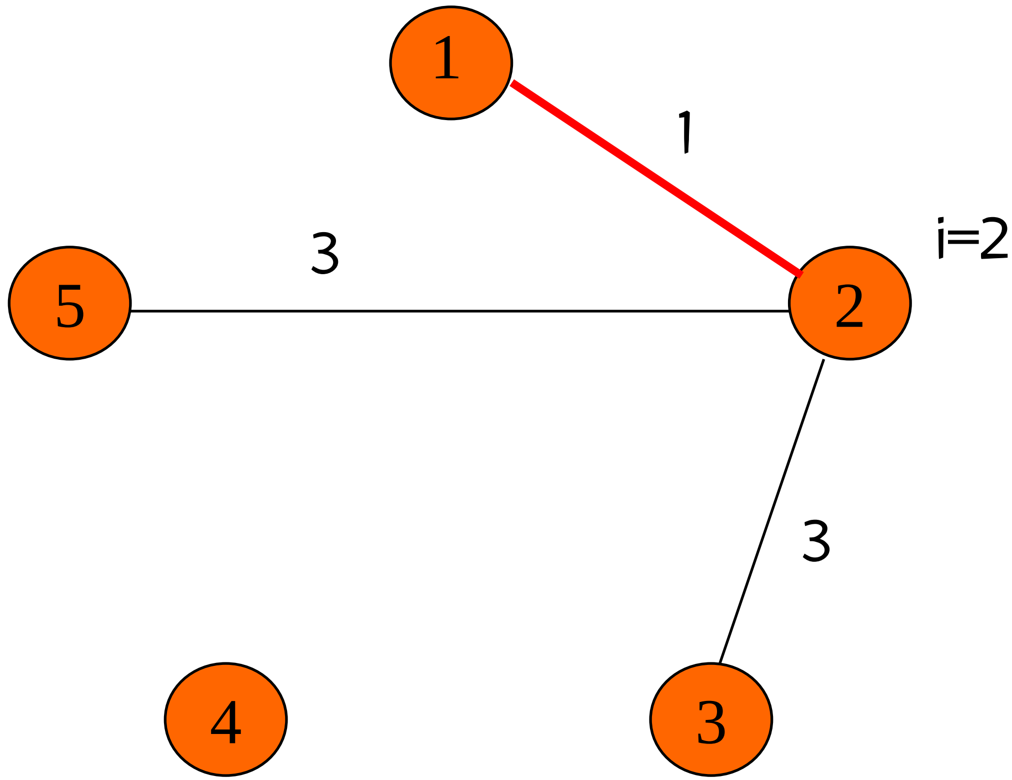
Problema do caixeiro viajante



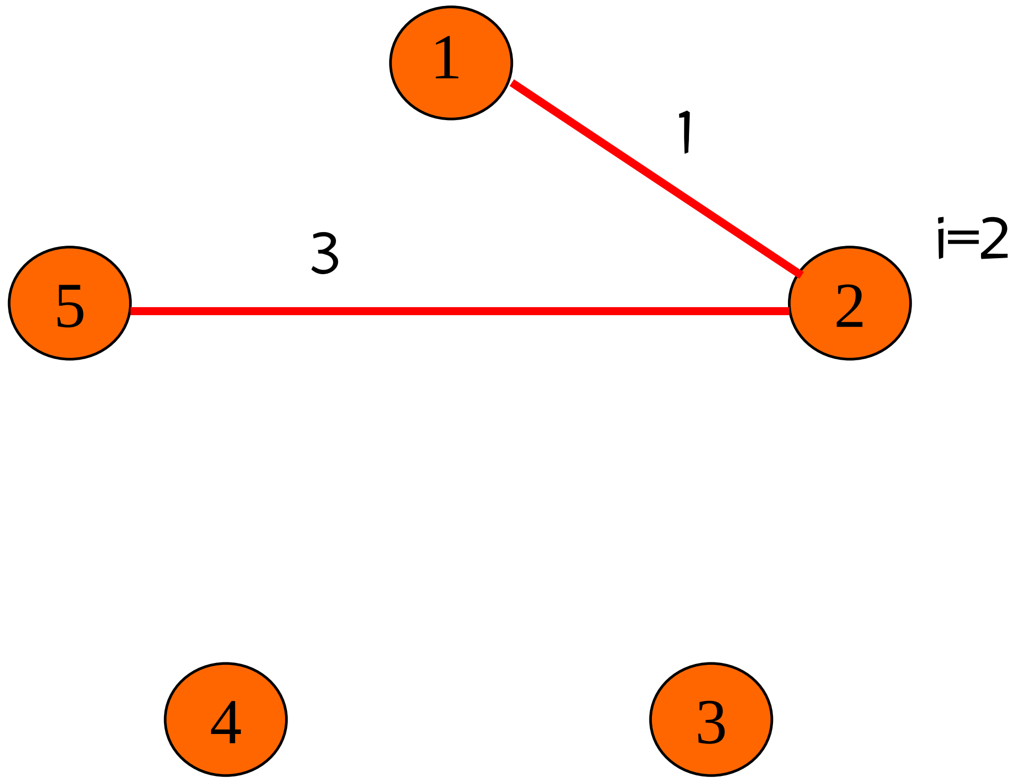
Problema do caixeiro viajante



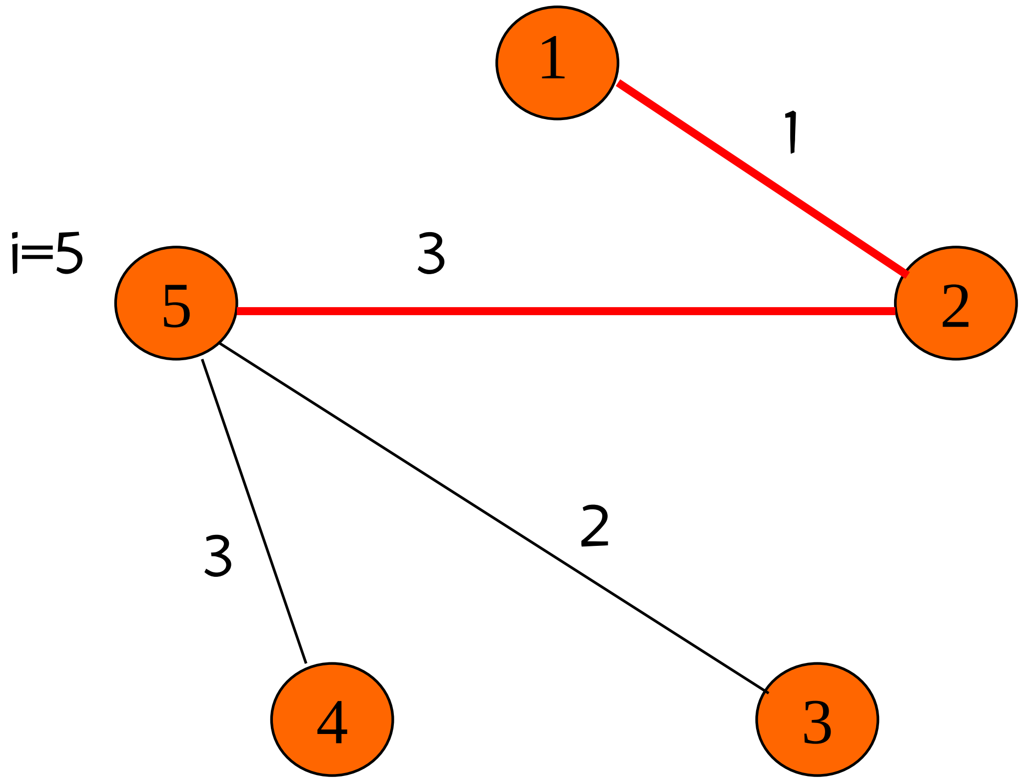
Problema do caixeiro viajante



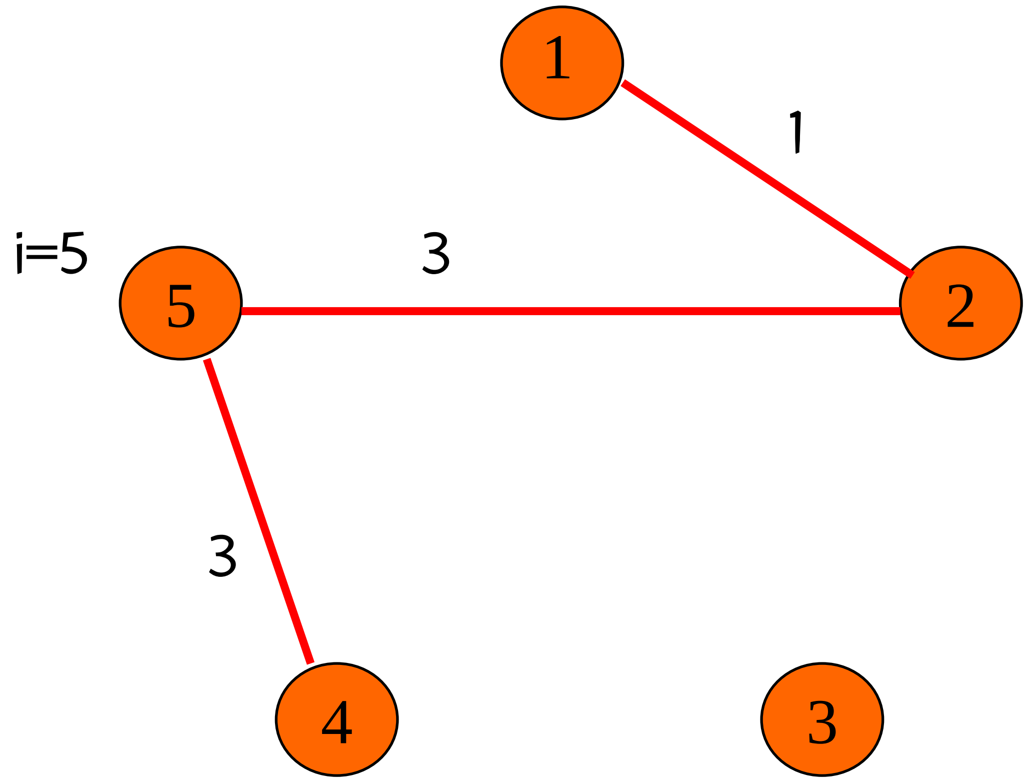
Problema do caixeiro viajante



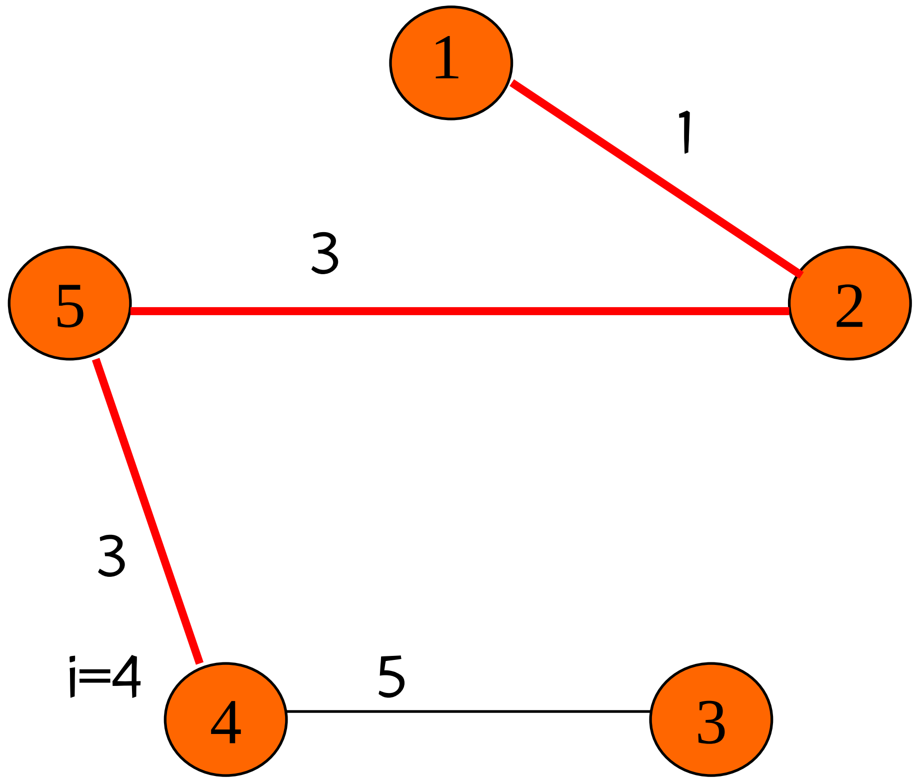
Problema do caixeiro viajante



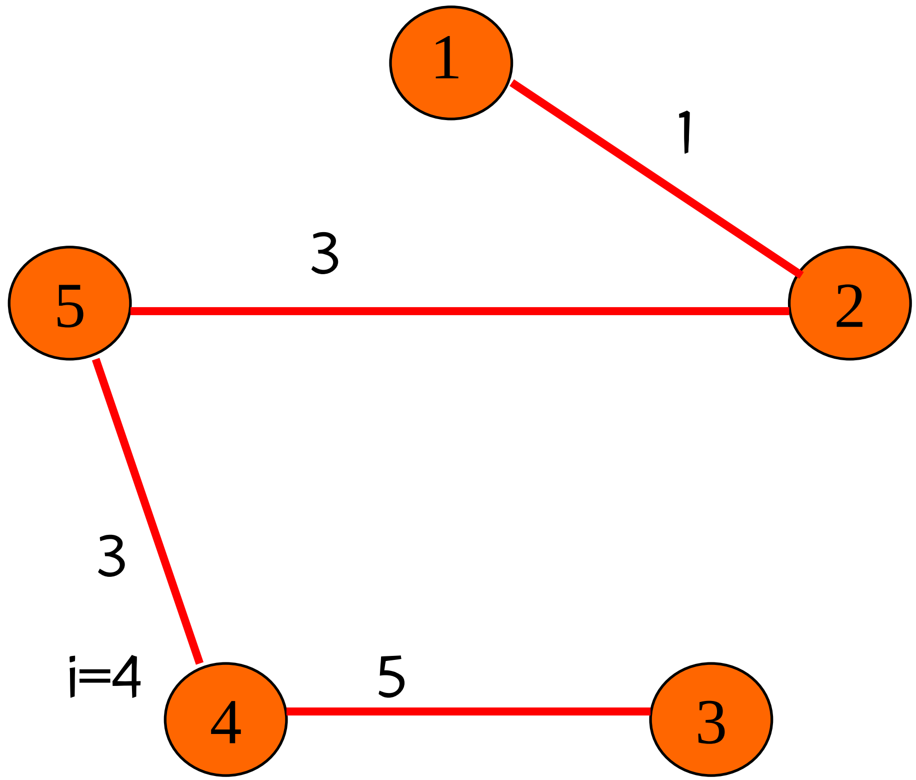
Problema do caixeiro viajante



Problema do caixeiro viajante

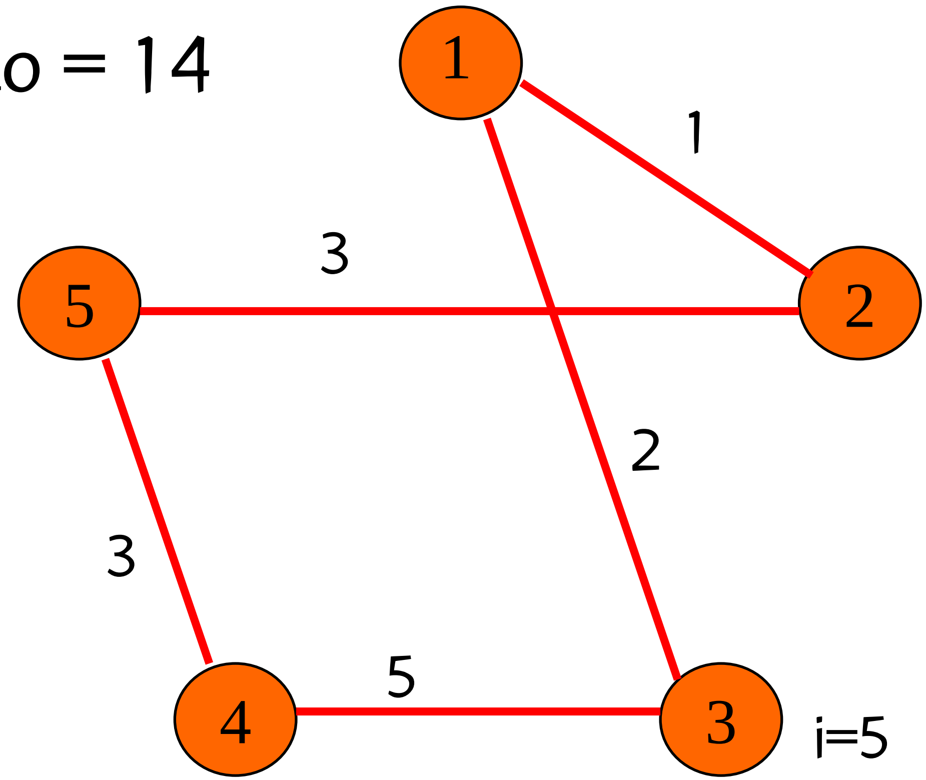


Problema do caixeiro viajante



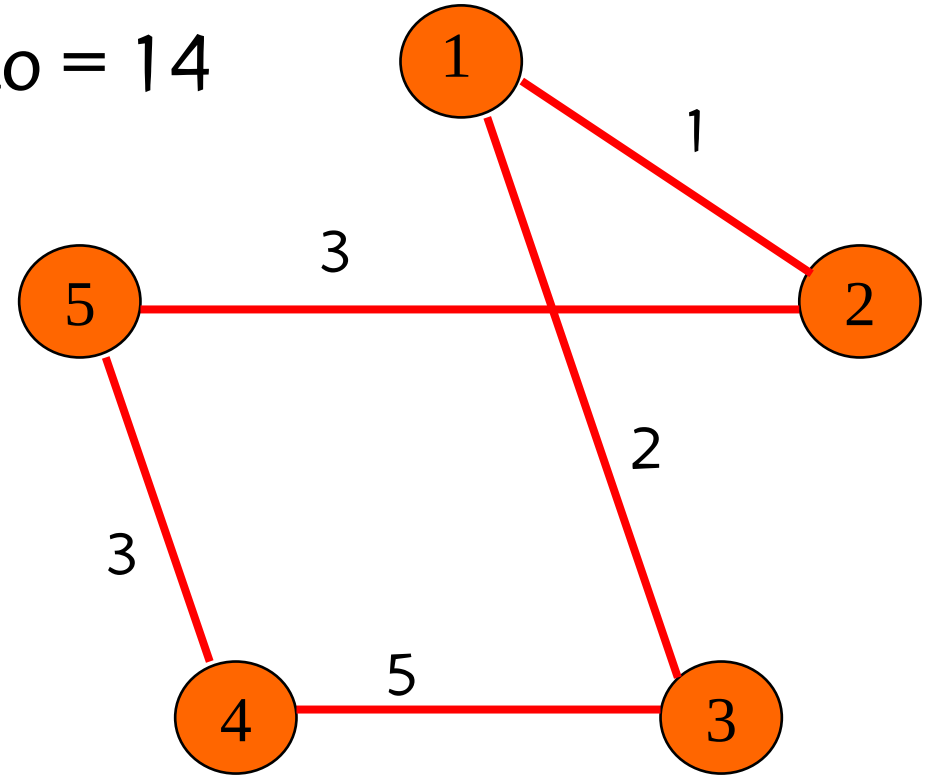
Problema do caixeiro viajante

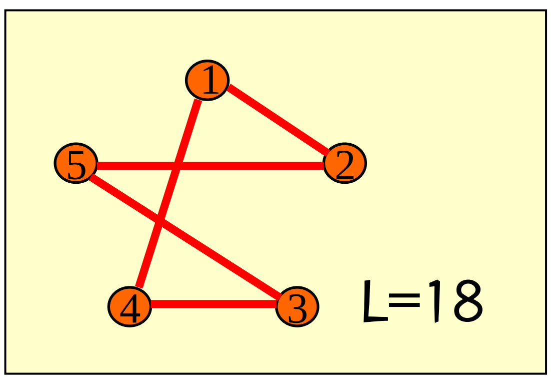
comprimento = 14



Problema do caixeiro viajante

comprimento = 14





Algoritmos gulosos randomizados

- A qualidade da solução obtida depende da qualidade dos elementos na lista de candidatos.
- A diversidade das soluções encontradas depende da cardinalidade da lista de candidatos.
- Casos extremos:
  - algoritmo guloso puro (o candidato único é o melhor)
  - solução gerada de forma completamente aleatória (todos os elementos pendentes são candidatos)

Algoritmos gulosos randomizados

- Quanto maior for o número de aplicações do algoritmo, maior a probabilidade de encontrar soluções melhores.
  - Melhores soluções, mas tempos de processamento maiores.
- Ajuste de parâmetros na implementação: equilibrar qualidade e diversidade

Busca local

- Técnica de exploração do espaço de soluções.
- Conjunto F de soluções (viáveis) formado por subconjuntos de um conjunto E (suporte/base) de elementos que satisfazem determinadas condições.
- Representação de uma solução: indicar quais elementos de E estão presentes e quais não estão.

Busca local

- Exemplo: problema da mochila

  n itens

  Solução é um vetor 0-1 com n posições:

  $x_j = 1$ se o item j é selecionado

  $x_j = 0$ caso contrário

  $n = 5$

  solução $x = (1,0,0,1,1)$: itens 1, 4 e 5 selecionados

Busca local

- Exemplo: problema do caixeiro viajante

  E: conjunto de arestas

  F: subconjuntos de E que formam um circuito hamiltoniano

  Solução é um vetor de $m = |E|$ posições:

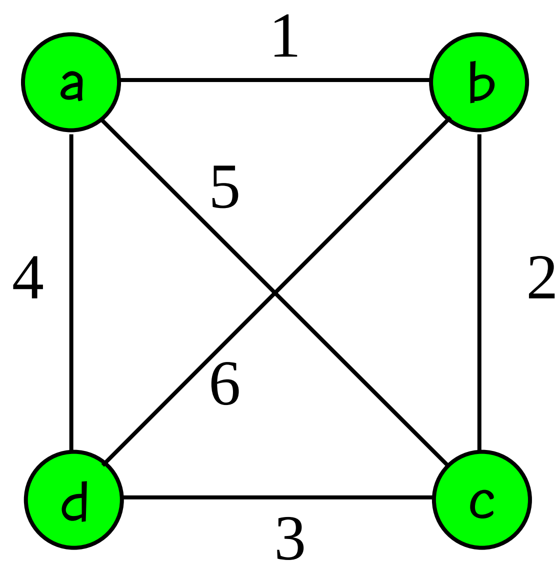
  $v_e = 1$, se a aresta e pertence ao circuito hamiltoniano

  $v_e = 0$, caso contrário.

Busca local

Soluções viáveis (64 vetores possíveis):

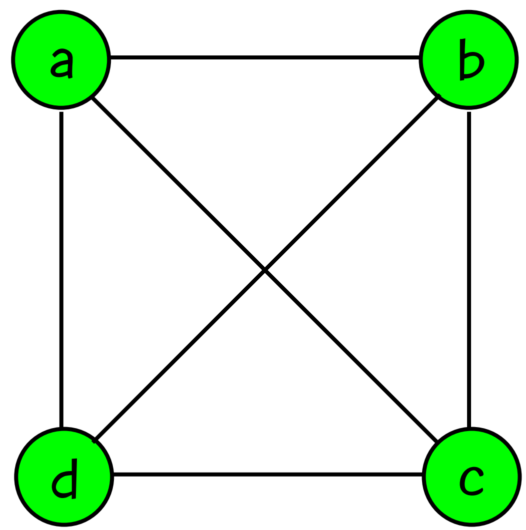
(1,1,1,1,0,0), (1,0,1,0,1,1), (0,1,0,1,1,1)



Busca local

Outra representação para as soluções do PCV: representar cada solução pela ordem em que os vértices são visitados, isto é, como uma permutação circular dos n vértices (já que o primeiro vértice é arbitrário)

(a)bcd
(a)bdc
(a)cbd
(a)cdb
(a)dbc
(a)dcb



Busca local

- Indicadores 0-1 de pertinência:
  - Problema da mochila
  - Problema de Steiner em grafos
  - Problemas de recobrimento e de particionamento, ...
- Indicadores gerais de pertinência:
  - Coloração de grafos
  - Localização, ...
- Permutações:
  - Problemas de escalonamento
  - Problema do caixeiro viajante, ...

Busca local

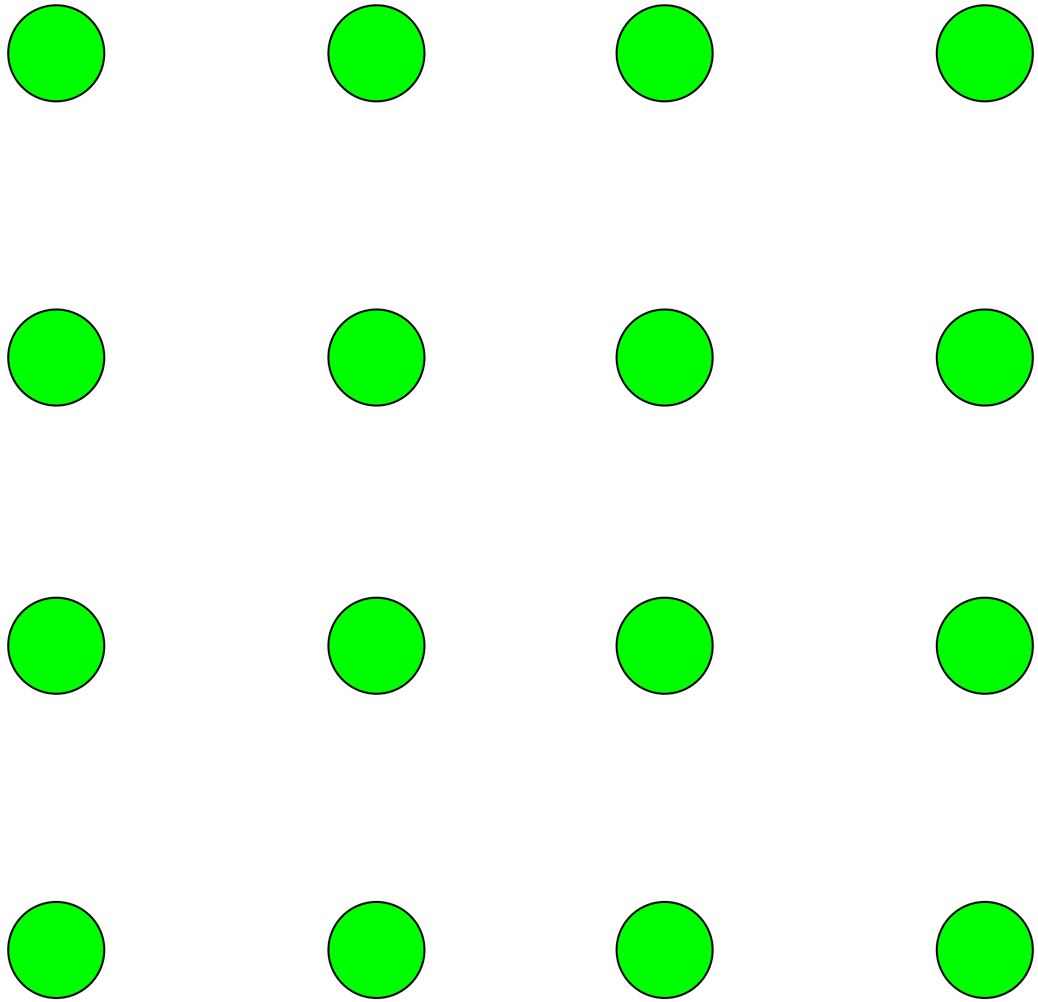
- Problema de otimização combinatória:
  $f(s^*) = \text{mínimo}\ \{f(s): s \in S\}$
  S é um conjunto discreto de soluções
- Vizinhança: elemento que introduz a noção de proximidade entre as soluções de S.
- Uma vizinhança é um mapeamento que associa cada solução s a um subconjunto de soluções (vizinhos).

  $N(s) = \{s_1, s_2, \ldots, s_k\}$: soluções vizinhas de s

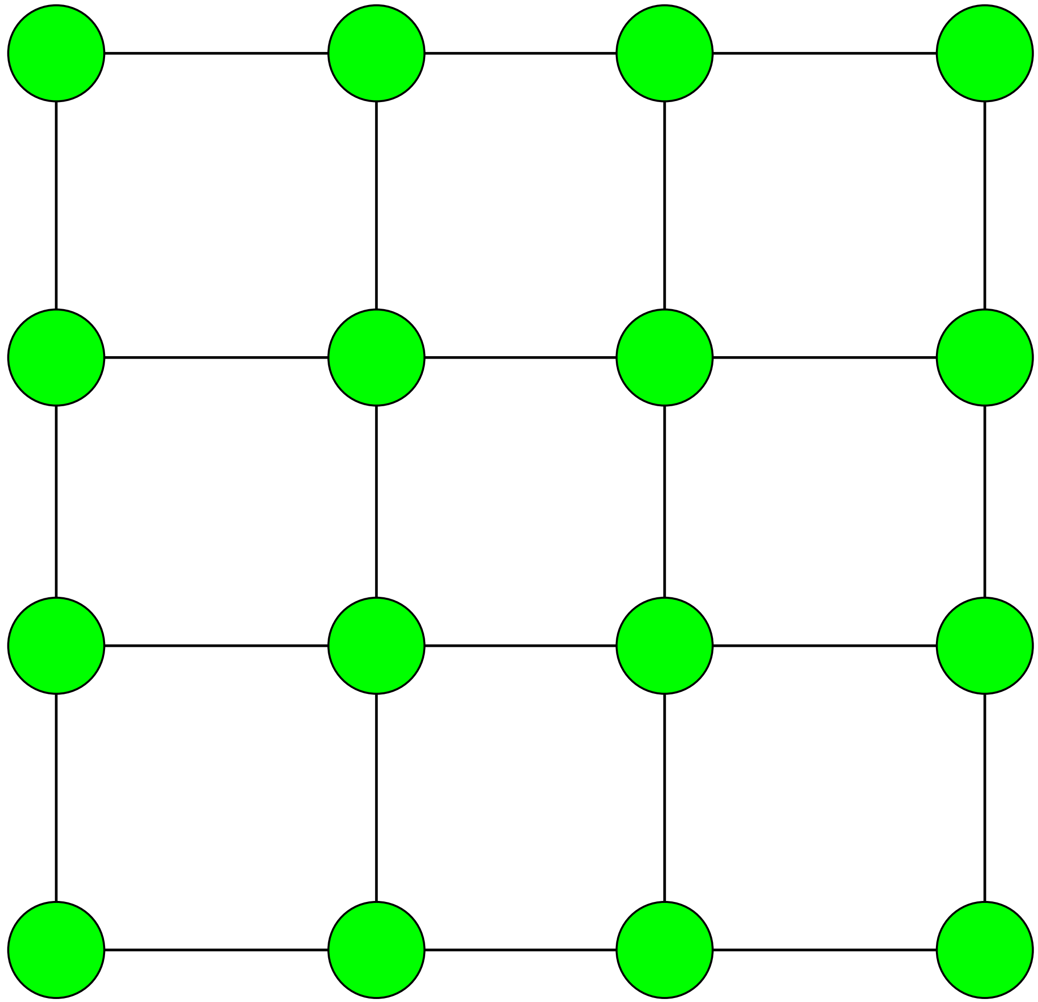
Busca local

- Boas vizinhanças permitem representar de forma compacta o conjunto de soluções vizinhas de uma solução s qualquer e percorrer (visitar, explorar) de maneira eficiente o conjunto de soluções.

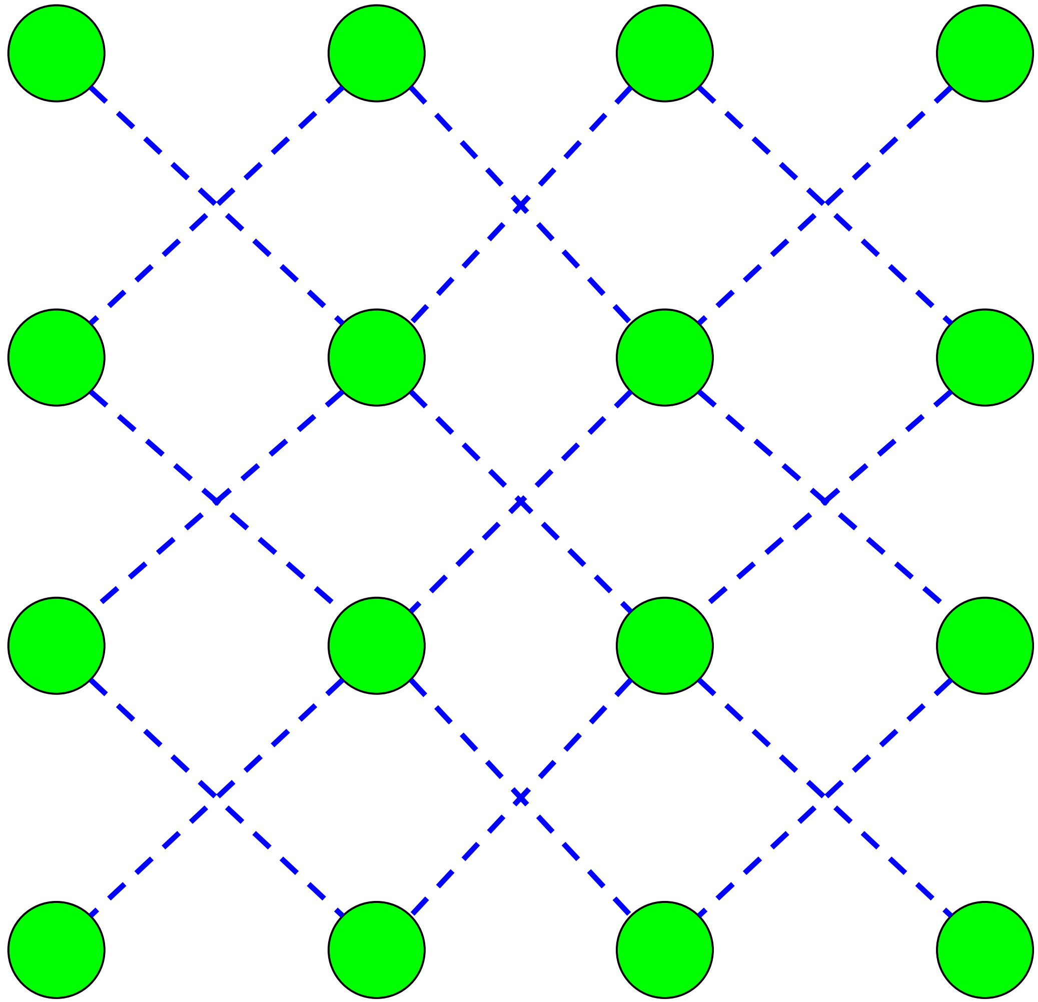
Busca local



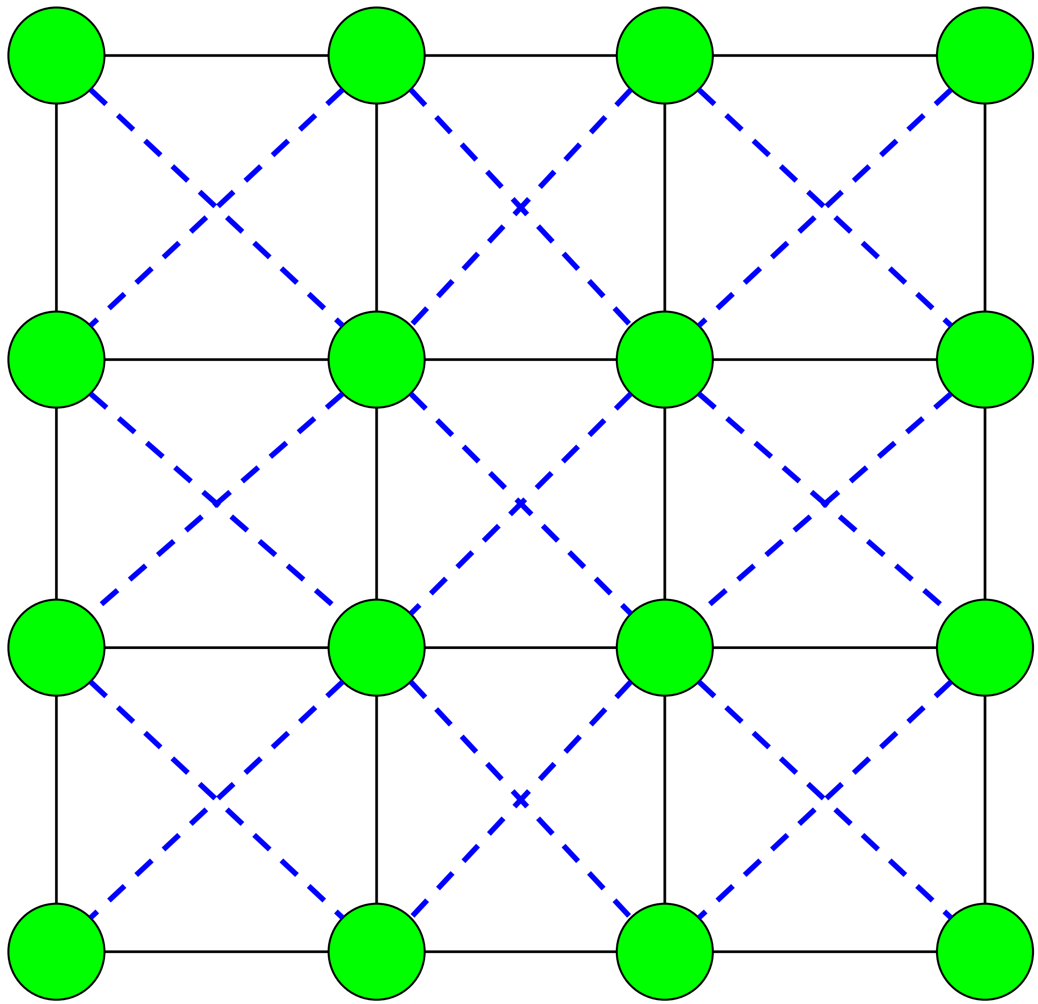
Busca local



Busca local



Busca local



Busca local

- Espaço de busca: definido pelo conjunto de soluções S e por uma vizinhança N
- Representação por um grafo:
  - Nós $\rightarrow$ soluções
  - Arestas $\rightarrow$ soluções vizinhas

Busca local

- Exemplo: problema da mochila
  vetor de pertinência 0-1

  Solução $s = (s_1,\ldots,s_i,\ldots,s_n)$ $s_i \in \{0,1\}$, $i=1,\ldots,n$

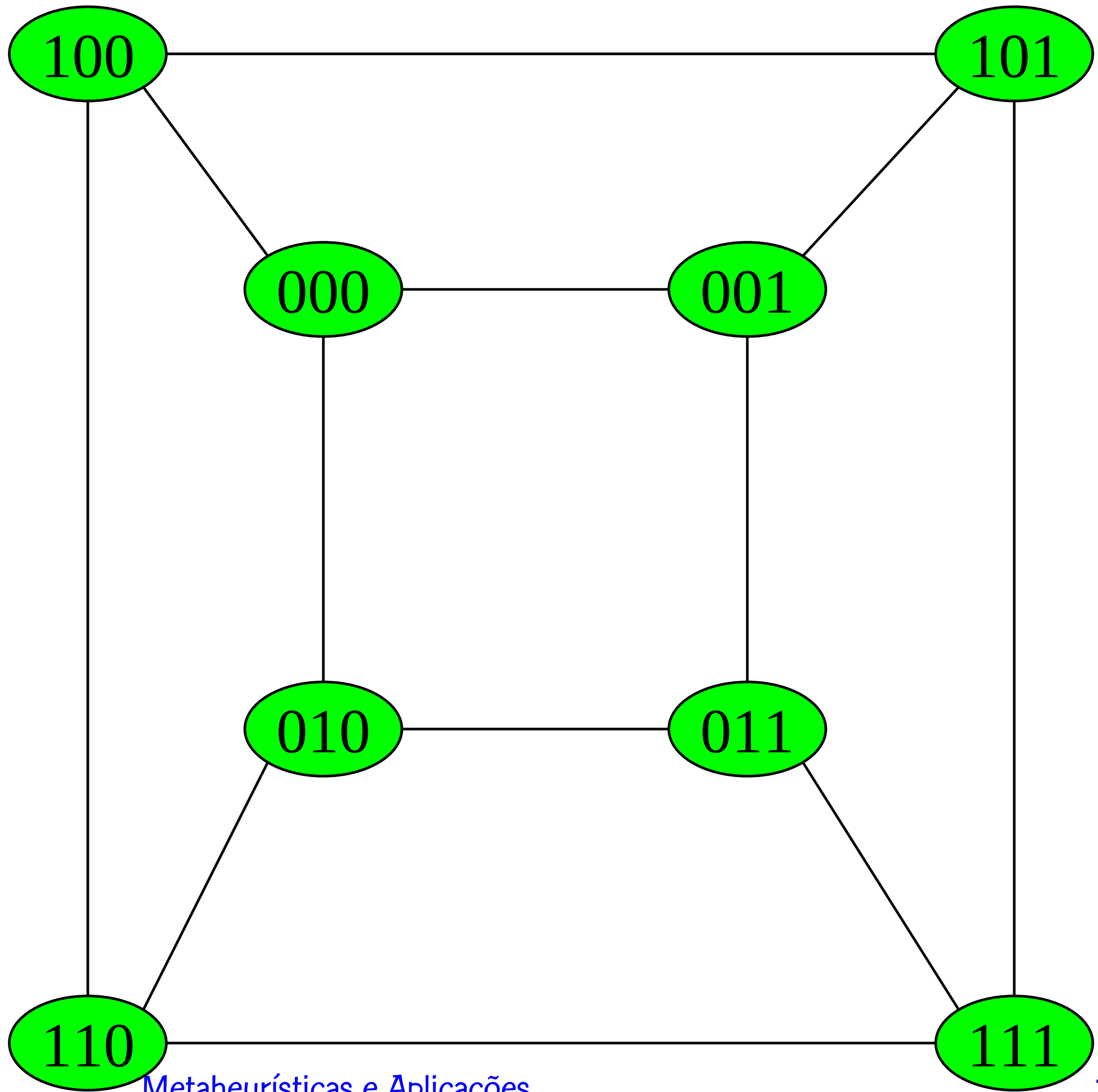
  $N(s)=\{(s_1,\ldots,1-s_i,\ldots,s_n): i=1,\ldots,n\}$

  Vizinhos de (1,0,1,1)=
  {(<u>0</u>,0,1,1),(1,<u>1</u>,1,1),(1,0,<u>0</u>,1),(1,0,1,<u>0</u>)}

Busca local

$n=3$

$s=(s_1,s_2,s_3)$



Busca local

- Exemplo: problema do caixeiro viajante (permutações)

  Solução $\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_j,\ldots,\pi_n)$

- $N_1(\pi)=\{(\pi_1,\ldots,\pi_{i+1},\pi_i,\ldots,\pi_n): i=1,\ldots, n-1\}$

  Troca da posição de duas cidades consecutivas:

  vizinhos de (1,2,3,4) = {(2,1,3,4),(1,3,2,4),(1,2,4,3)}

$$\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_n)$$

$$\pi'=(\pi_1,\ldots,\pi_{i-1},\pi_{i+1},\pi_I,\ldots,\pi_n)$$

Busca local

- Exemplo: problema do caixeiro viajante (permutações)

  Solução $\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_j,\ldots,\pi_n)$

- $N_2(\pi)=\{(\pi_1,\ldots,\pi_j,\ldots,\pi_i,\ldots,\pi_n)$: $i=1,\ldots,n-1$; $j=i+1,\ldots,n\}$

  Troca da posição de duas cidades quaisquer:

  vizinhos de (1,2,3,4)= {(2,1,3,4),(1,3,2,4),
  (1,2,4,3),(3,2,1,4),(1,4,3,2),(4,2,3,1)}

$$\pi=(\pi_1,\ldots,\pi_i,\ldots,\pi_j,\ldots,\pi_n)$$

$$\pi'=(\pi_1,\ldots,\pi_j,\ldots,\pi_i,\ldots,\pi_n)$$

Busca local

- <u>Exemplo</u>: problema do caixeiro viajante (permutações)

  Solução $\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_j,\ldots,\pi_n)$

- $N_3(\pi)=\{(\pi_1,\ldots, \pi_{i-1},\pi_{i+1},\ldots,\pi_j,\pi_i,\ldots,\pi_n)$: $i=1,\ldots,n-1$; $j=i+1,\ldots,n\}$

  Inserção de uma cidade em uma posição qualquer:

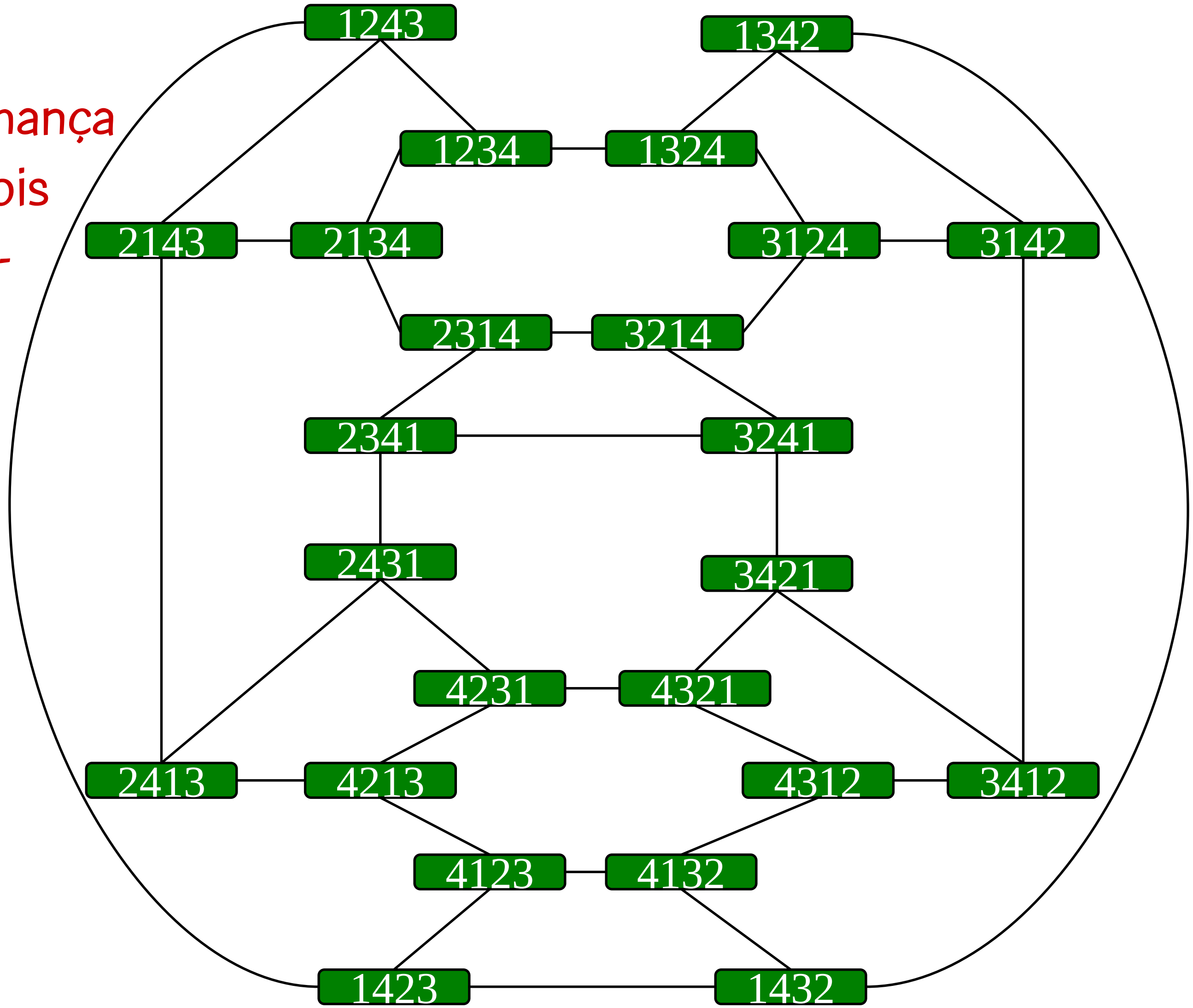
  vizinhos de (1,2,3,4)= {(2,1,3,4),(2,3,1,4), (2,3,4,1),(1,3,2,4),(1,3,4,2),(1,2,4,3)}

$$\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_j,\pi_{j+1},\ldots,\pi_n)$$

$$\pi'=(\pi_1,\ldots,\pi_{i-1},\pi_{i+1},\ldots,\pi_j,\pi_i,\pi_{j+1},\ldots,\pi_n)$$

Busca local

Exemplo: vizinhança de troca N_1 (dois elementos consecutivos)



Busca local

- O espaço de busca pode ser visto como um grafo cujos vértices são as soluções e no qual existem arestas entre pares de vértices associados a soluções vizinhas.
- Este espaço pode ser visto como uma superfície com vales e cumes definidos pelo valor e pela proximidade (vizinhança) das soluções (noção de proximidade induz o conceito de distância entre soluções)
- Um caminho no espaço de busca consiste numa seqüência de soluções, onde duas soluções consecutivas quaisquer são vizinhas.

Busca local

- Ótimo local: solução tão boa ou melhor do que qualquer solução vizinha
- Problema de minimização:

  s^+ é um ótimo local

  $$\Updownarrow$$

  $$f(s^+) \leq f(s),\ \forall s \in N(s^+)$$

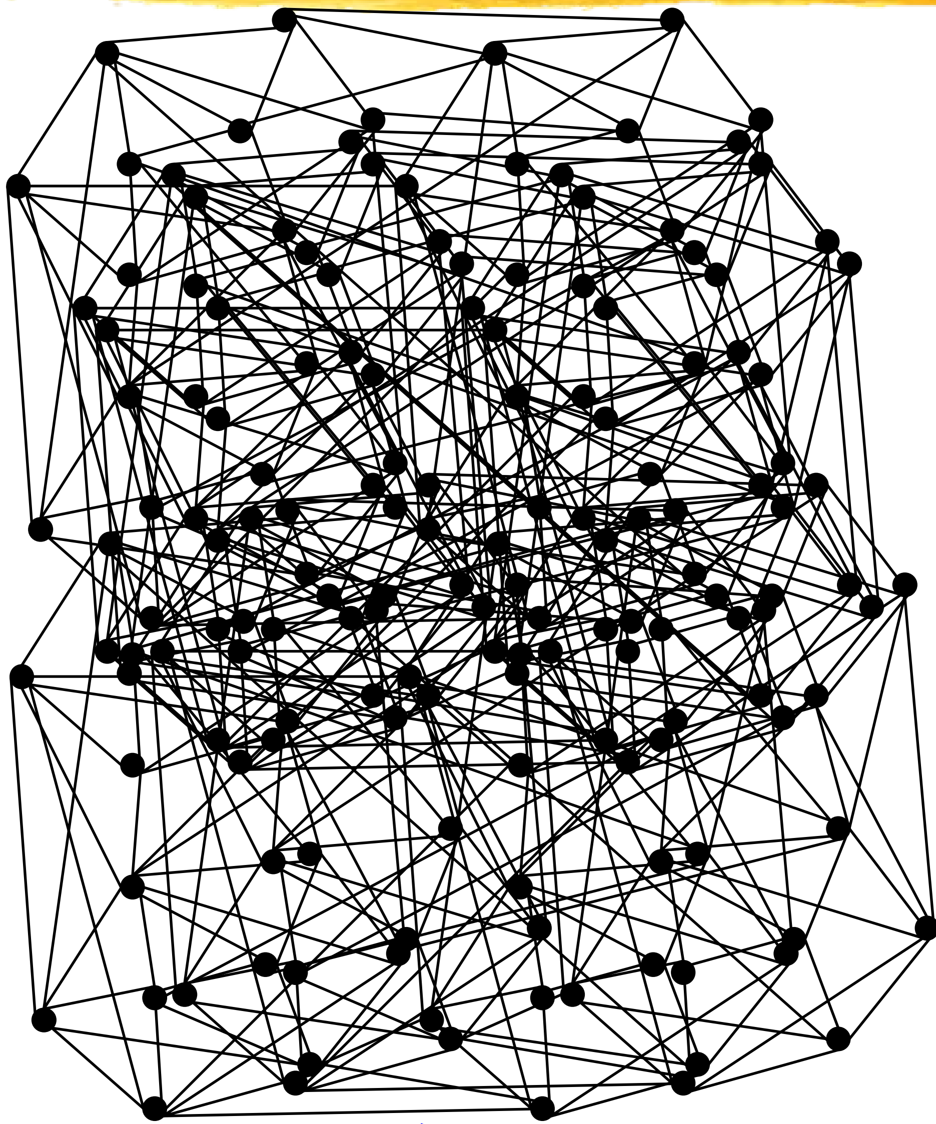
- Ótimo global ou solução ótima s^*:

  $$f(s^*) \leq f(s),\ \forall s \in S$$

Busca local

- Algoritmos de busca local são construídos como uma estratégia de exploração do espaço de busca.
- Partida: solução inicial obtida através de um método construtivo
- Iteração: melhoria sucessiva da solução corrente através de uma busca na sua vizinhança
- Parada: primeiro ótimo local encontrado (não existe solução vizinha aprimorante)
- Heurística subordinada utilizada para obter uma solução aprimorante na vizinhança

Busca local



Busca local

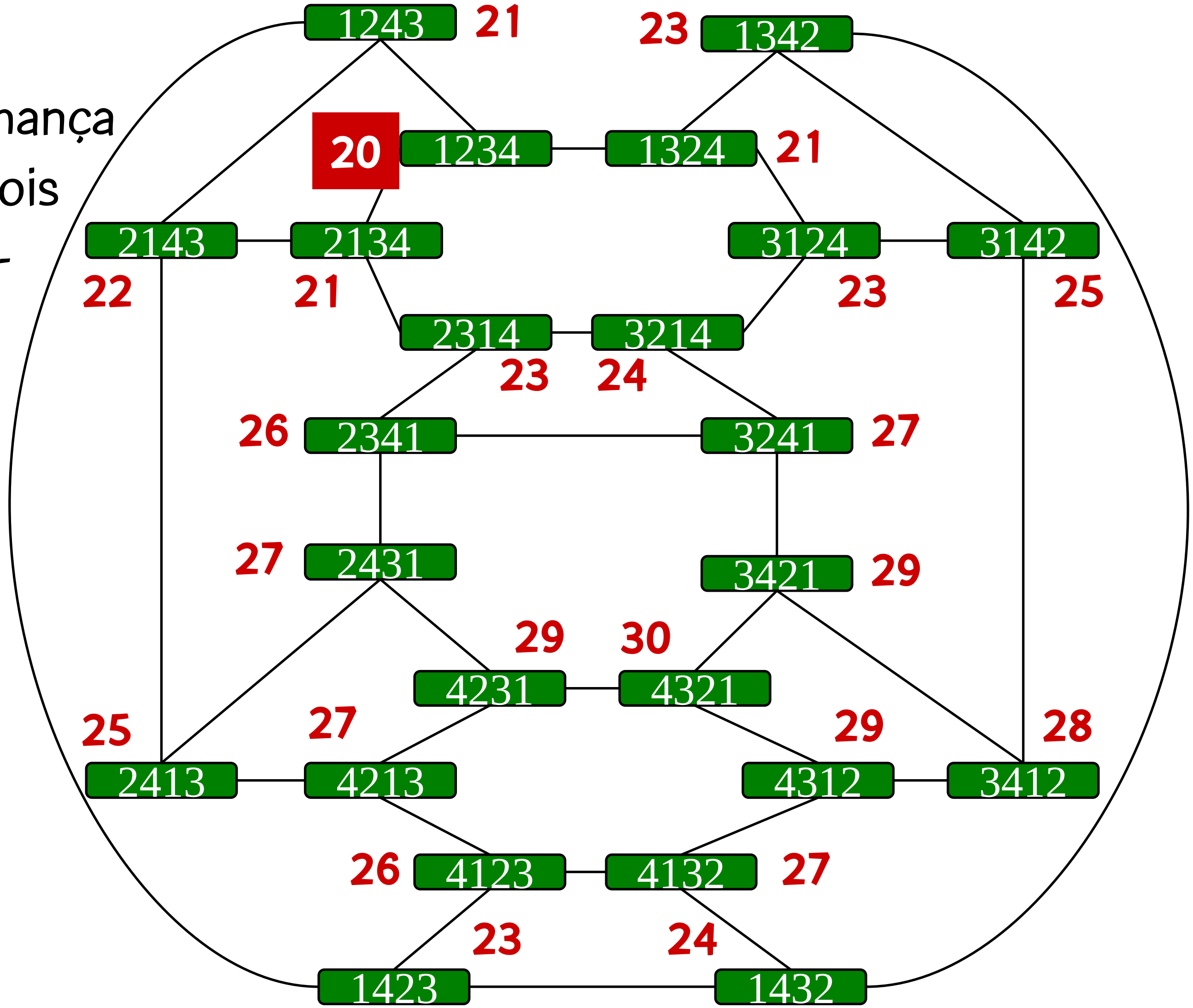
- Questões fundamentais:
  - Solução inicial
  - Definição da vizinhança
  - Estratégia de busca na vizinhança:
    - Melhoria iterativa: a cada iteração, selecionar qualquer (eventualmente a primeira) solução aprimorante na vizinhança
    - Descida mais rápida: a cada iteração, selecionar a melhor solução aprimorante na vizinhança

Busca local

- Questões fundamentais:
  - Complexidade de cada iteração:
    - Proporcional ao tamanho da vizinhança
    - Eficiência depende da forma como é calculada a função objetivo para cada solução vizinha: algoritmos eficientes são capazes de atualizar os valores quando a solução corrente se modifica, evitando cálculos repetitivos e desnecessários da função objetivo.

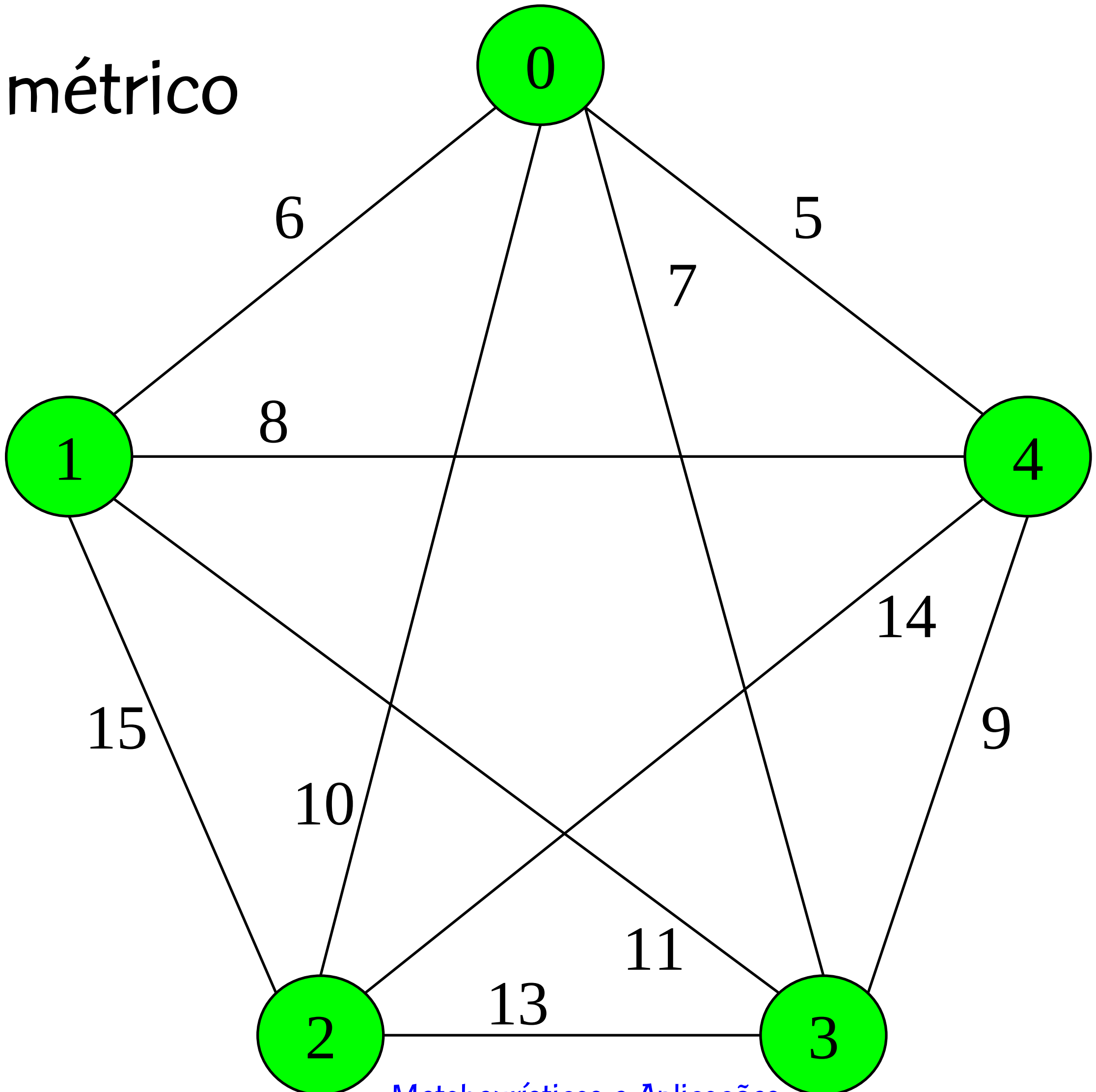
Busca local

<u>Exemplo</u>: vizinhança de troca N1 (dois elementos consecutivos)



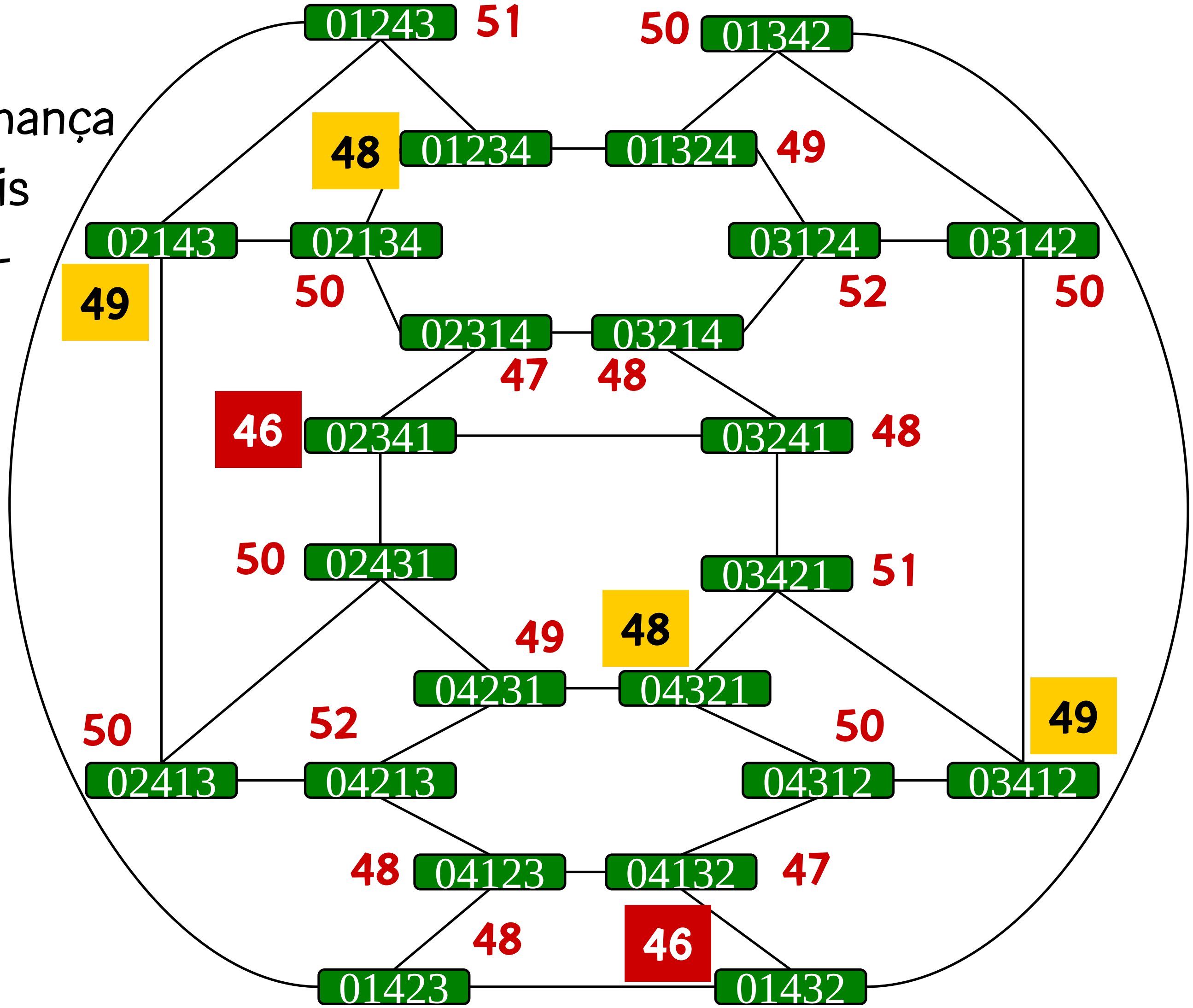
Problema do caixeiro viajante

Problema simétrico



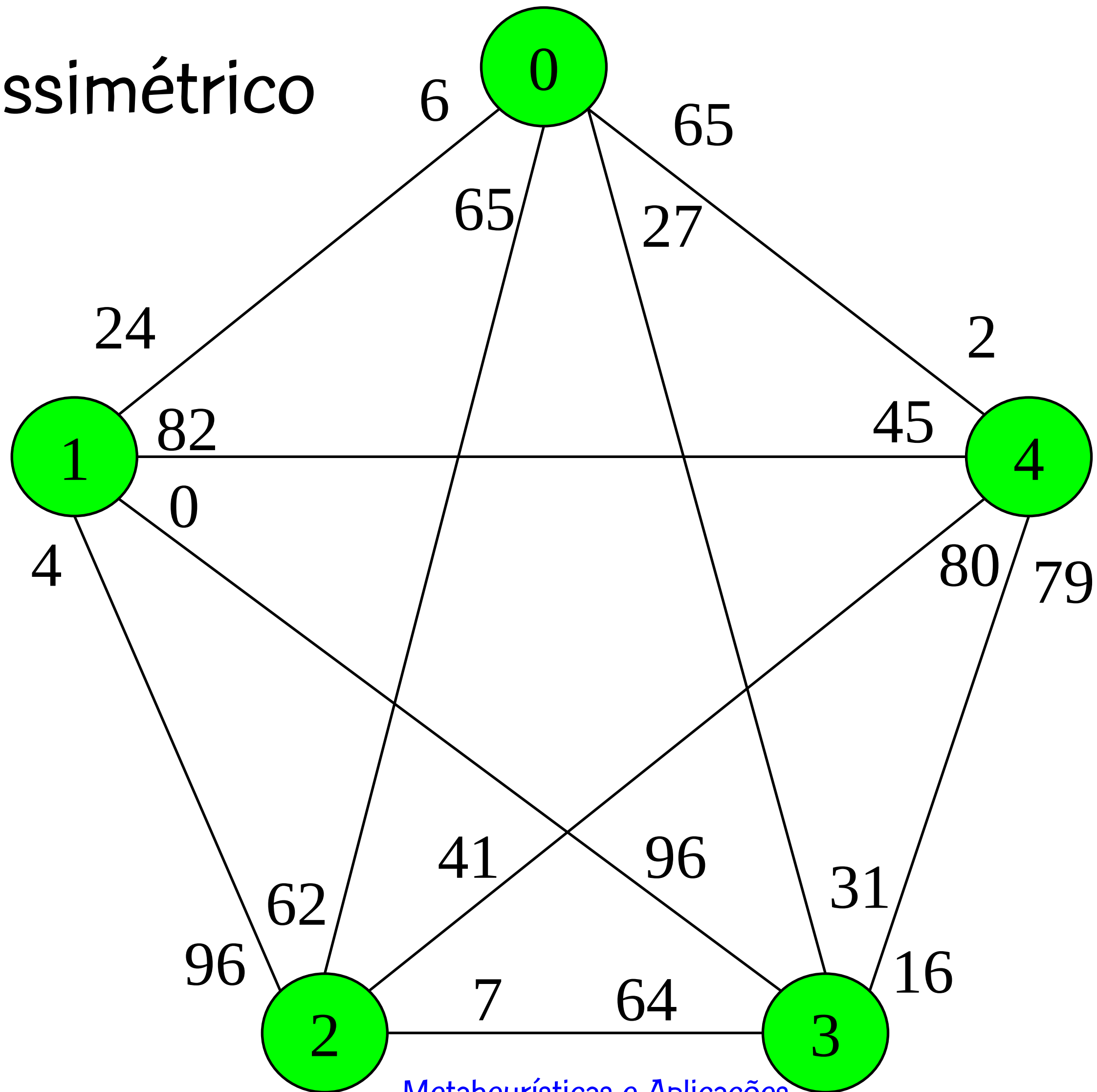
Problema do caixeiro viajante

<u>Exemplo</u>: vizinhança de troca de dois elementos consecutivos



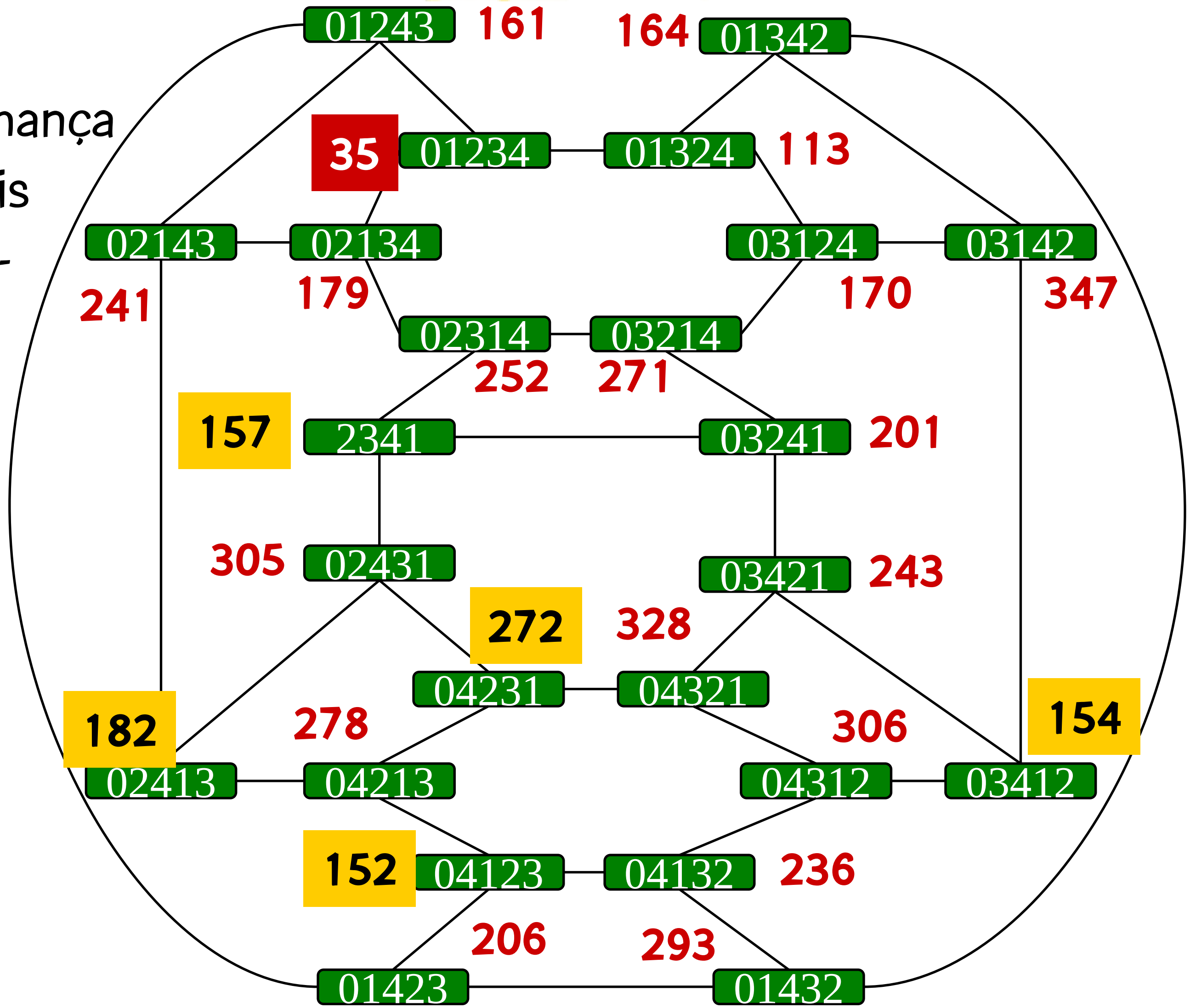
Problema do caixeiro viajante

Problema assimétrico



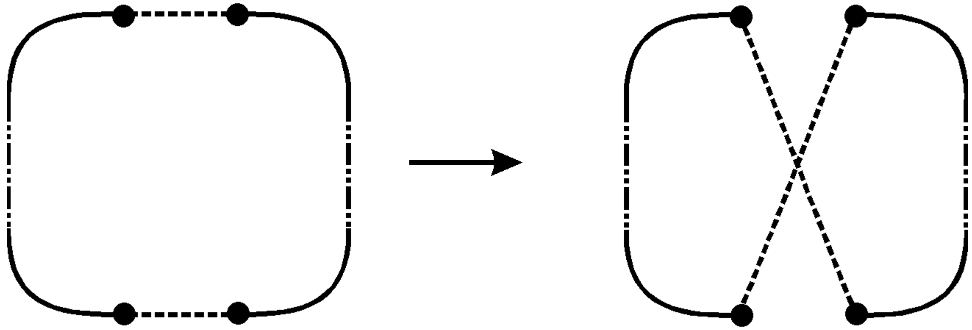
Problema do caixeiro viajante

<u>Exemplo</u>: vizinhança de troca de dois elementos consecutivos



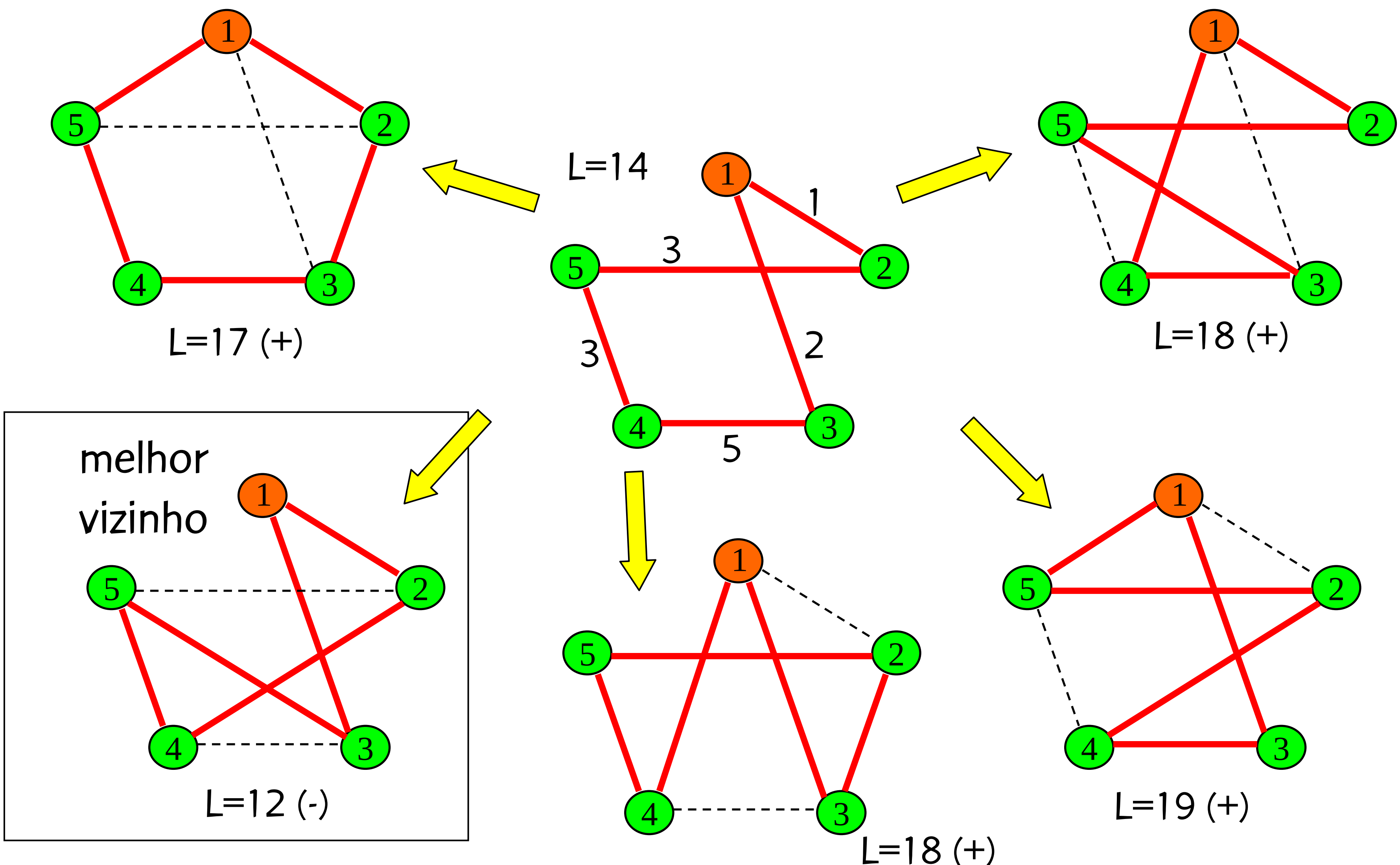
Problema do caixeiro viajante

- Vizinhança 2-opt para o problema do caixeiro viajante:



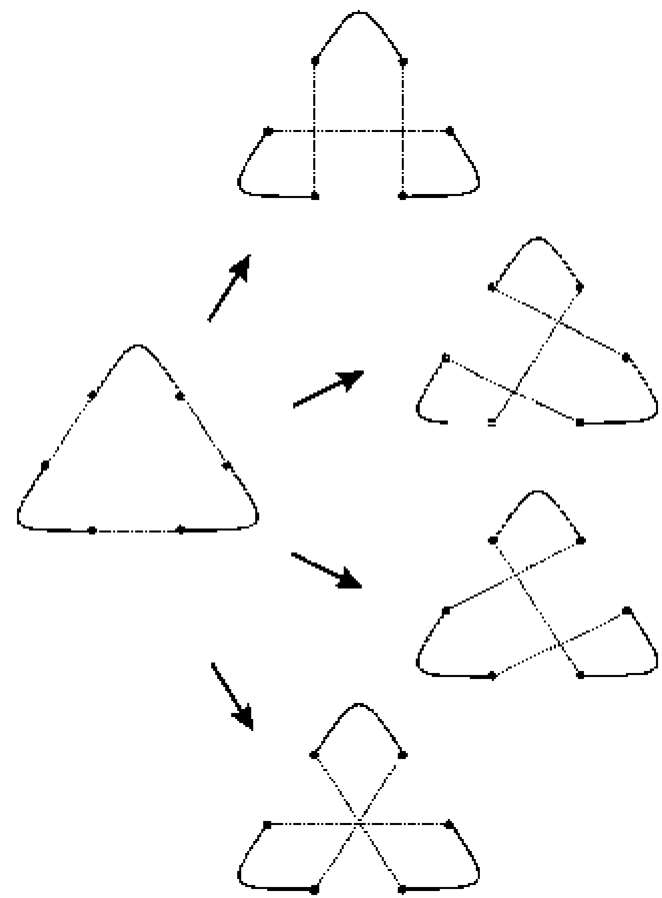
- Escolher duas arestas quaisquer, eliminá-las e substituí-las por outro par (único) de arestas de modo a formar novo circuito.
- Há da ordem de n^2 soluções vizinhas.
- O custo de cada solução vizinha pode ser facilmente recalculado.

Problema do caixeiro viajante



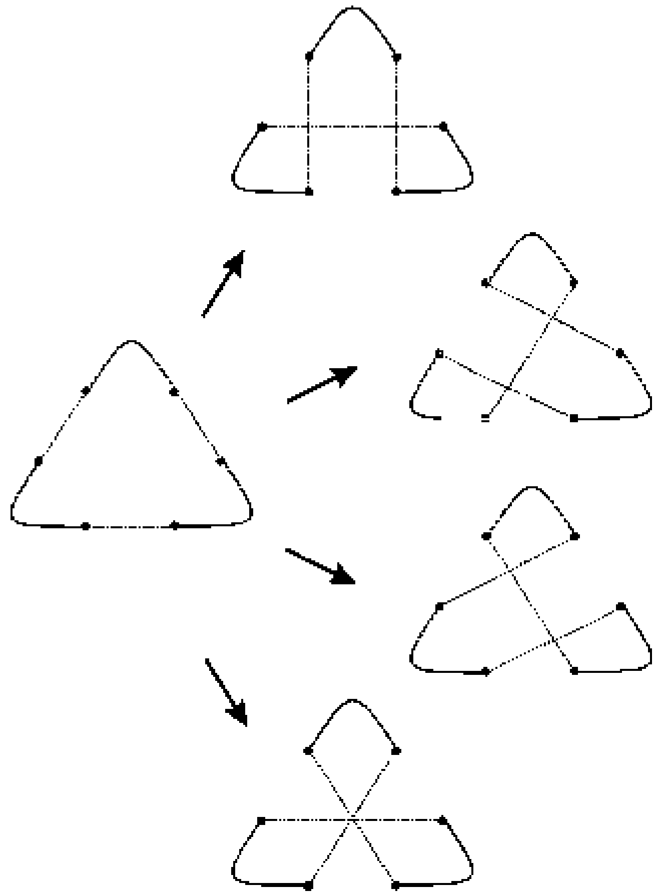
Problema do caixeiro viajante

- Vizinhança 3-opt para o problema do caixeiro viajante:
- Escolher três arestas quaisquer, eliminá-las e substituí-las por três outras arestas, de modo a formar novo circuito.
- Há da ordem de n^3 soluções vizinhas.
- O custo de cada solução vizinha continua podendo ser facilmente recalculado.



Problema do caixeiro viajante

- Vizinhança k-opt para o problema do caixeiro viajante:
- Extensão até n-opt corresponderia a uma busca exaustiva do espaço de soluções!
- Número de vizinhos e complexidade (tempo) de cada iteração aumentam com k, enquanto o ganho possível diminui progressivamente.



Problema de Steiner em grafos

- Grafo não-orientado G=(V,E)

  V: vértices

  E: arestas

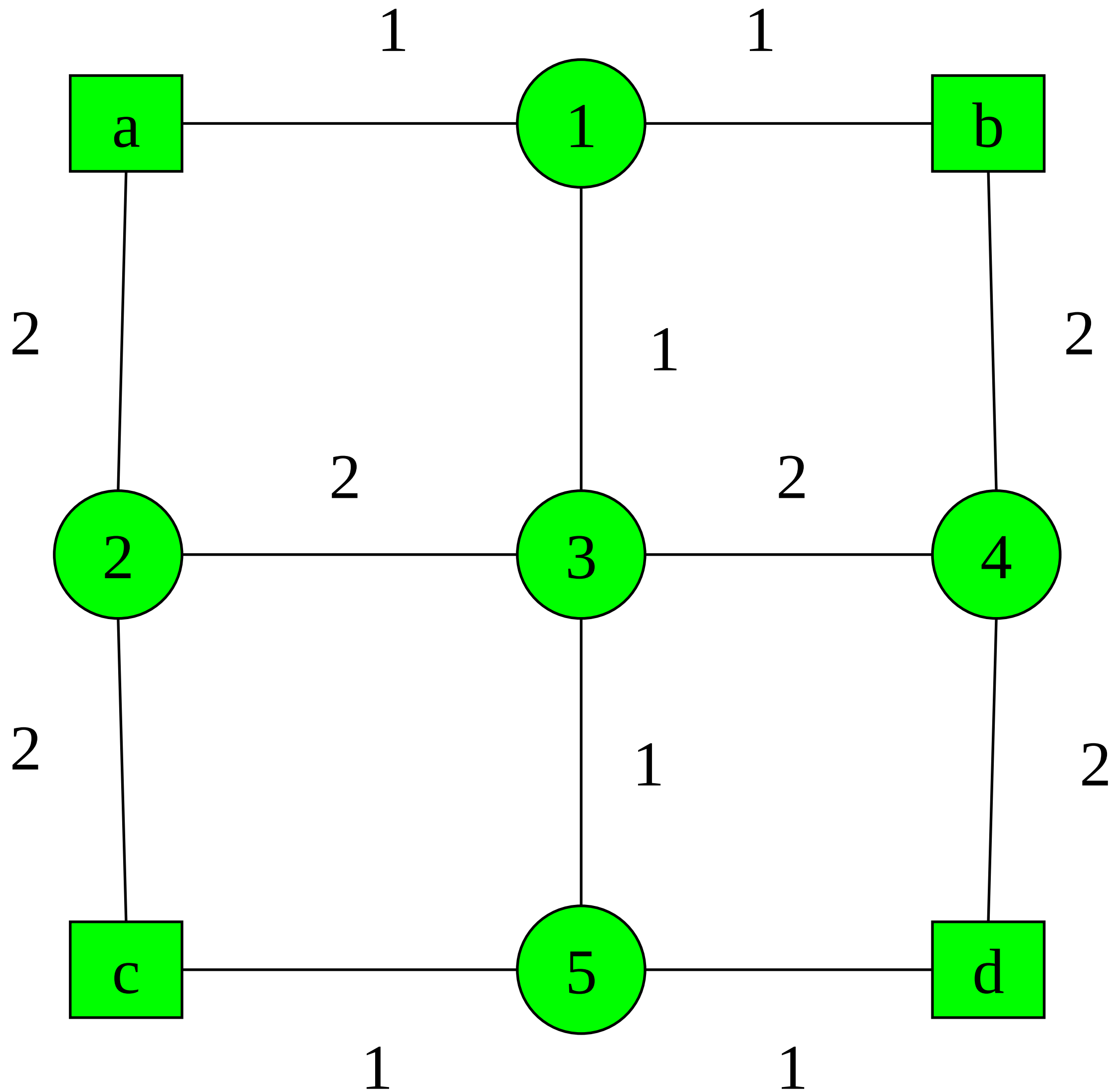
  T: vértices terminais (obrigatórios)

  c_e: peso (positivo) da aresta $e \in E$
- Problema: conectar os nós terminais com custo (peso) mínimo, eventualmente utilizando os demais nós como passagem.

Problema de Steiner em grafos

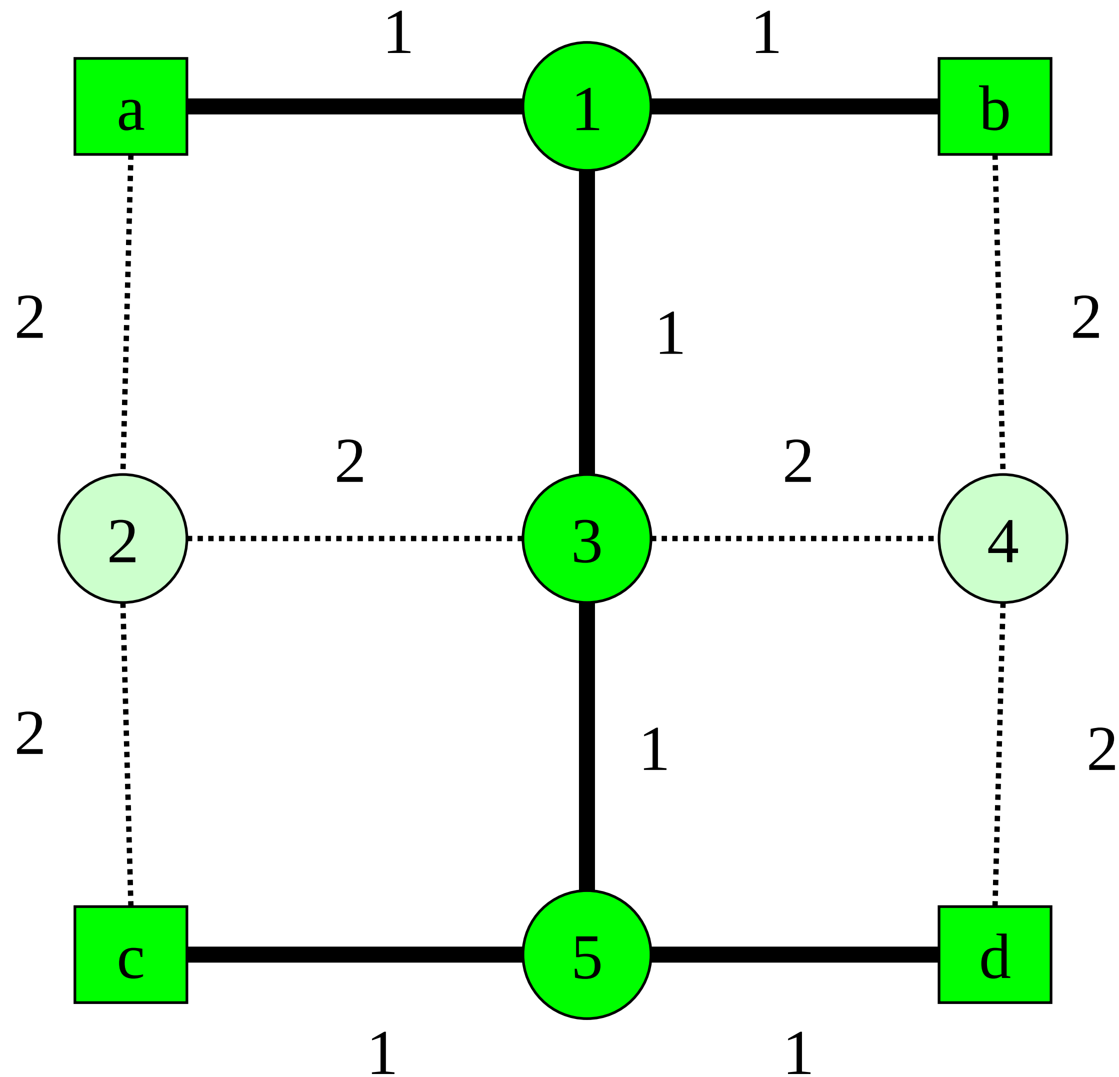
- Vértices de Steiner: vértices opcionais que fazem parte da solução ótima

Problema de Steiner em grafos



Problema de Steiner em grafos

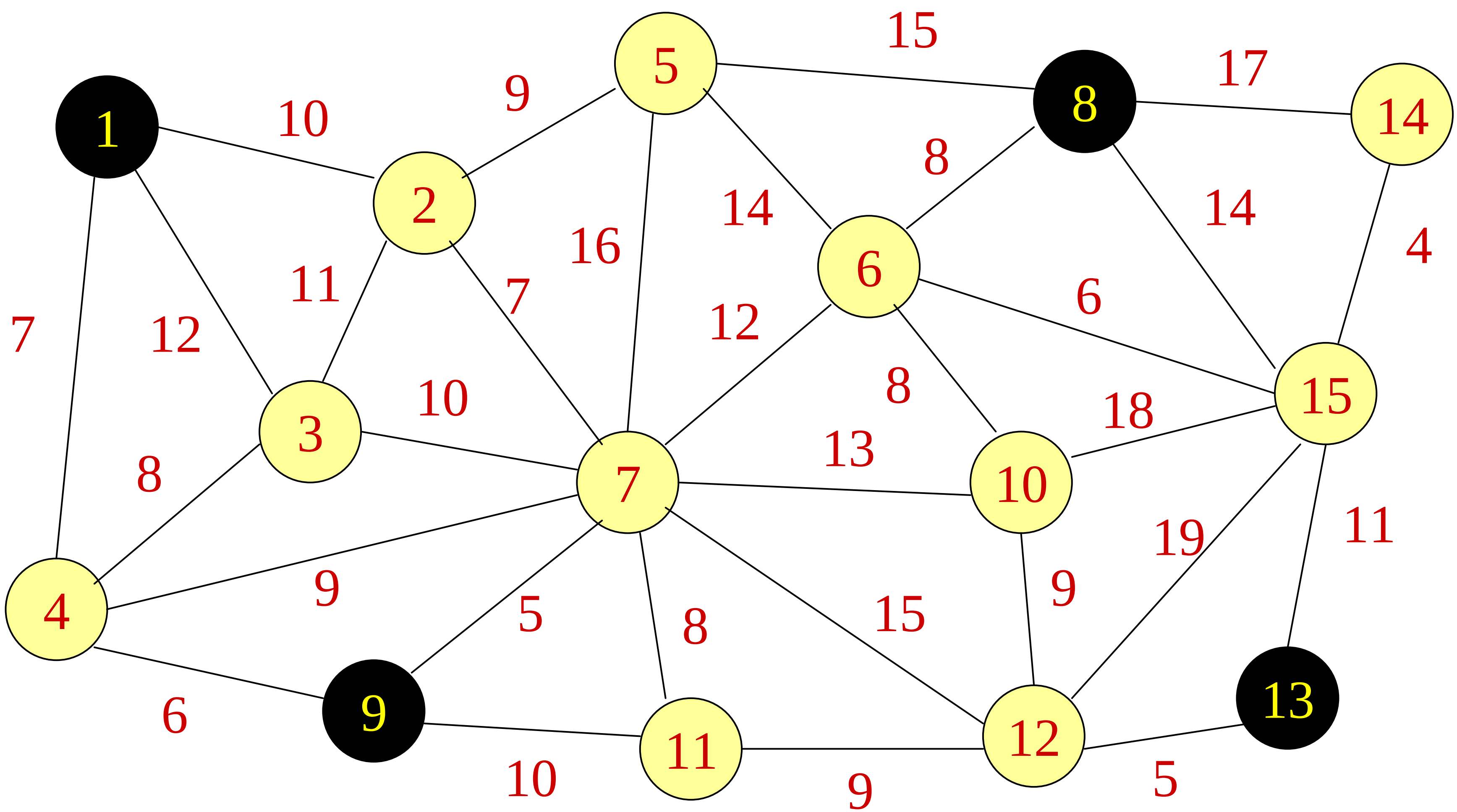
Mínimo: 6



Problema de Steiner em grafos

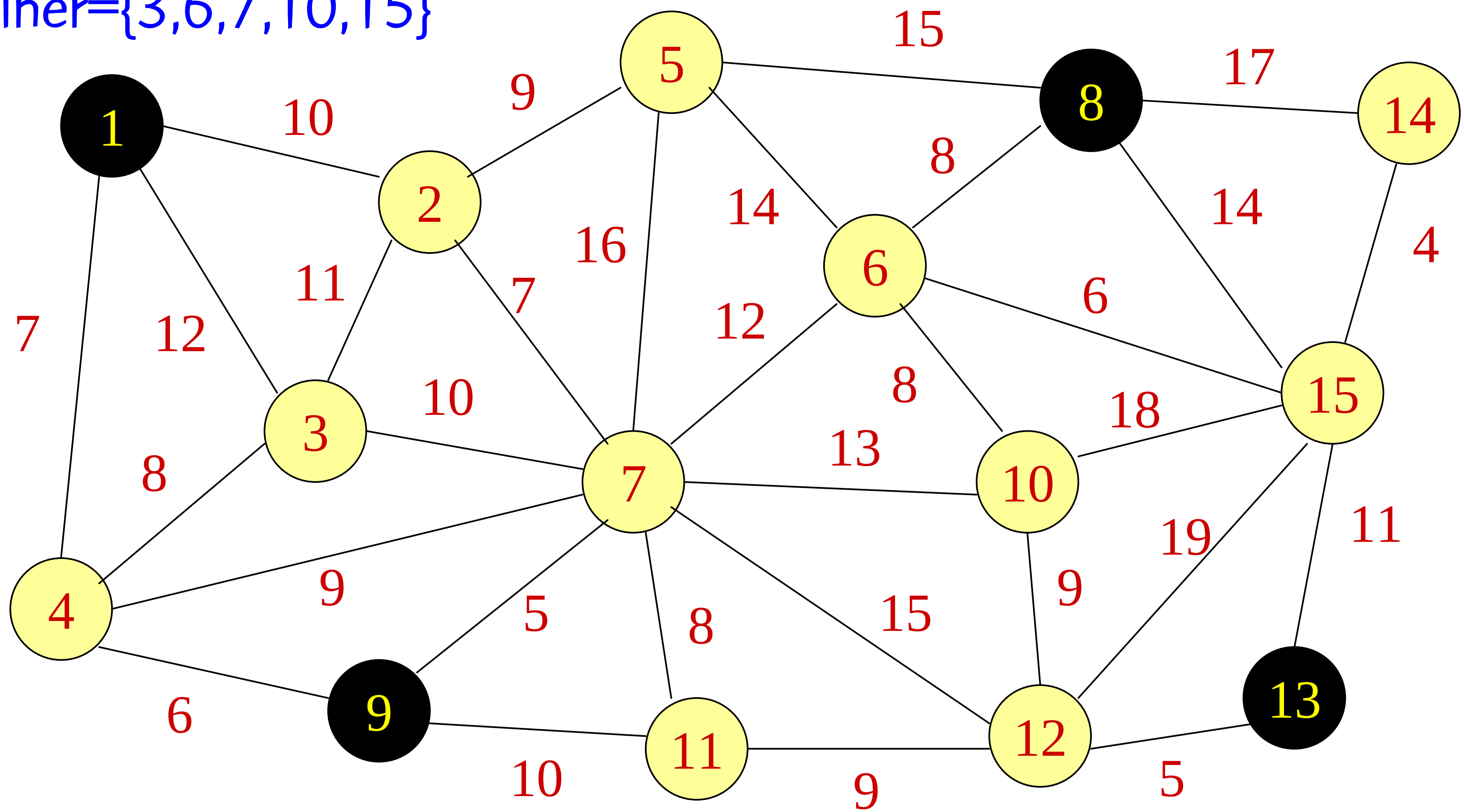
- Vértices de Steiner: vértices opcionais que fazem parte da solução ótima
- Dados os vértices obrigatórios (terminais) e um conjunto de vértices opcionais (vértices de Steiner), como calcular a árvore de Steiner correspondente?

Problema de Steiner em grafos



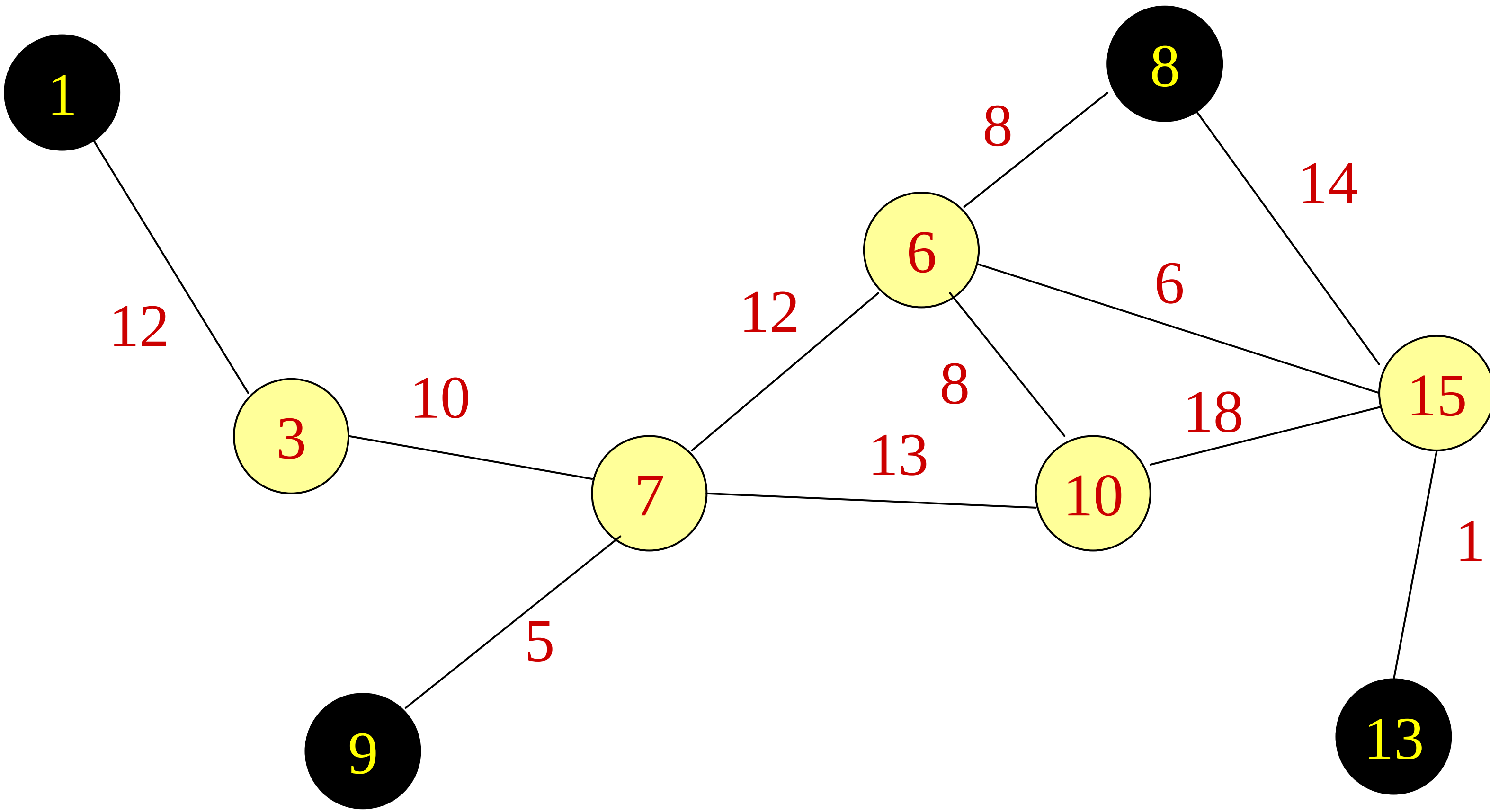
Problema de Steiner em grafos

Steiner={3,6,7,10,15}



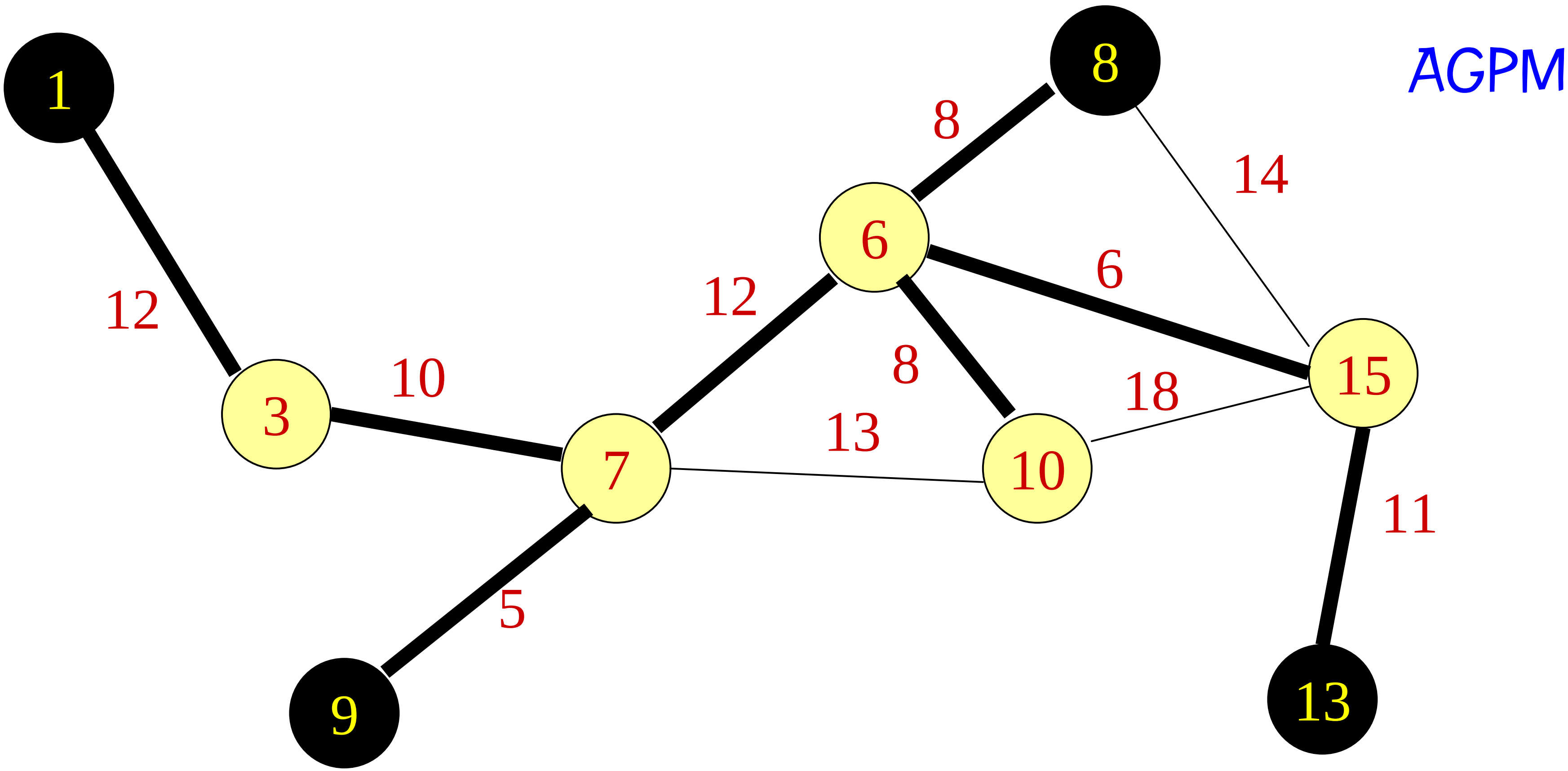
Problema de Steiner em grafos

Steiner={3,6,7,10,15}



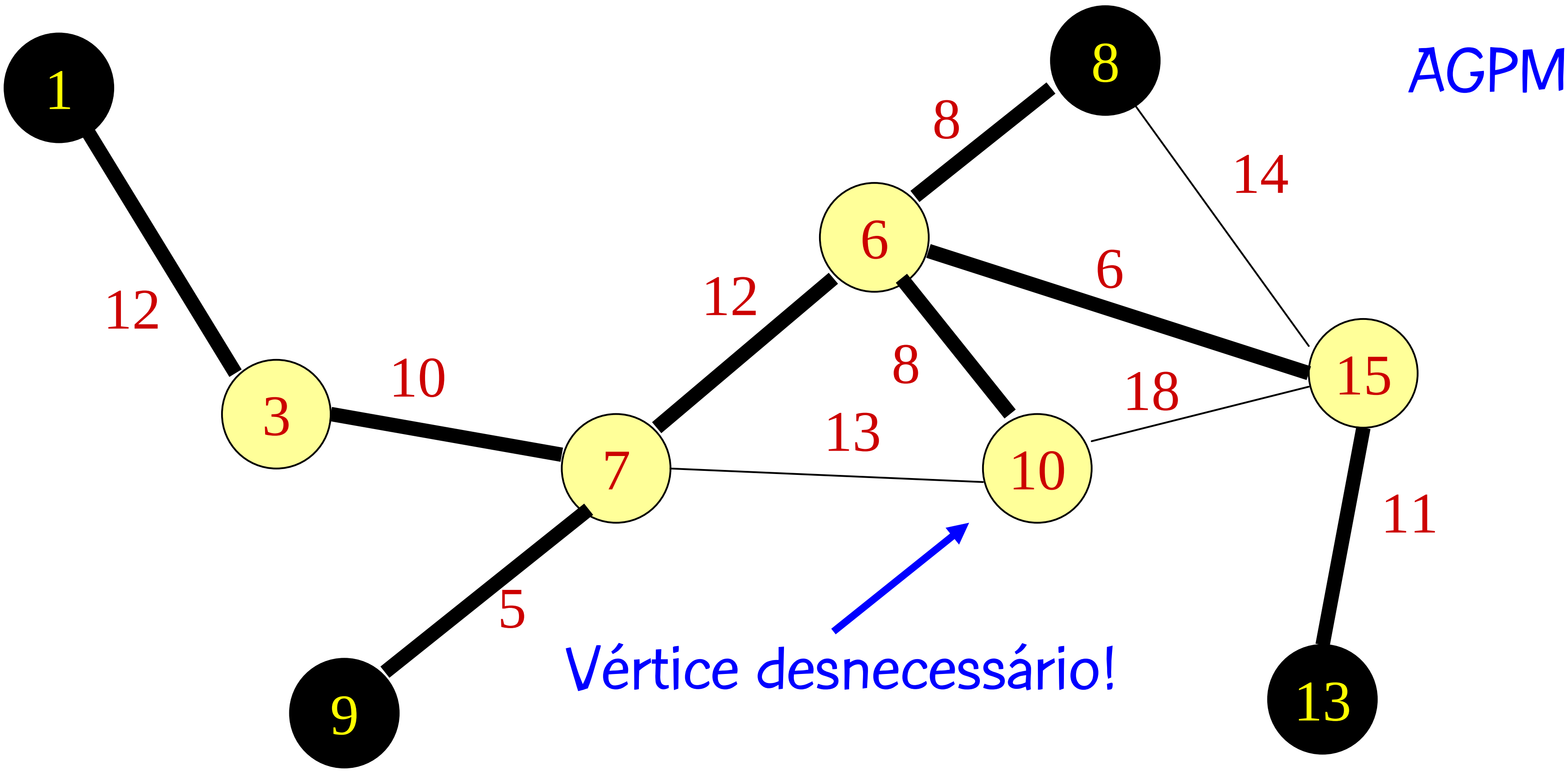
Problema de Steiner em grafos

Steiner={3,6,7,10,15}



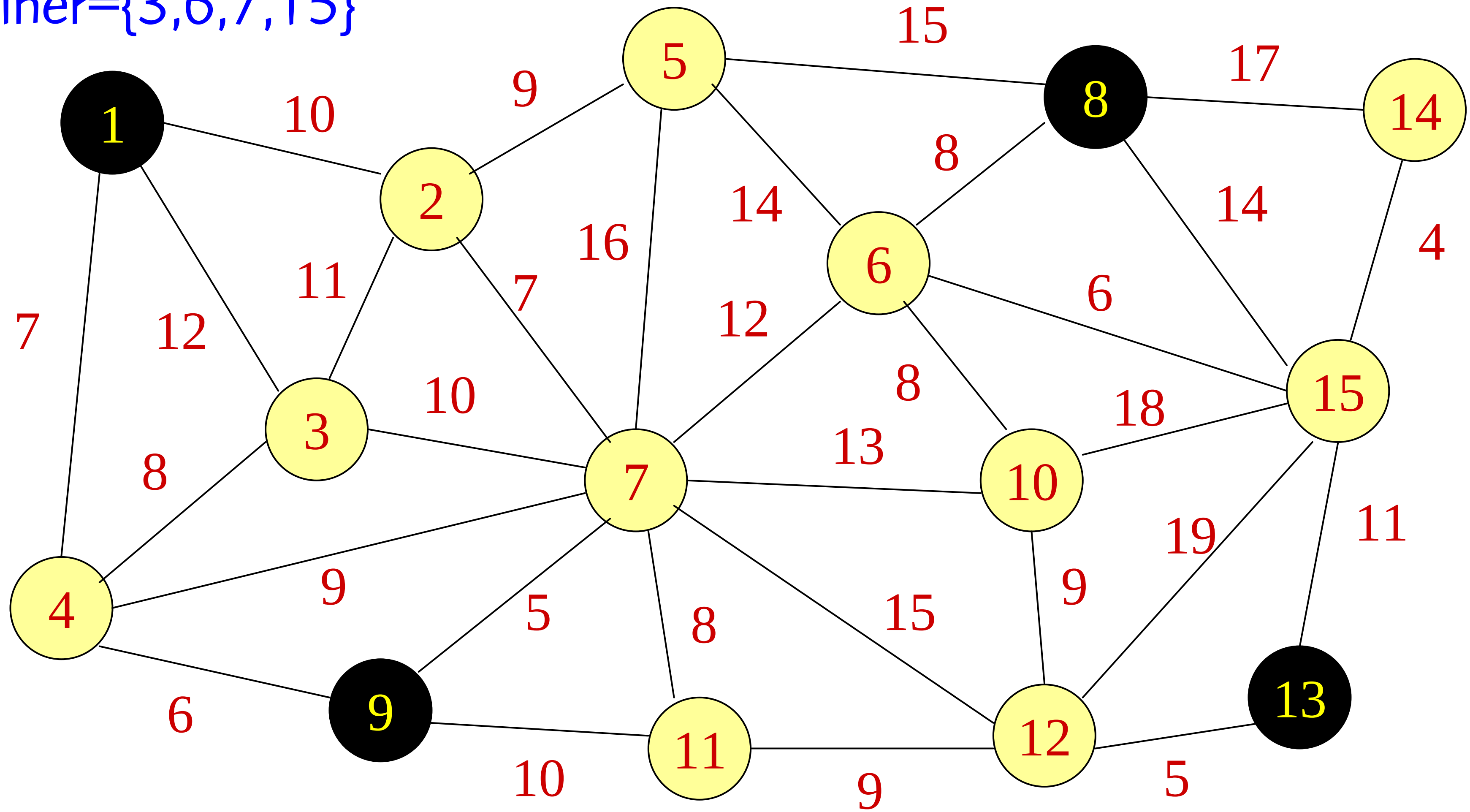
Problema de Steiner em grafos

Steiner={3,6,7,10,15}



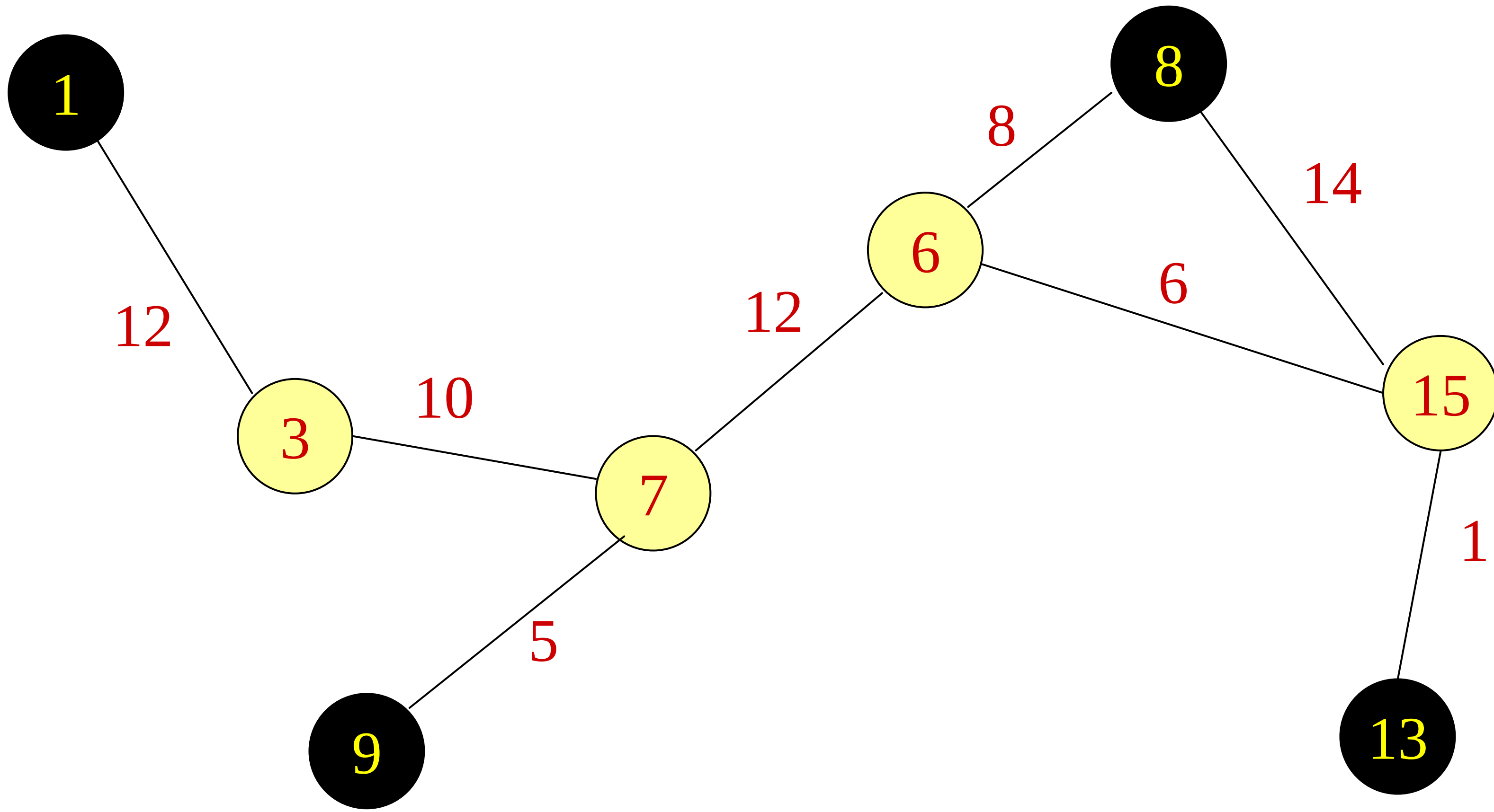
Problema de Steiner em grafos

Steiner={3,6,7,15}



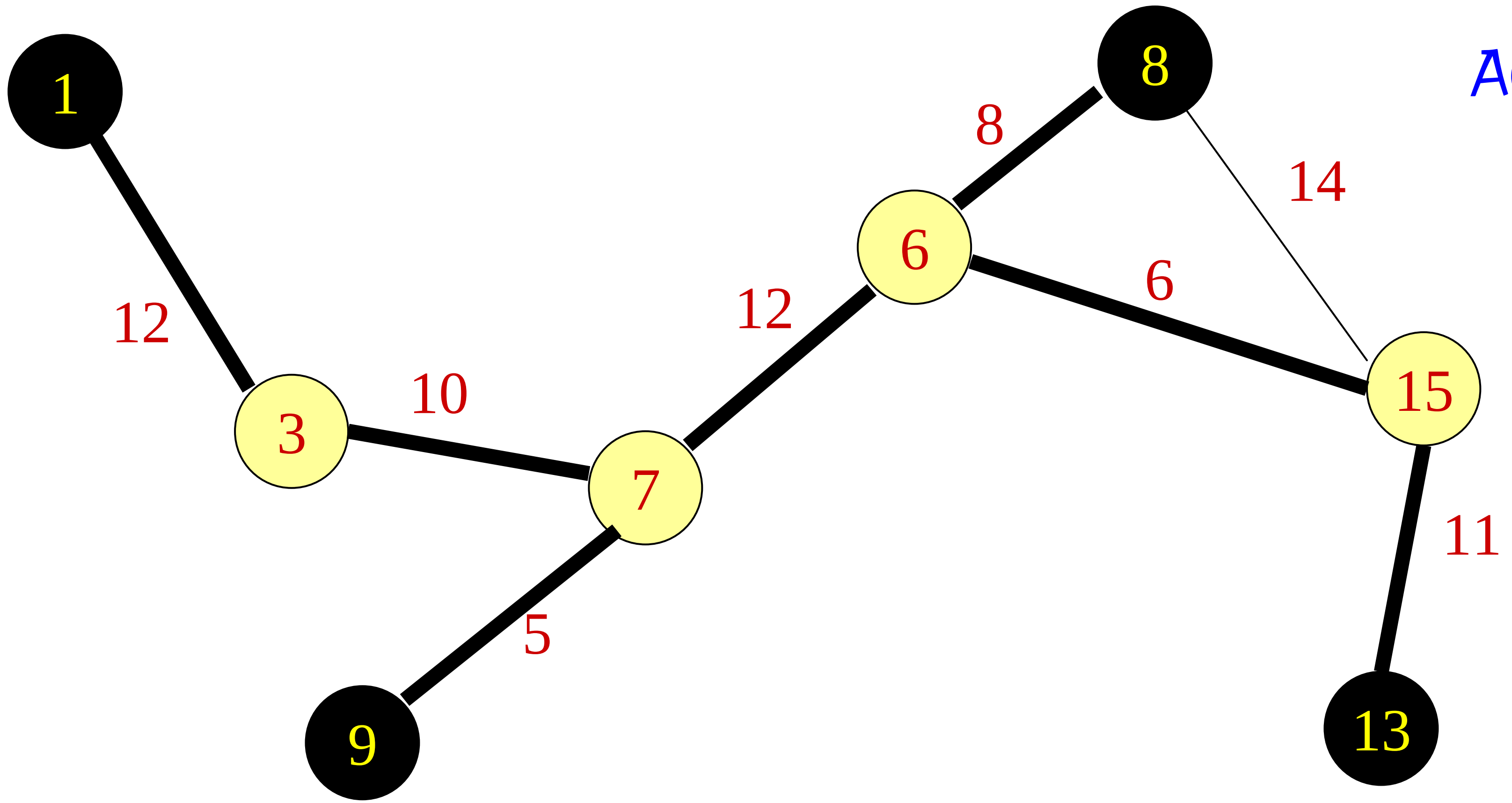
Problema de Steiner em grafos

Steiner={3,6,7,15}



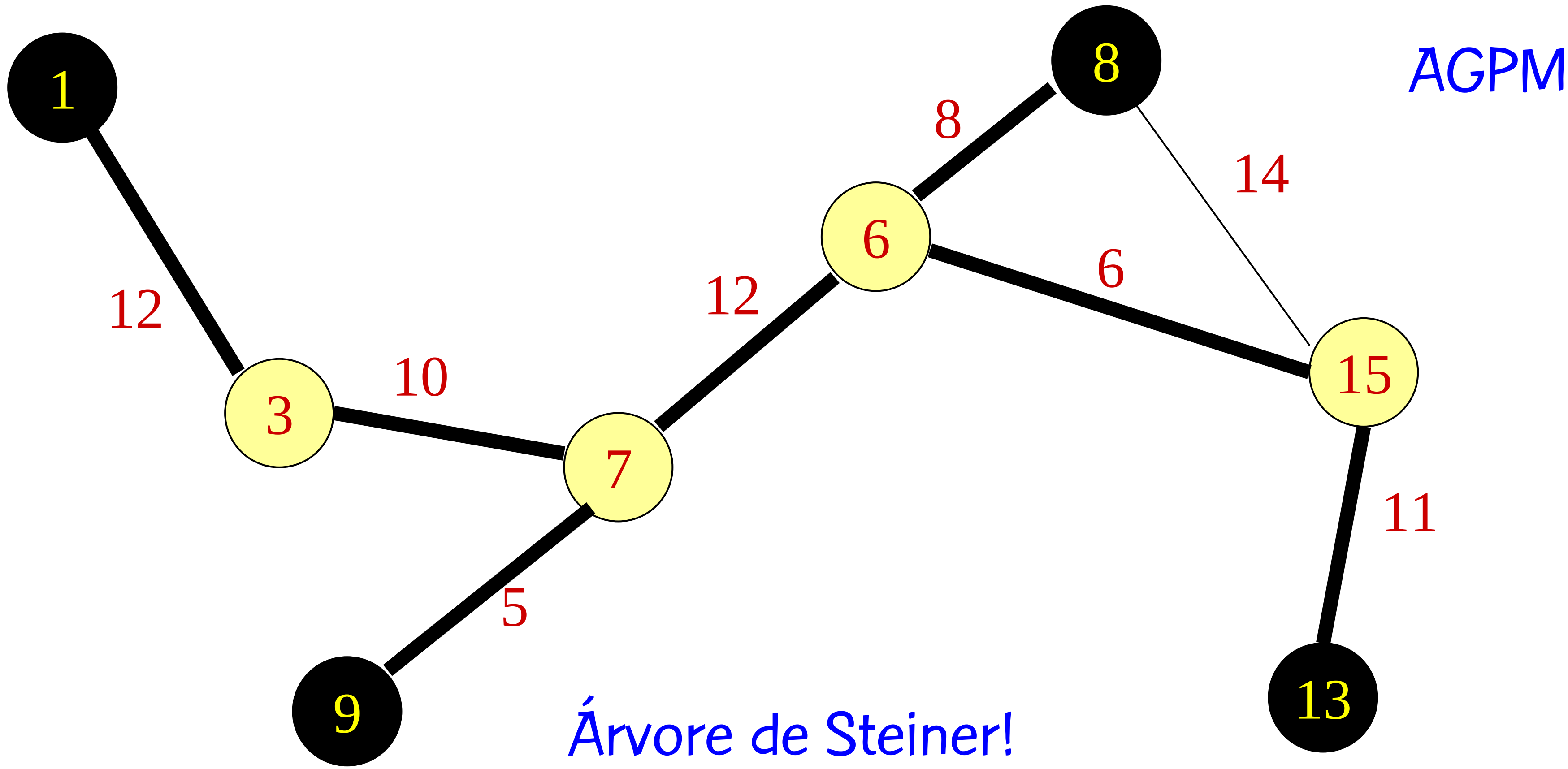
Problema de Steiner em grafos

Steiner={3,6,7,15}



Problema de Steiner em grafos

Steiner={3,6,7,15}



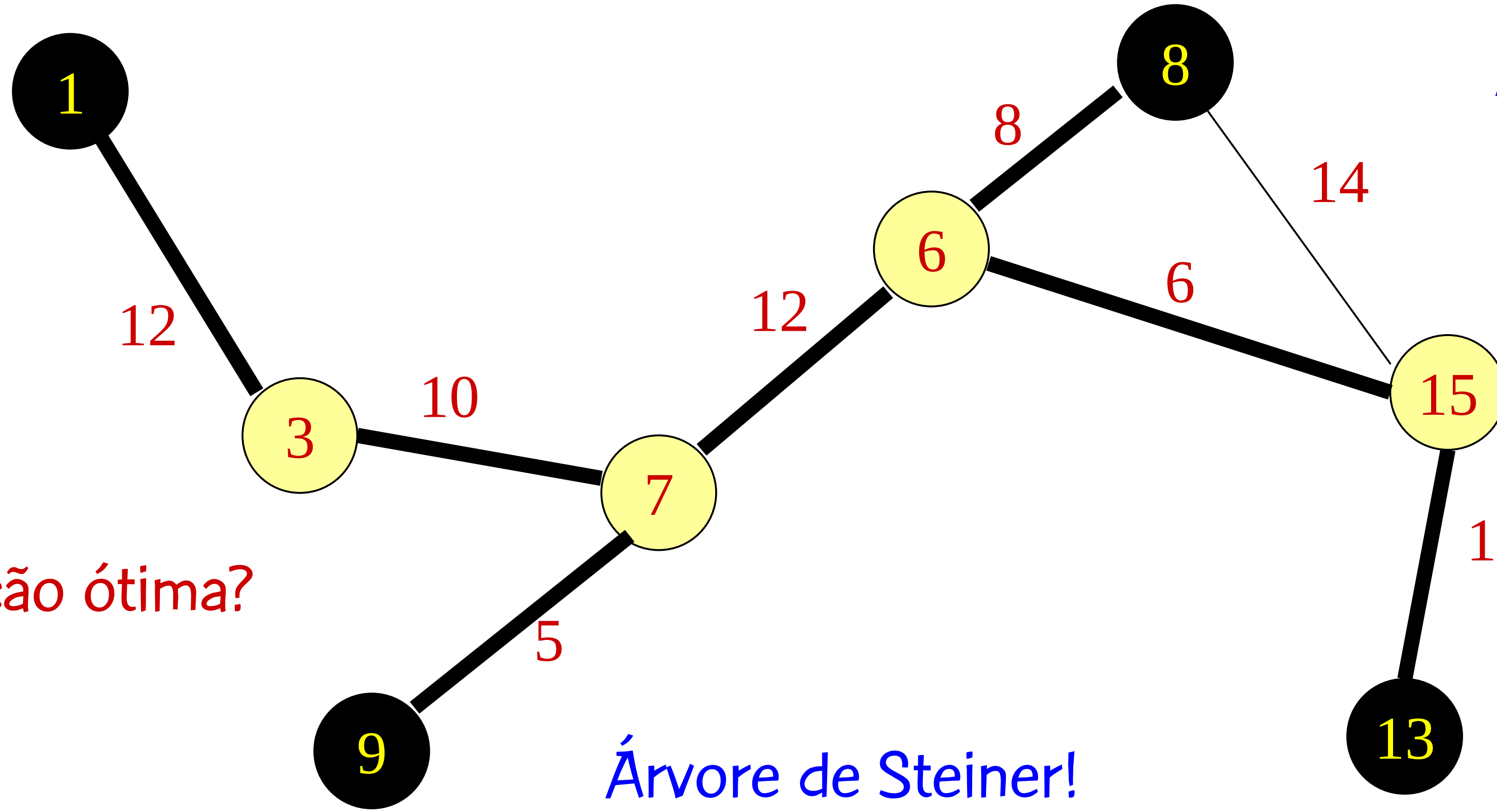
Árvore de Steiner!

custo = 64

Problema de Steiner em grafos

Steiner={3,6,7,15}

AGPM



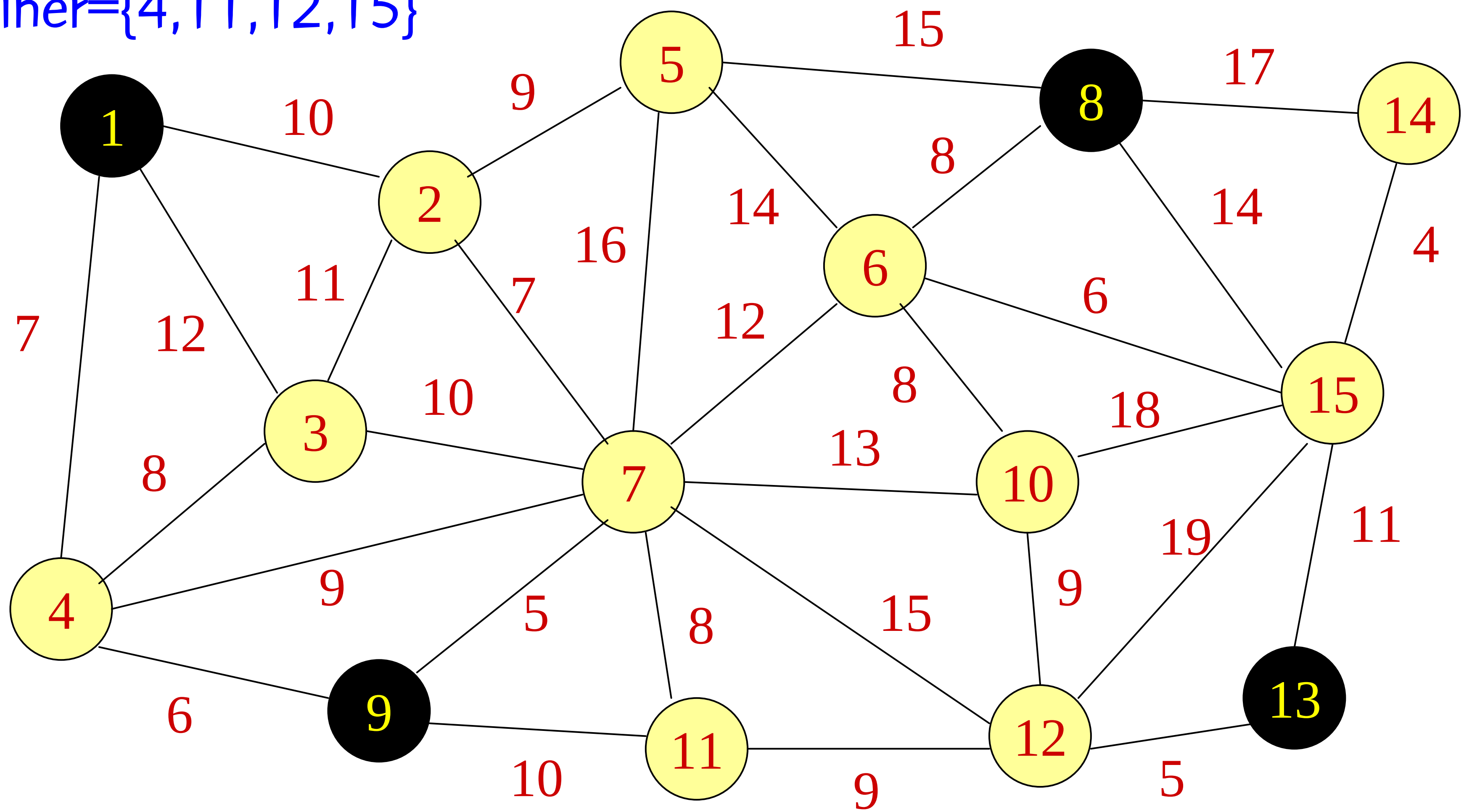
Solução ótima?

Árvore de Steiner!

custo = 64

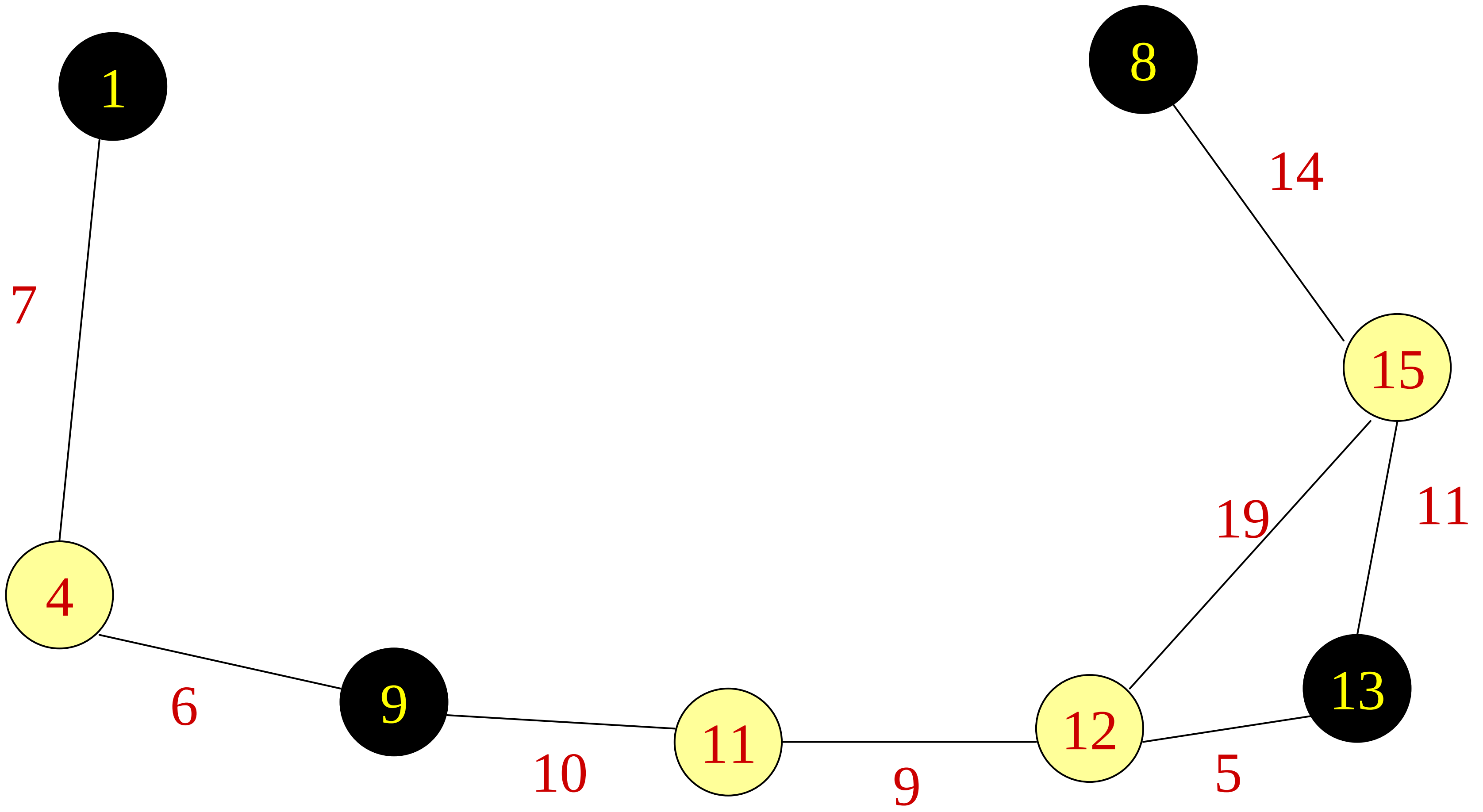
Problema de Steiner em grafos

Steiner={4,11,12,15}



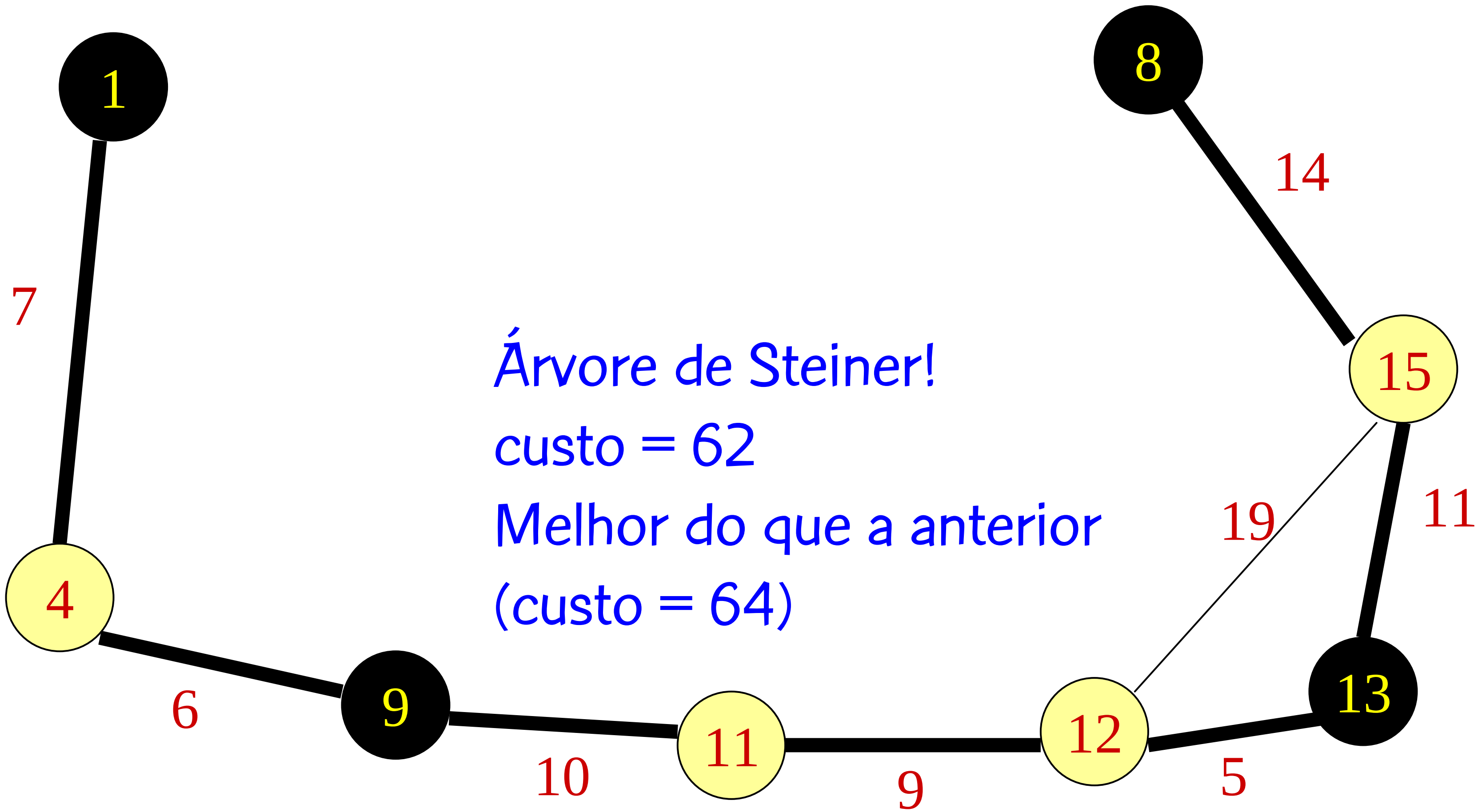
Problema de Steiner em grafos

Steiner={4,11,12,15}



Problema de Steiner em grafos

Steiner={4,11,12,15}



Árvore de Steiner!

custo = 62

Melhor do que a anterior

(custo = 64)

Problema de Steiner em grafos

- Logo, uma solução do problema de Steiner pode ser representada:
  - Explicitamente, pelo conjunto de nós opcionais utilizados
  - Completando-se implicitamente a solução, calculando-se a árvore geradora de peso mínimo do sugrafo gerado pela união dos nós obrigatórios e dos nós opcionais

Busca local

- Diferentes aspectos do espaço de busca influenciam o desempenho da busca local:
  - Conexidade: deve existir um caminho no espaço de busca entre qualquer par de soluções
  - Distância: número de soluções visitadas ao longo do caminho mais curto entre duas soluções
  - Diâmetro: distância entre as duas soluções mais afastadas (diâmetros reduzidos)
  - Bacia de atração de um ótimo local: conjunto de soluções iniciais a partir das quais o algoritmo de descida mais rápida leva a este ótimo local

Busca local

- Dificuldades e problemas do método de busca local:
  - Término no primeiro ótimo local encontrado
  - Sensível à solução de partida
  - Sensível à vizinhança escolhida
  - Sensível à estratégia de busca
  - Pode exigir um número exponencial de iterações!
- Como melhorar seu desempenho?

Busca local

- Extensões e melhorias:
  - Multipartida: iniciar a busca por diferentes soluções iniciais.
  - Redução da vizinhança: explorar parcialmente a vizinhança (aleatorização, soluções mais promissoras, etc.).
  - Listas de candidatos: recalculadas periodicamente.
  - Multivizinhanças: explorar novas vizinhanças, por exemplo após atingir um ótimo local segundo uma delas. Ao atingir um ótimo local, inicia outra busca local usando outra vizinhança: termina quando a solução corrente é um ótimo local em relação a todas as vizinhanças empregadas.
  - ...
  - Metaheurísticas: estratégias para escapar de ótimos locais.

Metaheurísticas

- Diferentes estratégias para percorrer o espaço de busca e para escapar de ótimos locais
- Quase todas utilizam-se de diferentes maneiras dos mesmos componentes básicos comuns:
  - construções gulosas
  - randomização
  - busca local
  - multiplicidade de vizinhanças
  - memória
  - intensificação
  - diversificação, etc.

Metaheurísticas

- Metaheurísticas:
  - *Simulated annealing*
  - Busca tabu
  - GRASP
  - VNS *(Variable Neighborhood Search)*
  - Algoritmos genéticos
  - *Scatter search*
  - Colônias de formigas
- Freqüentemente inspiradas em paradigmas da natureza (processos físicos e biológicos)

GRASP

- GRASP: Greedy Randomized Adaptive Search Procedures
- Princípio: combinação de um algoritmo construtivo (randomizado) com busca local, em um procedimento iterativo com iterações independentes

  (a) construção de uma solução

  (b) busca local

  (c) atualização da melhor solução

GRASP

- Algoritmo básico (minimização):

$f(s^*) \leftarrow +\infty$

Para i = 1,...,MaxIterações faça:

    Construir uma solução s usando um algoritmo guloso randomizado

    Aplicar um procedimento de busca local a partir de s, obtendo a solução s'

    Se $f(s') < f(s^*)$, então fazer $s^* \leftarrow s'$

Fim-enquanto

GRASP

- Fase de construção: aplicar um algoritmo guloso randomizado
- Cada iteração da fase de construção:
  - Avaliar o benefício de cada elemento fora da solução, usando uma função gulosa.
  - Criar uma lista de candidatos formada pelos melhores elementos.
  - Selecionar aleatoriamente um elemento da lista de candidatos
  - Adaptar a função gulosa considerando o elemento incluído

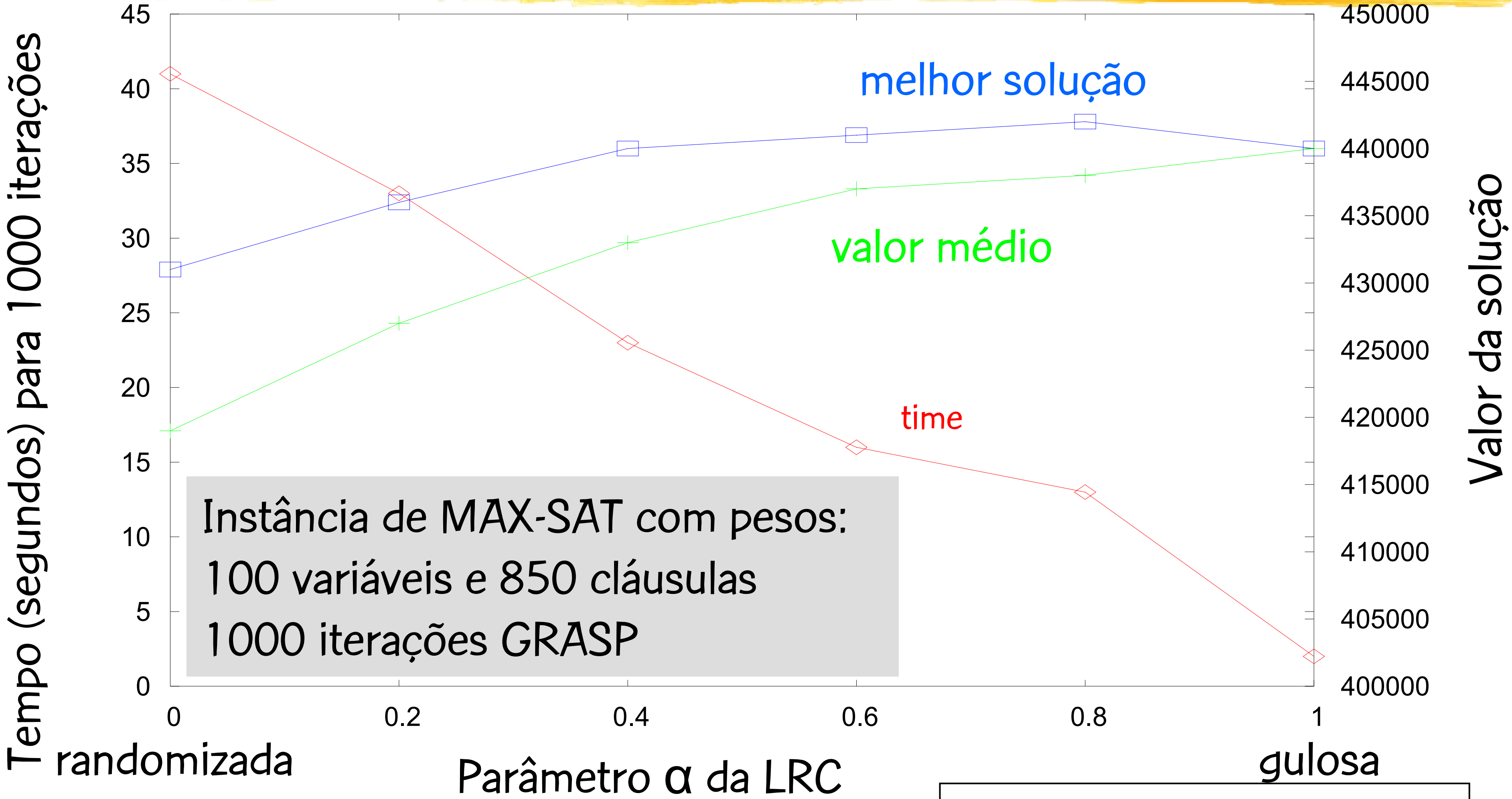
GRASP

- Construção gulosa randomizada:
  - Solução $\leftarrow \emptyset$
  - Avaliar os custos incrementais dos elementos candidatos
  - Enquanto Solução não é completa faça:
    - Construir a lista restrita de candidatos (LRC).
    - Selecionar um elemento s de LRC aleatoriamente.
    - Solução $\leftarrow$ Solução $\cup$ {s}
    - Reavaliar os custos incrementais.
  - endwhile

GRASP

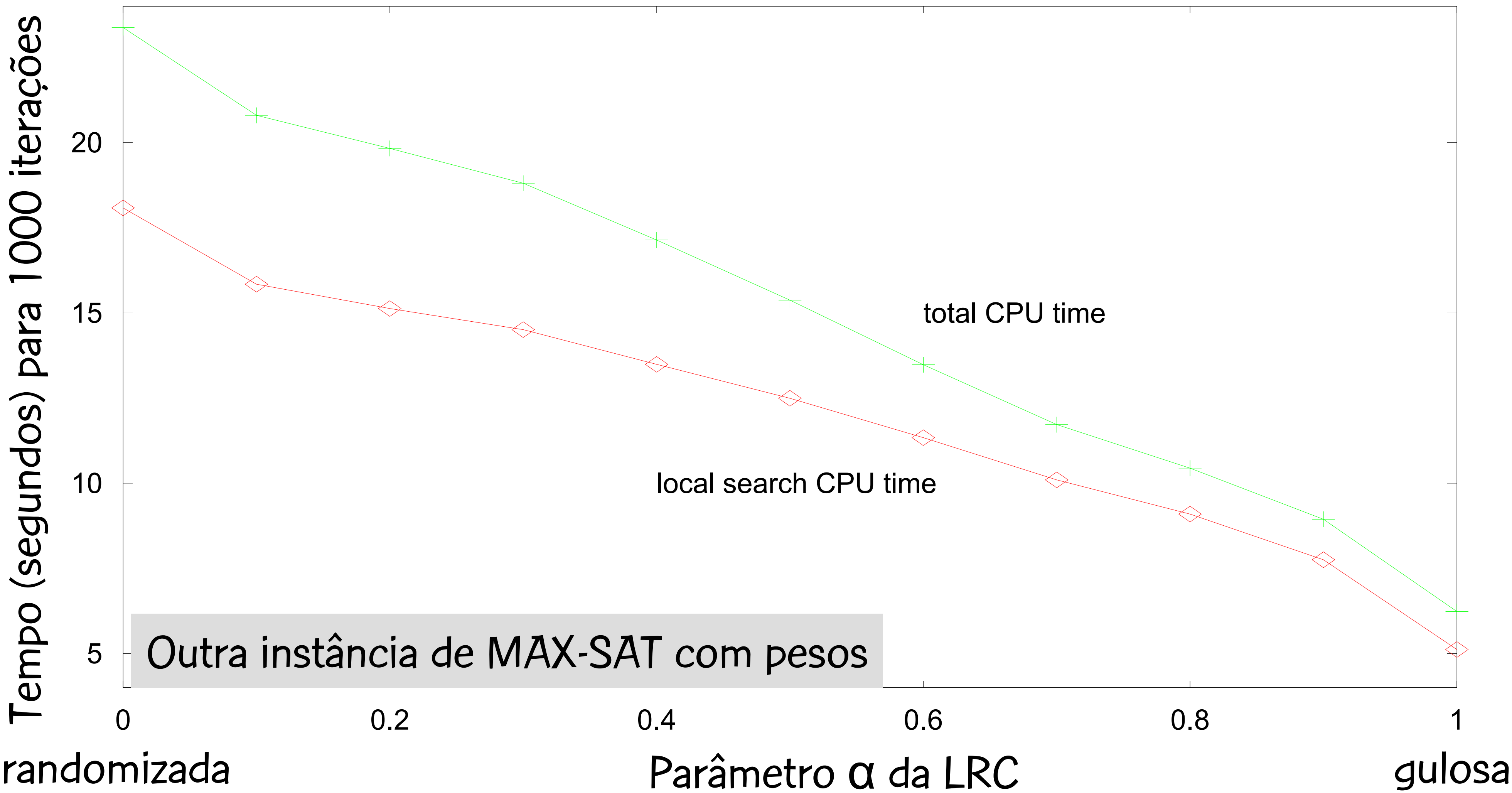
- Restrição dos elementos na lista de candidatos baseada:
  - ... no número máximo de elementos na lista
  - ... na qualidade dos elementos na lista (em relação à escolha puramente gulosa)
- Seleção aleatória feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como no caso guloso):
  - A qualidade média das soluções encontradas depende da qualidade dos elementos na lista.
  - A diversidade das soluções construídas depende do número de elementos na lista.

Parâmetro da LRC



SGI Challenge 196 MHz

Parâmetro da LRC



SGI Challenge 196 MHz

GRASP

- Construção com base na cardinalidade:
  - p elementos com menores custos incrementais
- Construção com base na qualidade:
  - Parâmetro α define a qualidade dos elementos na LRC.
  - RCL (min) contém elementos com custos incrementais $c^{min} \leq c(e) \leq c^{min} + \alpha\ (c^{max} - c^{min})$
  - $\alpha = 0$: construção puramente gulosa
  - $\alpha = 1$: construção puramente randomizada
- Selecionar aleatoriamente da LRC usando uma distribuiçao de probabilidade uniforme

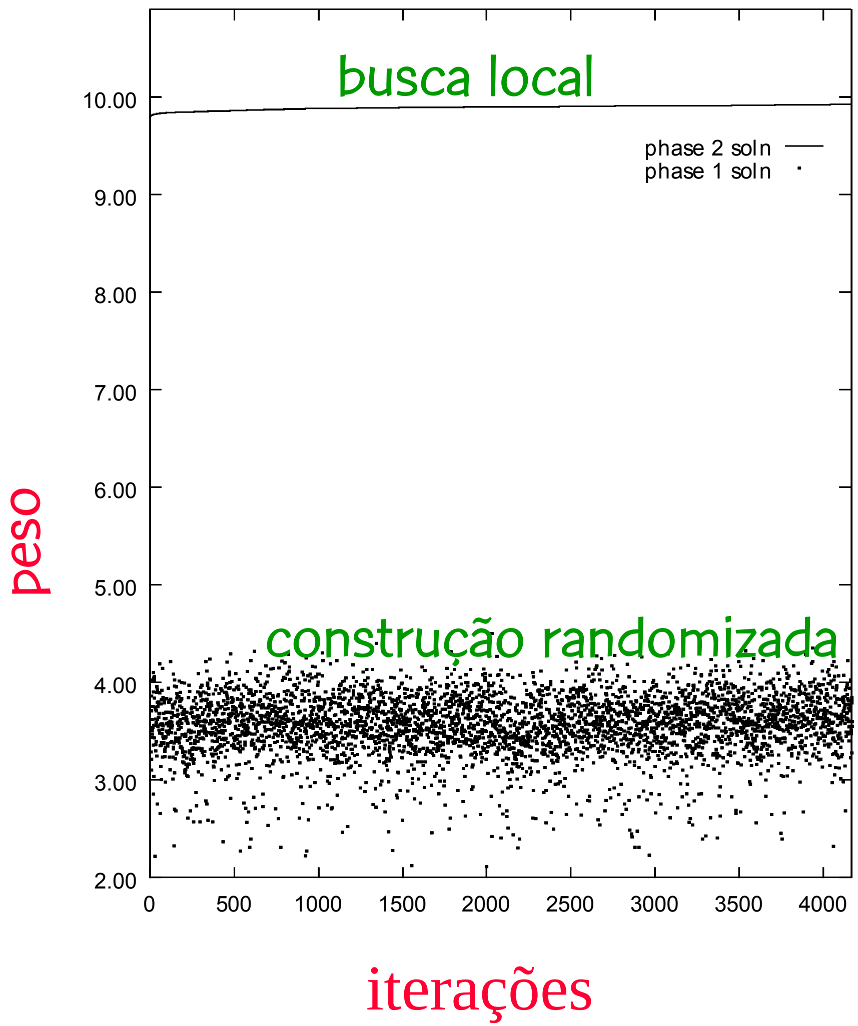
GRASP

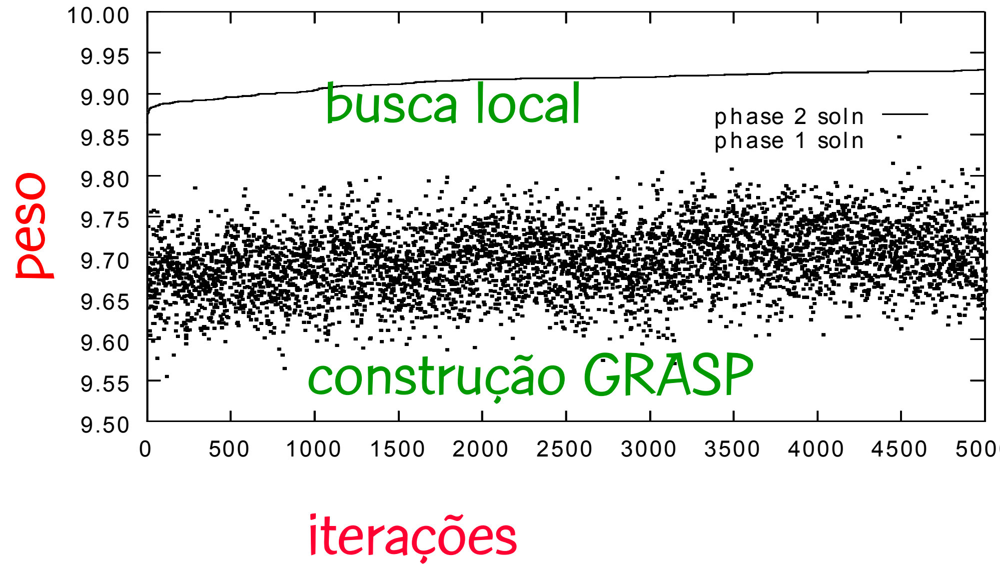
- Diversificação baseada em randomização controlada: diferentes soluções construídas em diferentes iterações GRASP.
- Fase de busca local: melhorar as soluções construídas
  - Escolha da vizinhança
  - Estruturas de dados eficientes para acelerar a busca local
  - Boas soluções iniciais permitem acelerar a busca local

GRASP

- A utilização de um algoritmo guloso randomizado na fase de construção permite acelerar muito cada aplicação da busca local (soluções próximas de um ótimo local).
- É comprovadamente mais rápido e encontra soluções melhores do que um método multi-partida simples.

GRASP



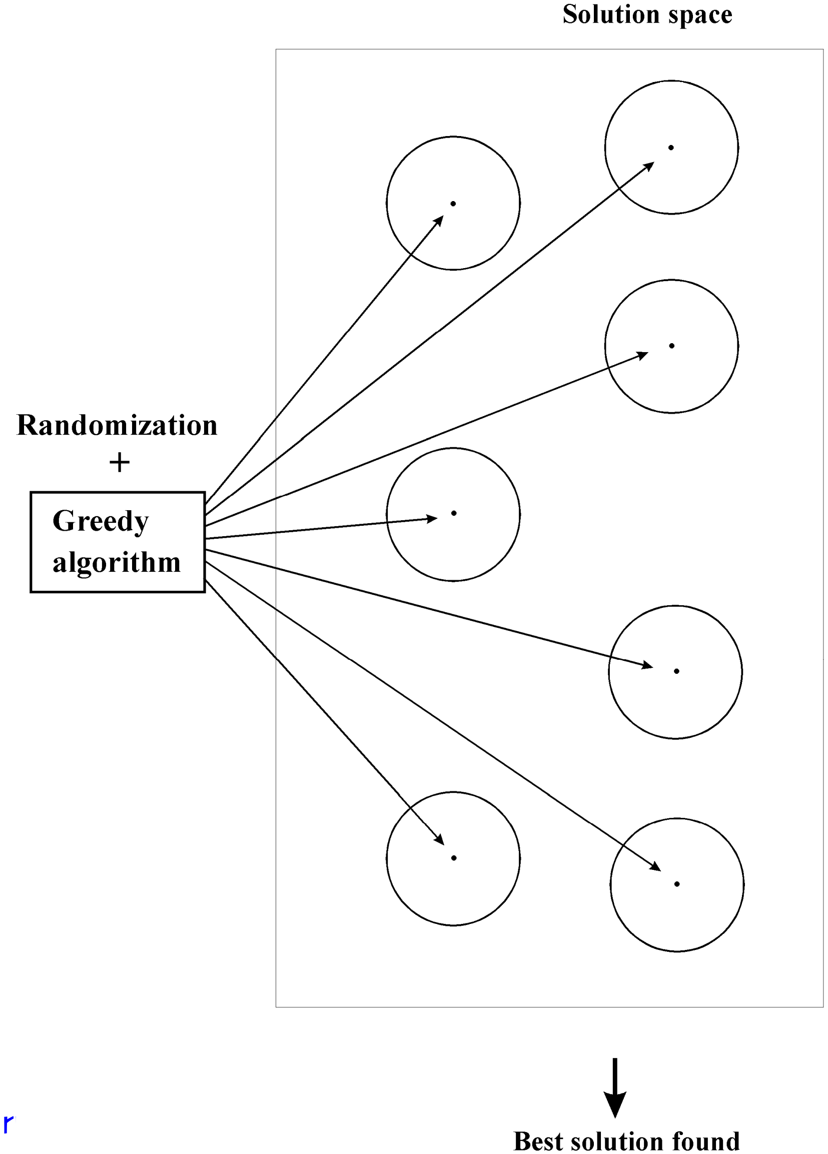


Eficiência da construção gulosa randomizada vs puramente randomizada:

Aplicação: colocação de modens (problema de cobertura com peso máximo) ($\alpha = 0.85$)

GRASP

- Pode ser visto como uma técnica de amostragem no espaço de busca:



GRASP

- Implementação simples: algoritmo guloso e busca local
- Heurísticas gulosas são simples de projetar e implementar.
- Poucos parâmetros a serem ajustados:
  - restritividade da lista de candidatos
  - número de iterações
- Depende de boas soluções iniciais: baseado apenas na randomização de uma iteração para outra, cada iteração se beneficia da qualidade de sua solução inicial.

GRASP

- Problema: falta de "memória"
  - Em uma determinada iteração, não são utilizadas informações sobre as soluções construídas e visitadas nas iterações precedentes.
  - Esta dificuldade pode ser superada por implementações mais avançadas, que incorporam memória:
    - GRASP reativo: controle automático do nível apropriado de randomização durante a fase de construção
    - Reconexão por caminhos: explorar trajetórias entre a solução atual e boas soluções construídas nas iterações precedentes

GRASP com reconexão por caminhos

- Mantém um conjunto de soluções de elite encontradas durante as iterações GRASP precedentes.
- Após cada iteração GRASP (construção e busca local):
  - Usar a solução GRASP como solução inicial.
  - Selecionar aleatoriamente uma solução de elite: solução guia (também pode ser selecionada com probabilidades proporcionais aos custos ou à diferença simétrica em relação à solução inicial).
  - Executar reconexão por caminhos entre as duas soluções.

GRASP com reconexão por caminhos

- Repetir ao longo de Max_Iterations:
  - Construir uma solução gulosa randomizada.
  - Usar busca local para melhorar a solução construída.
  - Aplicar reconexão por caminhos para melhorar a solução corrente.
  - Atualizar o conjunto de soluções de elite.
  - Atualizar a melhor solução encontrada.

GRASP com reconexão por caminhos

- Variantes: compromisso entre tempo de processamento e qualidade de solução
  - Explorar diferentes trajetórias (backward, forward): melhores resultados começando da melhor, pois a vizinhança da solução inicial é inteiramente explorada
  - Explorar ambas trajetórias: dobro do tempo, freqüentemente apenas com ganhos marginais de qualidade
  - Não aplicar reconexão por caminhos em todas iterações, mas apenas periodicamente (similar à filtragem na busca local)
  - Truncar a busca, não seguir toda a trajetória
  - Também pode ser aplicado como uma estratégia de pós-otimização a todos a todos os pares de soluções de elite

GRASP com reconexão por caminhos

- Aplicações bem sucedidas:

1) Problema de Steiner
2) Problema de Steiner com prêmios
3) Problema das p-medianas
4) Árvore geradora de peso mínimo com capacidades
5) Projeto de redes com 2-caminhos
6) Corte máximo
7) Problema quadrático de atribuição
8) Seqüenciamento de tarefas
9) Roteamento de circuitos virtuais privados
10) Árvores filogenéticas

GRASP com reconexão por caminhos

- P é um conjunto de soluções de elite.
- Cada uma das |P| primeiras iterações GRASP adiciona uma nova solução a P (se diferente das demais).
- Após as |P| primeiras iterações: a solução x é promovida a P se:
  - x é melhor do que a melhor solução em P.
  - x não é melhor do que a melhor solução em P, mas é melhor do que a pior e é suficientemente diferente de todas as soluções em P.

GRASP

- Extensões e melhorias:
  - Uso de filtros: aplicar busca local apenas...
  - ... à melhor solução construída ao longo de uma seqüência de aplicações do algoritmo guloso randomizado (já que a busca local é a componente mais cara em termos de tempo de processamento).
  - ... às soluções construídas que satisfazem um determinado limiar de aceitação (soluções potencialmente boas).
  - Substituir busca local por busca tabu curta
- Processamento paralelo: a estrutura do algoritmo é altamente favorável a implementações paralelas eficientes com acelerações lineares em clusters.

Algoritmos genéticos

- Algoritmo probabilístico baseado na analogia com o processo de seleção natural e evolução.
- Pertence à classe dos métodos populacionais, que consideram simultaneamente uma população de soluções, e não apenas uma única solução.
- Durante a evolução, populações evoluem de acordo com os princípios de seleção natural e de "sobrevivência dos mais adaptados" (evolução das propriedades genéticas da população).

Algoritmos genéticos

- Indivíduos mais bem sucedidos em adaptar-se a seu ambiente têm mais chances de sobreviver e de se reproduzirem: genes dos indivíduos mais bem adaptados vão espalhar-se para um maior número de indivíduos em sucessivas gerações.
- Espécies evoluem tornando-se cada vez mais adaptadas ao seu ambiente, geração após geração.
- População representada pelos seus cromossomos:
  cromossomo $\Leftrightarrow$ indivíduo (solução)
  aptidão $\Leftrightarrow$ função objetivo

Algoritmos genéticos

- Novos cromossomos (ou indivíduos, ou soluções) são gerados a partir da população corrente e incluídos na população, enquanto outros são excluídos.
- A geração de novos cromossomos é feita através de mecanismos de reprodução e de mutação.
- Representação básica de um cromossomo: sequência de bits 0-1, conforme uma característica esteja presente ou não (forma mais simples, há outros modelos e representações).

Algoritmos genéticos

- Reprodução: selecionar os cromossomos (indivíduos) pais e executar uma operação de cruzamento, que é uma combinação simples das representações de cada cromossomo (indivíduo).
  - Os filhos recebem parte do material genético de cada pai.
- Mutação: modificacão arbitrária de uma parte (pequena) do cromossomo

Algoritmos genéticos

- Algoritmo básico:

```
Gerar uma população inicial.
Enquanto o critério de parada não for satisfeito faça:
    Escolher cromossomos (indivíduos) reprodutores.
    Fazer o cruzamento dos reprodutores.
    Gerar mutações da população atual.
    Avaliar a aptidão dos novos indivíduos gerados.
    Atualizar a população, eliminando os menos adaptados.
Fim
```

Algoritmos genéticos

- Principais parâmetros e decisões de implementação:
  - representação das soluções
  - tamanho da população
  - critério de seleção dos reprodutores
  - operação de reprodução
  - taxa de mutação
  - operação de mutação
  - critério de sobrevivência (seleção) dos cromossomos
  - critério de parada (estabilização da população, impossibilidade de melhorar a melhor solução, número de gerações)

Algoritmos genéticos

- <u>População inicial</u>:
  - Originalmente, soluções geradas aleatoriamente.
    - Problema: tempos de processamento elevados (soluções ruins) ou dificuldade para encontrar soluções viáveis
  - Utilizar heurísticas, por exemplo algoritmos gulosos aleatorizados.
  - Má qualidade das soluções da população inicial pode levar a soluções ruins ou tempo de convergência excessivo...
    - ... mas diversidade é necessária!

Algoritmos genéticos

- Função de aptidão:
  - Quantifica a qualidade genética dos cromossomos.
  - Tipicamente, corresponde à função objetivo em problemas de otimização.
    - Pode envolver outros termos que, por exemplo, penalizem a inviabilidade de algumas soluções.
  - Utilizada para:
    - ... selecionar os indivíduos reprodutores.
    - ... selecionar os sobreviventes de uma geração para outra.

Algoritmos genéticos

- Cruzamento: operação probabilística (originalmente), onde os indivíduos mais adaptados têm maior chance de participar.
- Exemplo: soluções (cromossomos) com 10 elementos

  Pai 1: (0,1,0,1,1,1,1,0,0,1)

  Pai 2: (0,0,1,1,0,1,0,0,1,1)

  Alguns filhos possíveis por cruzamento:

  (0,1,0,1,1,1,0,0,1,1)

  (0,1,0,1,0,1,0,0,0,1)

  (0,0,0,1,1,1,1,0,0,1)

Algoritmos genéticos

- Cruzamento uniforme: cada bit de um filho é gerado escolhendo-se aleatoriamente um dos pais e repetindo-se o bit do pai escolhido

  $p_1 = (0,1,0,1,1,1,1,0,0,1)$

  $p_2 = \underline{(0,0,1,1,0,1,0,0,1,1) +}$

  $f_1 = (0,0,0,1,1,1,1,0,0,1)$

- Cruzamento por fusão: como o uniforme, mas a escolha aleatória do pai a ter seu material propagado é feita com probabilidade proporcional à função de aptidão.

Algoritmos genéticos

- Exemplo: cruzamento combina pai p_1 com pai p_2 para produzir filho f

Pai p_1: aptidão a_1
Pai p_2: aptidão a_2

O filho herda com maior probabilidade os gens do pai com maior valor da função de aptidão.

Para todos os gens i = 1,2,...,n fazer:
    Gerar x = random[0,1]
    Se $x \leq a_1/(a_1+a_2)$
        Então $f[i] = p_1[i]$
        Senão $f[i] = p_2[i]$
fim

Algoritmos genéticos

- Cruzamento de um ponto:
  pais: $a = (a_1,\ldots,a_k,\ldots,a_n)$ e $b = (b_1,\ldots,b_k,\ldots,b_n)$
  operação: $k \in \{1,\ldots,n\}$ aleatório
  filhos: $(a_1,\ldots,a_k,b_{k+1},\ldots,b_n)$ e $(b_1,\ldots,b_k,a_{k+1},\ldots,a_n)$
  $p_1 = (0,1,0,1,1,1,1,0,0,1)$
  $p_2 = \underline{(0,0,1,1,0,1,0,0,1,1) +}$
  $s_1 = (0,1,0,1,1,1,0,0,1,1)$
  $s_2 = (0,0,1,1,0,1,1,0,0,1)$
- Cruzamento de dois (ou mais) pontos: mesma idéia, cortando em dois (ou mais) pontos

Algoritmos genéticos

- Dificuldade: como tratar restrições e soluções inviáveis?
  - Utilizar uma representação que automaticamente garanta que todas as soluções representáveis sejam viáveis.
  - Utilizar um operador heurístico de reparação que garanta a transformação de qualquer solução inviável em outra viável.
    - Exemplo: retirar itens selecionados no problema da mochila
  - Separar a avaliação da adaptação e da viabilidade
  - Utilizar uma função de penalização para penalizar a adaptação de soluções inviáveis, de modo que nunca sejam selecionadas.
- Aplicações bem sucedidas exigem que questões de viabilidade possam ser tratadas facilmente (mochila, cobertura)

Algoritmos genéticos

- Nada proíbe (recomenda-se!) a utilização de técnicas de otimização em algoritmos genéticos (mais inteligência)
- Cruzamento otimizado: aplicar idéias de otimização às operações de cruzamento
  - Algoritmos mais “inteligentes”, que não são baseados apenas em critérios probabilísticos para os cruzamentos.
  - Selecionar para cruzamento pais que otimizem a qualidade dos filhos gerados.
  - Selecionar pais por compatibilidade: selecionar um pai, em seguida escolher o segundo pai como o indivíduo mais compatível com o primeiro.
  - Aplicar busca local às soluções obtidas por cruzamento.

Algoritmos genéticos

- Utilizar consenso no crossover: repetir bits comuns aos dois reprodutores
  - Crossover por reconexão por caminhos
  - Outra maneira de fazer um crossover otimizado

Algoritmos genéticos

- Mutação: normalmente implementada com a complementação de bits de uma solução.
- Seleção aleatória dos bits a serem modificados: baixo percentual do total de bits na população de cromossomos.
- Mutações não passam por testes de aptidão:
  - Cruzamentos permitem apenas a evolução da população, conduzindo a uma população homogênea.
  - Mutação é o mecanismo usado para introduzir diversidade na população.
  - Mutação adaptativa: alterar bits para garantir viabilidade.
- Exemplo: (1,0,0,0,1,0,1,1) → (1,0,0,0,0,0,1,1)

Algoritmos genéticos

- Critérios de atualização da população podem também permitir que pais e filhos permaneçam na população, removendo-se sempre os menos aptos, sendo possível a utilização de cromossomos pais e filhos como reprodutores.

Algoritmos genéticos

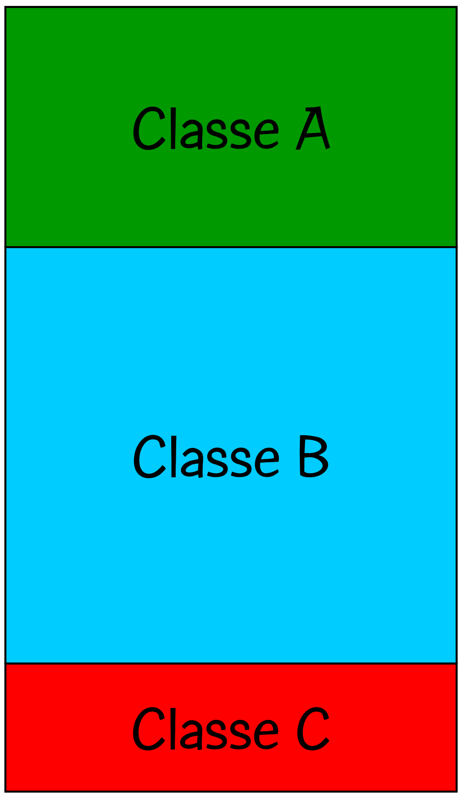
- Exemplo: mutação aplicada diretamente ao cruzamento que combina pai p_1 com pai p_2 para produzir filho f

Com probabilidade pequena, ocorre uma mutação (bit do filho é gerado aleatoriamente).

Caso contrário, o filho herda com maior probabilidade os gens do pai com maior valor da função de aptidão.

```
Para todos os gens i = 1,2,...,n fazer:
    Gerar x = rrandom[0,1]
    Se x ≤ ProbMutação
    Então f[i] = irandom[0,1]
    Senão  Se x ≤ a1/(a1+a2)
               Então f[i] = p1[i]
               Senão f[i] = p2[i]
Fim
```

AGs com chaves aleatórias



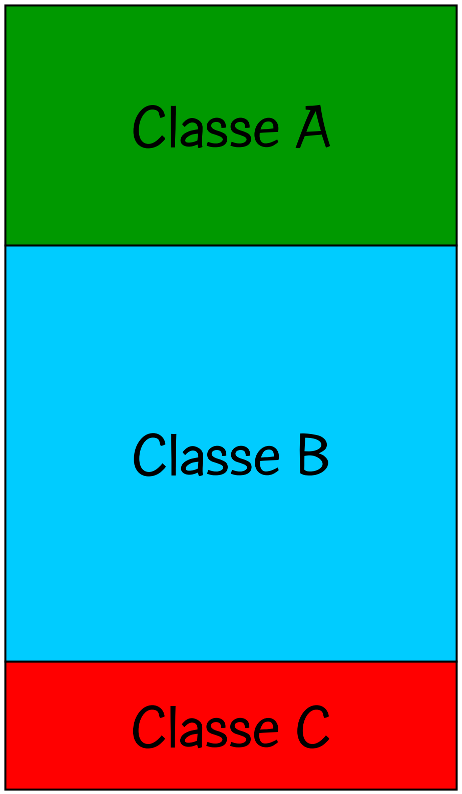
25% mais adaptados

População ordenada de acordo com a função de avaliação e soluções classificadas em três categorias.

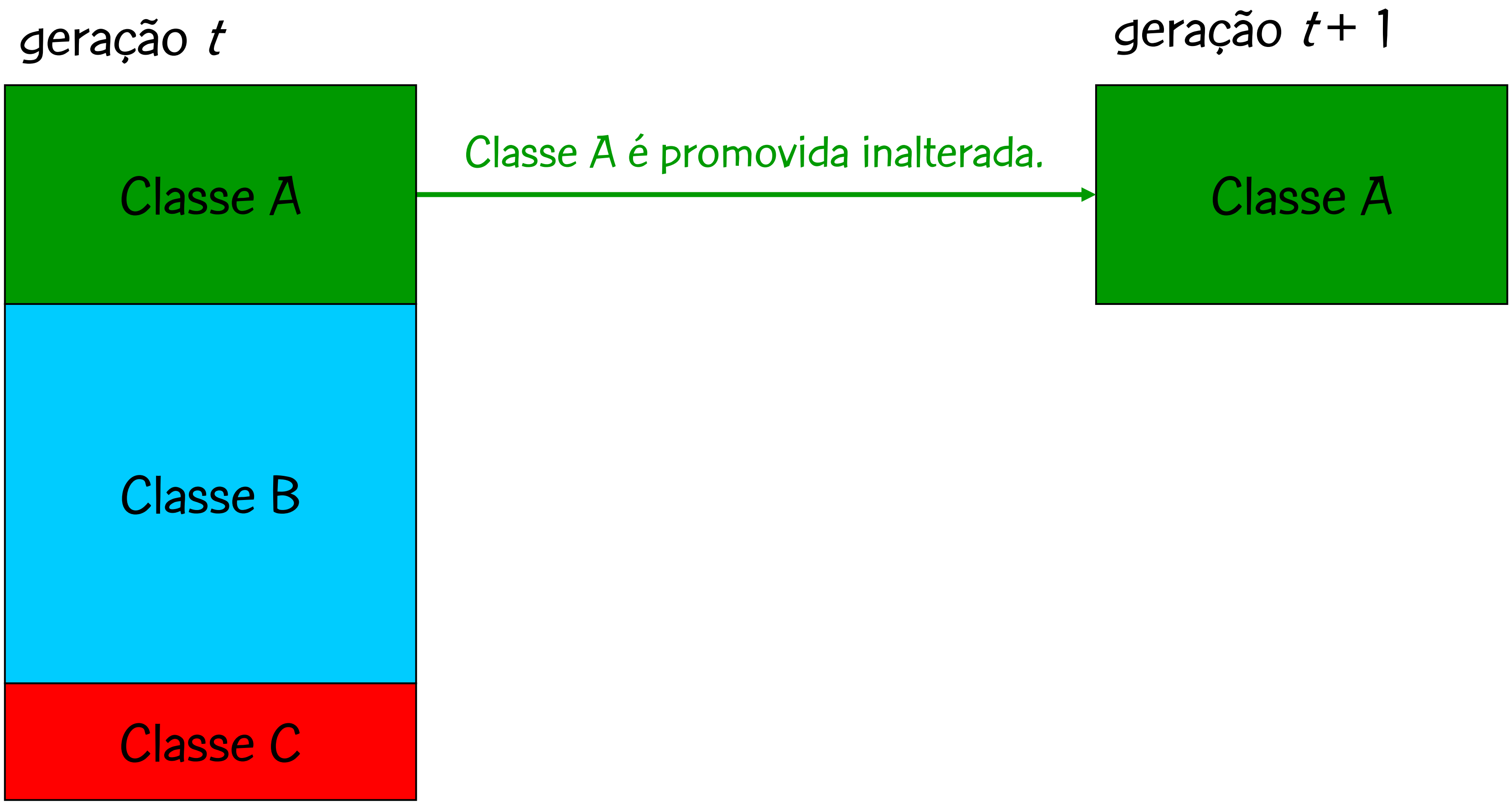
5% menos adaptados

AGs com chaves aleatórias

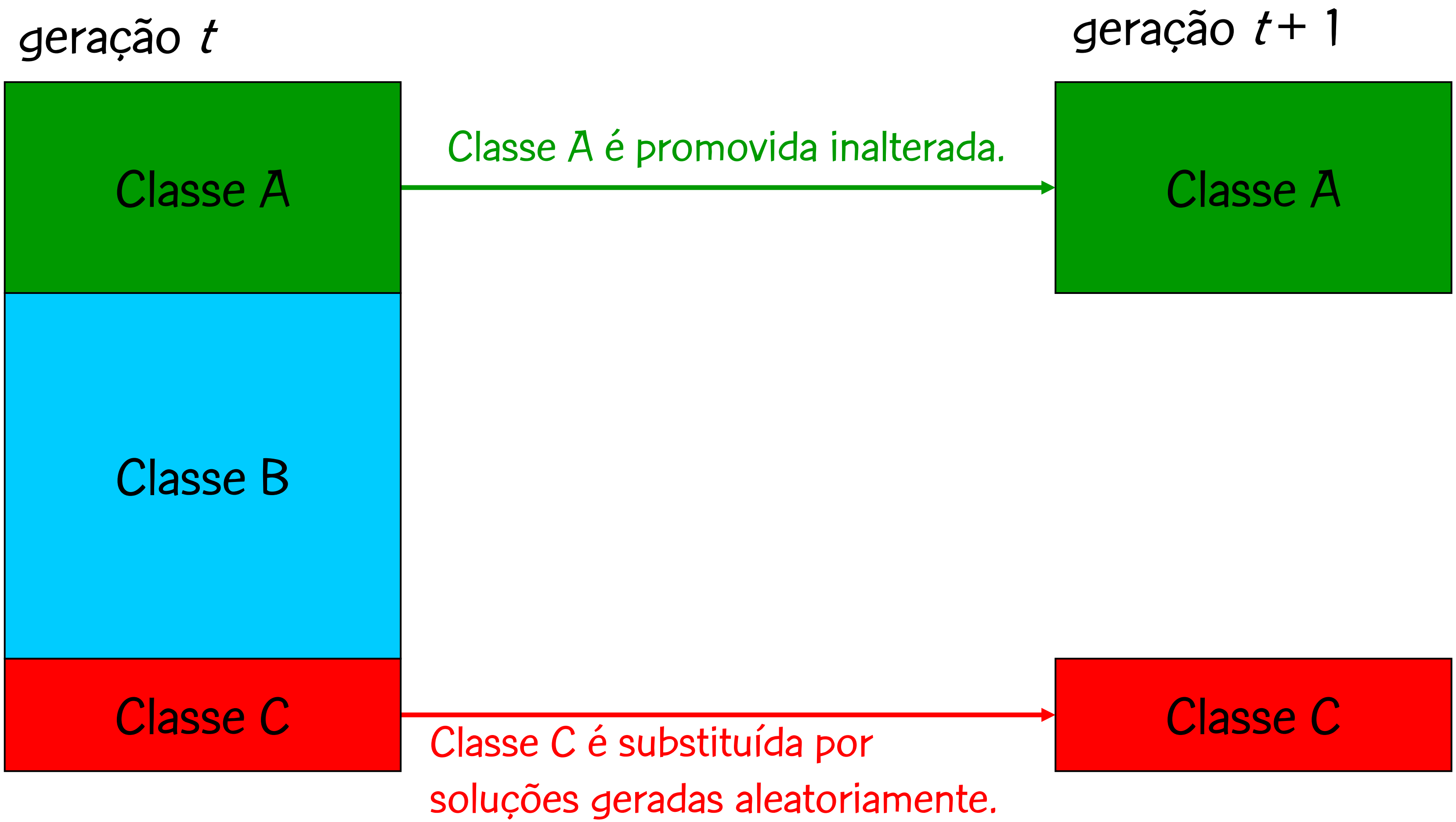
geração *t*



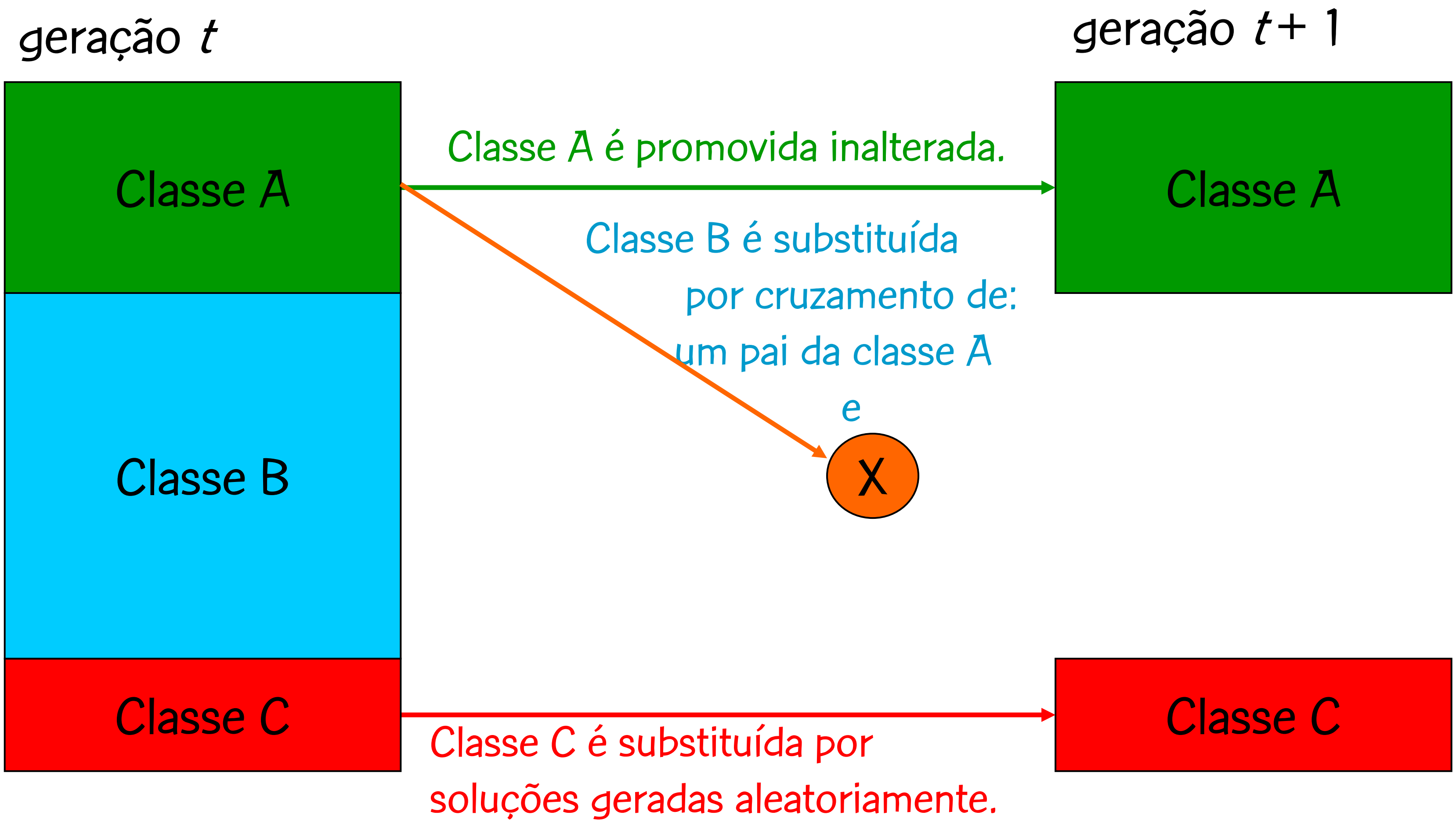
AGs com chaves aleatórias



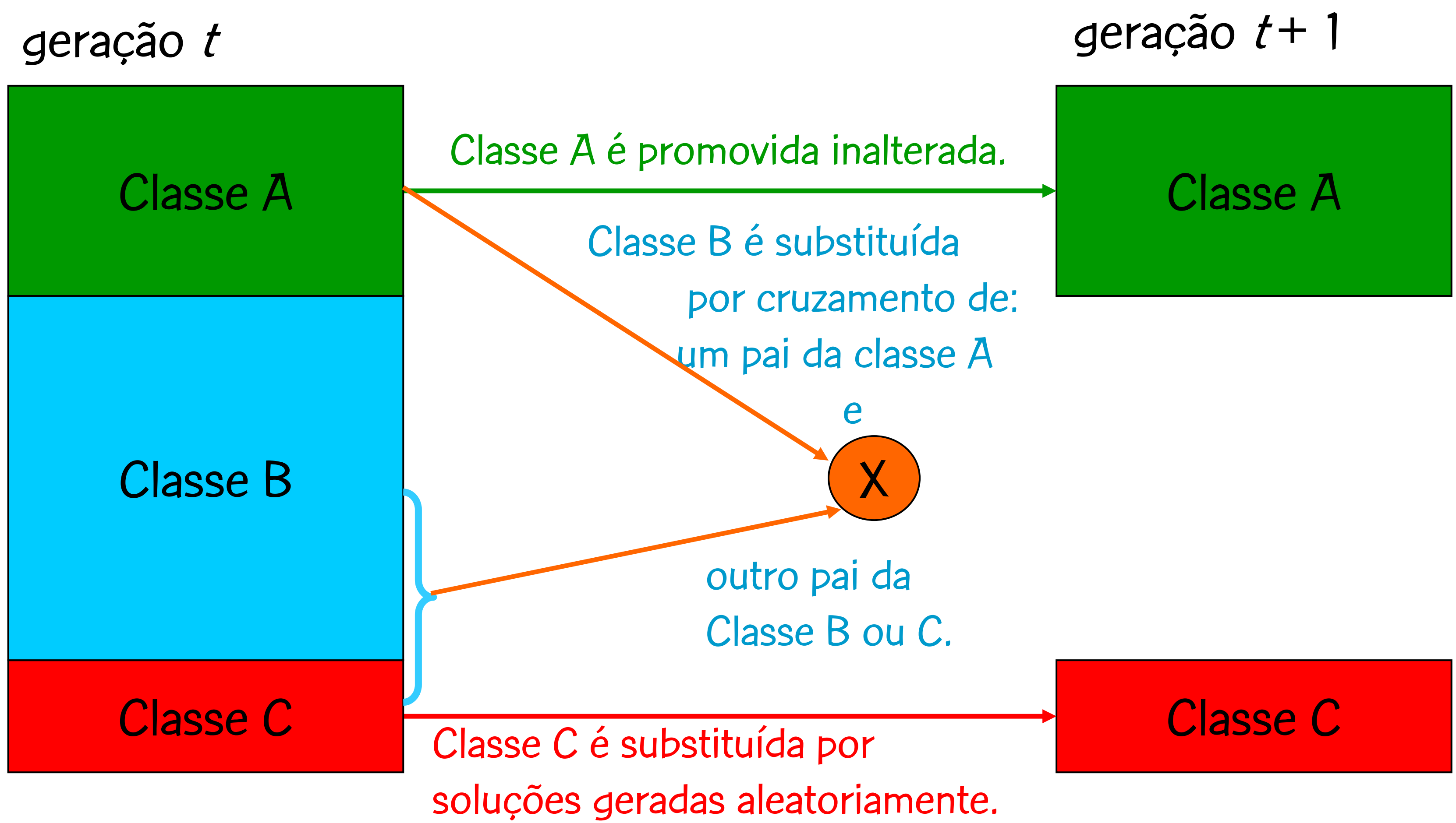
AGs com chaves aleatórias



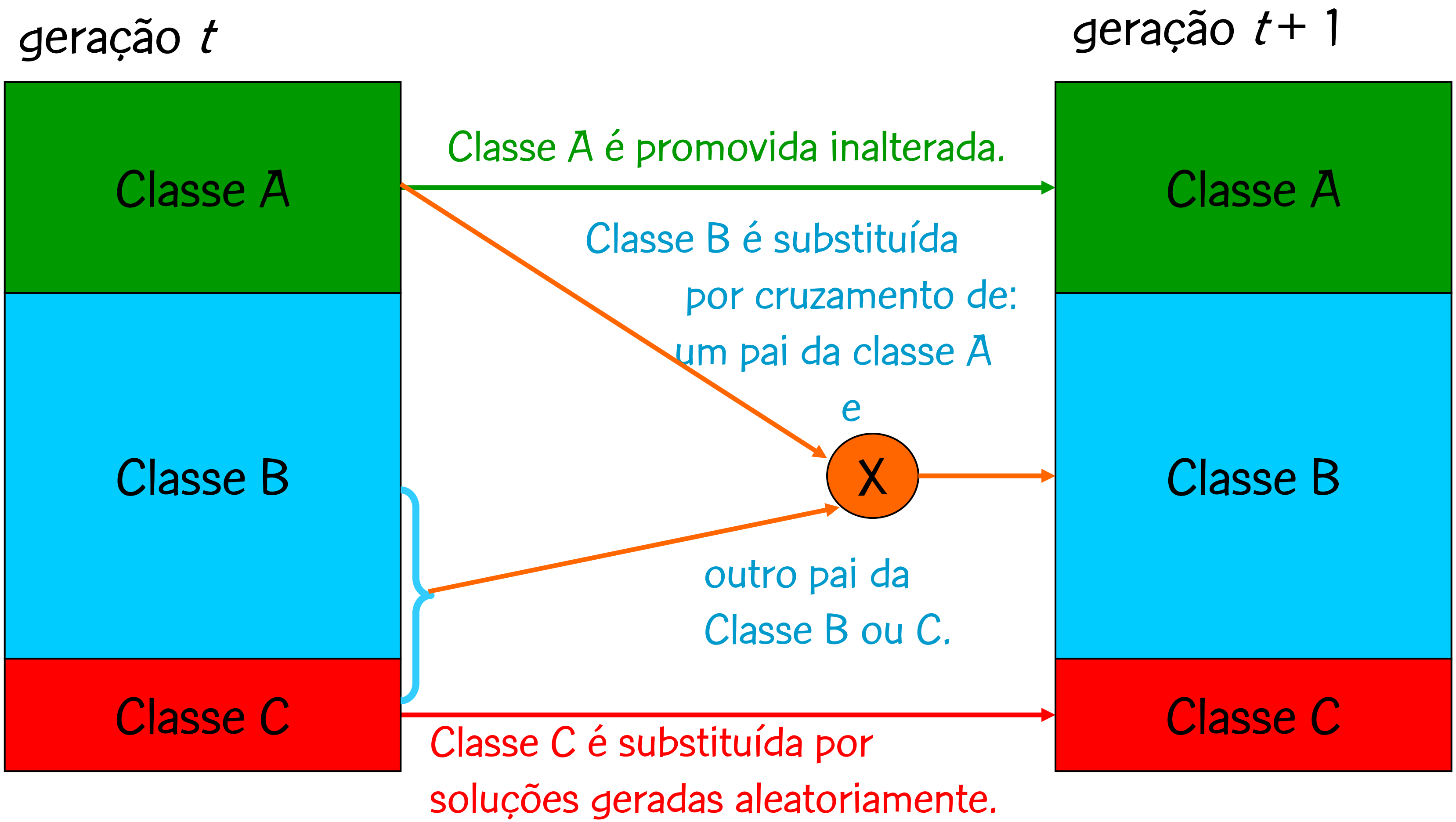
AGs com chaves aleatórias



AGs com chaves aleatórias



AGs com chaves aleatórias



AGs com chaves aleatórias

- Pais escolhidos aleatoriamente:
  - um pai da Classe A (elite).
  - um pai da Classe B ou C (não-elite).
- Seleção duplicada é permitida, i.e. pais podem ser combinados mais do que uma vez por geração.
- Indivíduos melhores terão maiores chances de reproduzirem-se.

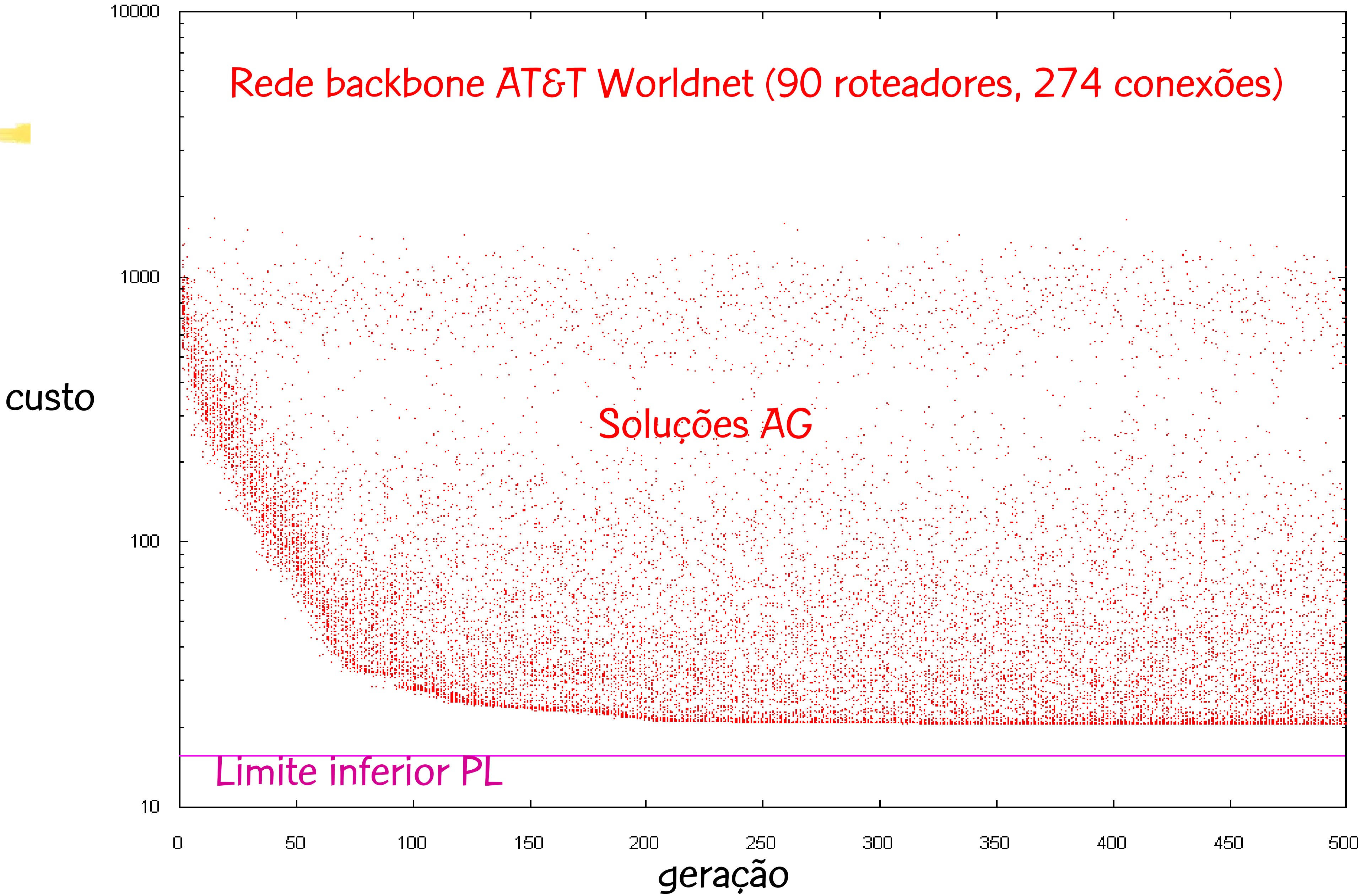
AGs com chaves aleatórias

Cruzamento combina pai elite p_1 com pai não-elite parent p_2 para obter filho c:

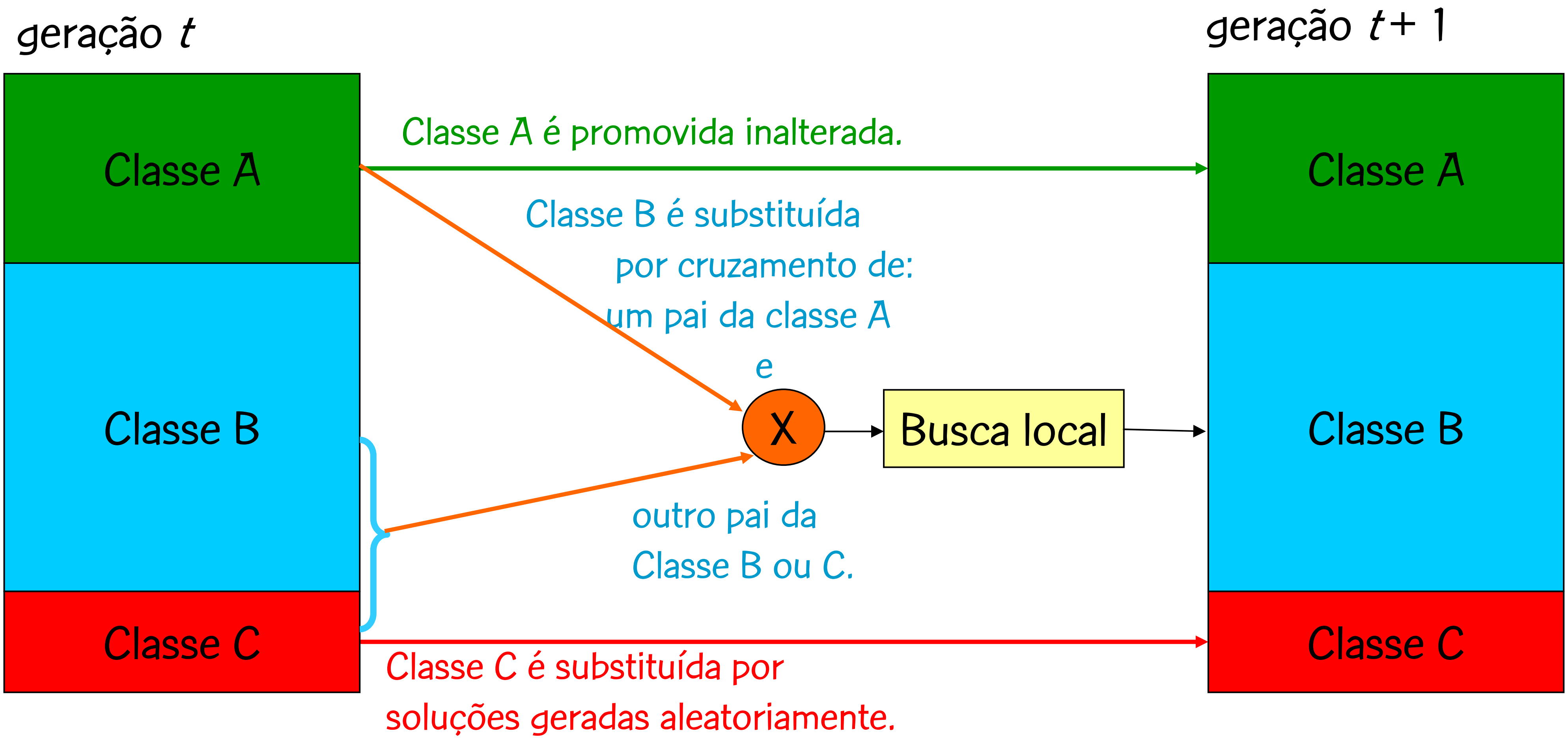
Filho sofre apenas mutação de um gene com probabilidade pequena.

Filho herda gene do pai elite com maior probabilidade.

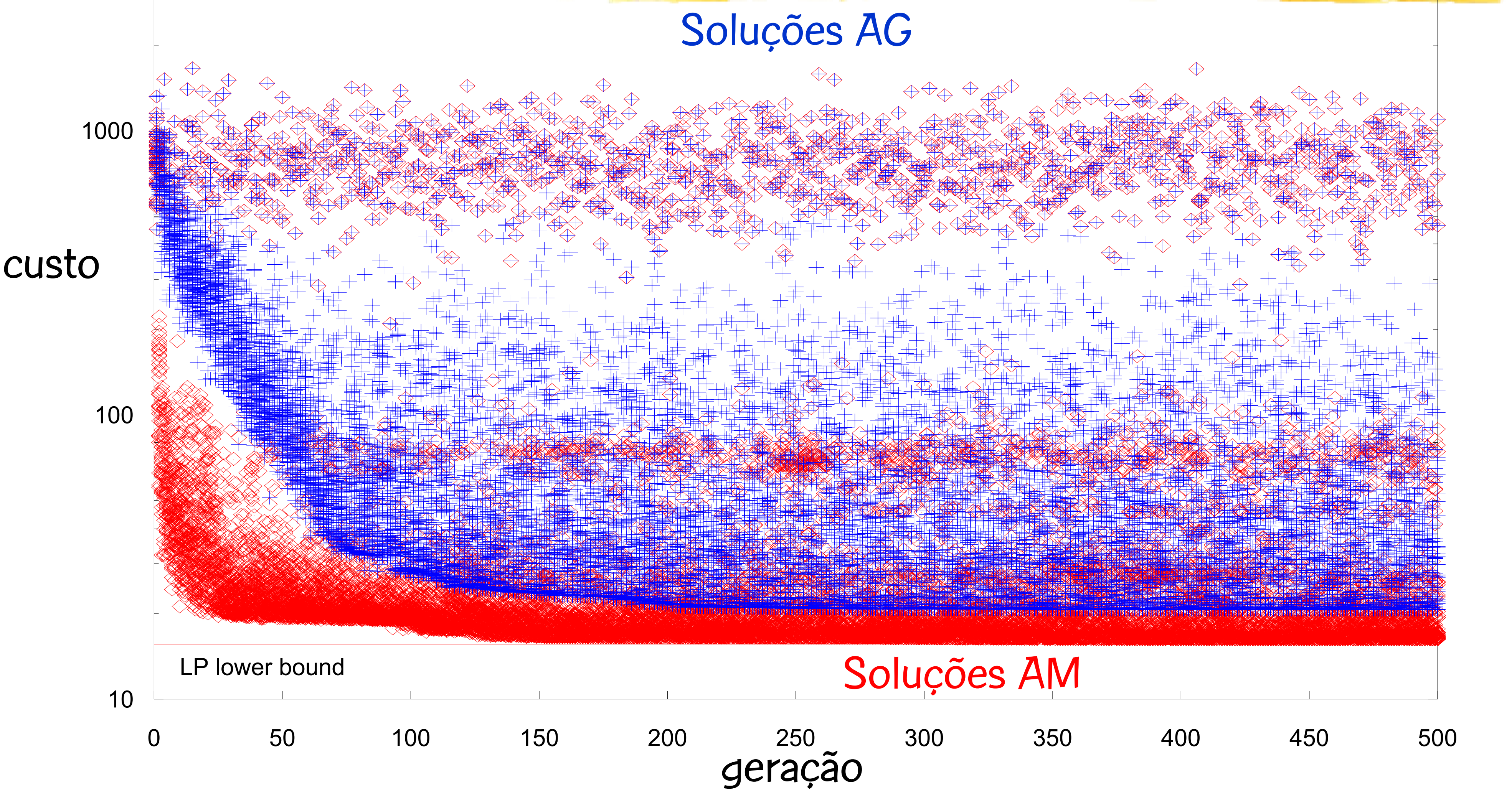
for all genes $i = 1,2,\ldots,|A|$ do
  if rrandom[0,1] < 0.01 then
    $c[i]$ = irandom[1, w_{max}]
  else if rrandom[0,1] < 0.7 then
    $c[i] = p_1[i]$
  else $c[i] = p_2[i]$
  end


Rede backbone AT&T Worldnet (90 roteadores, 274 conexões)
custo
Soluções AG
Limite inferior PL
10000
1000
100
10
0
50
100
150
200
250
300
350
400
450
500
geração

AGs com chaves aleatórias

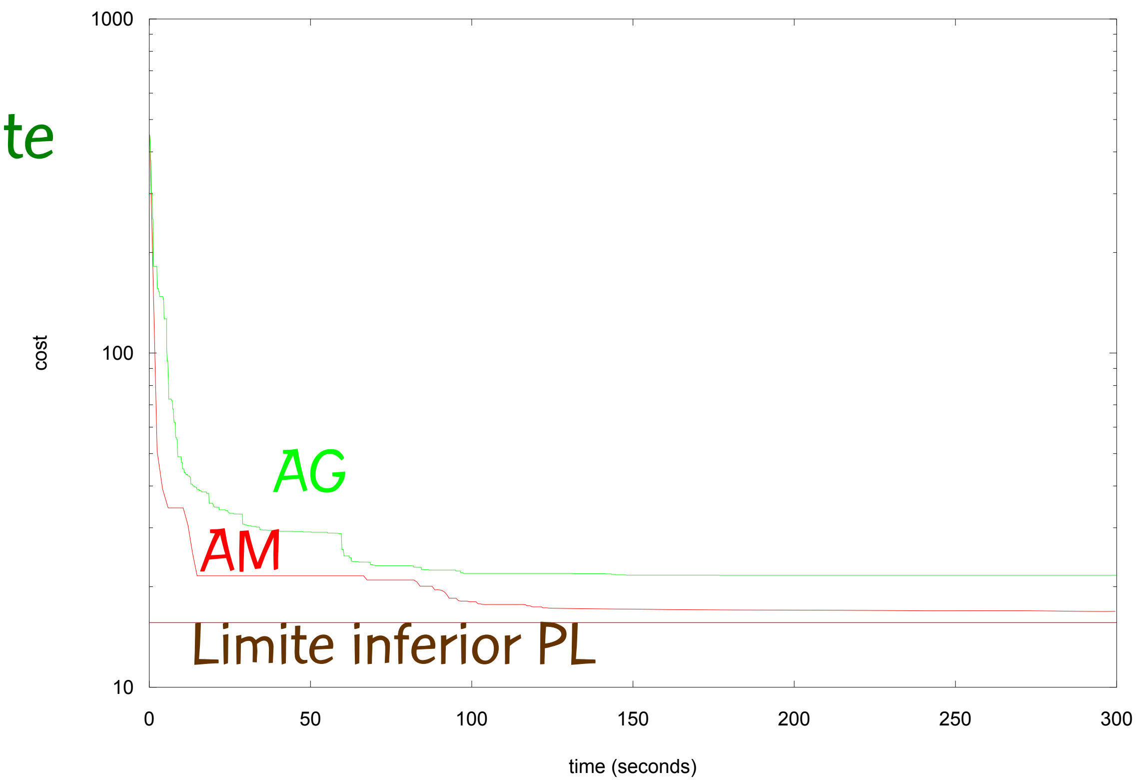


Rede backbone AT&T Worldnet (90 roteadores, 274 conexões)



AGs com chaves aleatórias

- Algoritmo memético = Algoritmo genético + busca local
- Algoritmo memético melhora algoritmo genético de duas maneiras, pois encontra:
  - soluções mais rapidamente
  - soluções melhores



Algoritmos genéticos

- Processamento paralelo:
  - Baixo nível: paralelizar as operações de cada fase
    - Avaliar os indivíduos em paralelo.
    - Realizar seleção e cruzamentos em paralelo.
    - Aplicação: otimização da exploração de reservatórios de petróleo para a Petrobrás (determinar os parâmetros ótimos de exploração), a avaliação de cada solução consiste em executar um modelo de simulação de reservatórios (tempos elevados, solução de EDPs).
  - Alto nível: dividir a população original em diversas subpopulações, cujas evoluções ocorrem em paralelo em diferentes processadores com eventuais trocas dos melhores indivíduos entre as subpopulações (modelo de ilhas).