Problemas de otimização

- Problemas de decisão: "Existe uma solução satisfazendo certa propriedade?"
  Resultado: "sim" ou "não"

- Problemas de otimização: "Entre todas as soluções satisfazendo determinada propriedade, obter aquela que otimiza certa função de custo."
  Resultado: uma solução viável ótima

Problemas de otimização

- Exemplo: problema do caixeiro viajante
  Entrada: n cidades e distâncias $c_{i,j}$

  Problema de decisão: "dado um inteiro L, existe uma rota que visite cada cidade exatamente uma vez e cujo comprimento seja menor ou igual a L?"

  Problema de otimização: "obter uma rota que visite cada cidade exatamente uma vez e cujo comprimento seja mínimo."

Problemas de otimização

- <u>Exemplo</u>: problema da mochila
  Entrada: n itens, peso máximo b, lucros c_j e pesos a_j associados a cada item $j=1,\ldots,n$

  Problema de decisão: "dado um inteiro L, existe $S \subseteq \{1,\ldots,n\}$ tal que $\sum_{j\in S} a_j \leq b$ e $\sum_{j\in S} c_j \geq L$?"

  Problema de otimização: "obter um subconjunto S* maximizando $\sum_{j\in S} c_j$ entre todos os conjuntos $S \subseteq \{1,\ldots,n\}$ tais que $\sum_{j\in S} a_j \leq b$."

Problemas de otimização

- Estes problemas de decisão pertencem à classe dos problemas NP-completos, para os quais não são conhecidos algoritmos determinísticos de complexidade polinomial.
- A versão de otimização destes problemas (e de muitos outros) também são intrinsecamente "intratáveis" do ponto de vista computacional, não se conhecendo algoritmos eficientes (polinomiais) para sua solução exata.

Problemas de otimização

- Como tratar e resolver estes problemas?
  - Algoritmos exatos não-polinomiais
  - Algoritmos pseudo-polinomiais em alguns casos
  - Processamento paralelo: aceleração na prática, mas sem redução da complexidade!
  - Casos especiais polinomiais
  - Algoritmos aproximativos: encontram uma solução com custo a distância máxima garantida do valor ótimo
  - Algoritmos probabilísticos: convergência em valor esperado ou em probabilidade

Problemas de otimização

- Como tratar e resolver estes problemas?
  - Heurísticas: métodos aproximados projetados 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
    - Algoritmos construtivos
    - Busca local (algoritmos de melhoria)
    - Metaheurísticas

Problemas de otimização

- Avanços no estudo e no desenvolvimento de heurísticas:
  - 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

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 $F \subseteq 2^E$ é o conjunto de soluções viáveis do problema (minimização).
- Construção de uma solução: selecionar seqüencialmente elementos de E, eventualmente descartando alguns já selecionados, terminando quando for encontrada uma solução viável

Algoritmos construtivos

- Exemplo: problema do caixeiro viajante
  - E: conjunto de arestas
  - F: subconjuntos de E que formam um circuito hamiltoniano, visitando cada cidade exatamente uma vez
  - $c(S) = \Sigma_{e \in S}\, c_e$
  - $c_e = c_{i,j}$: custo da aresta $e=(i,j)$

Algoritmos construtivos

- Algoritmo do vizinho mais próximo:

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

Enquanto $N \neq \emptyset$ fazer:

Obter $j \in N$: $c_{i,j} = \min_{k \in N} \{c_{i,k}\}$

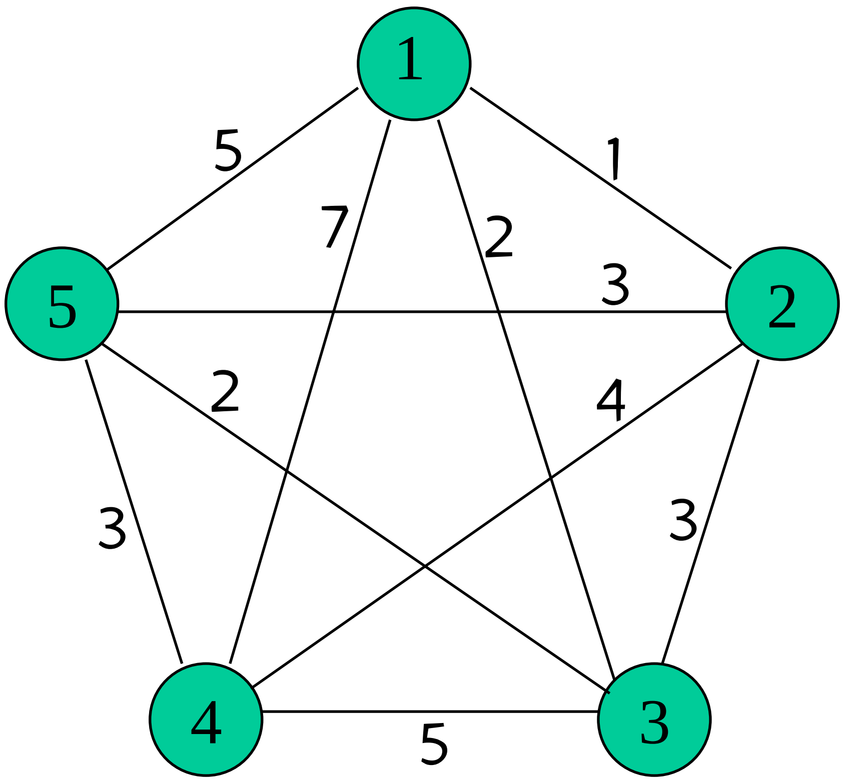
$N \leftarrow N-\{j\}$

$i \leftarrow j$

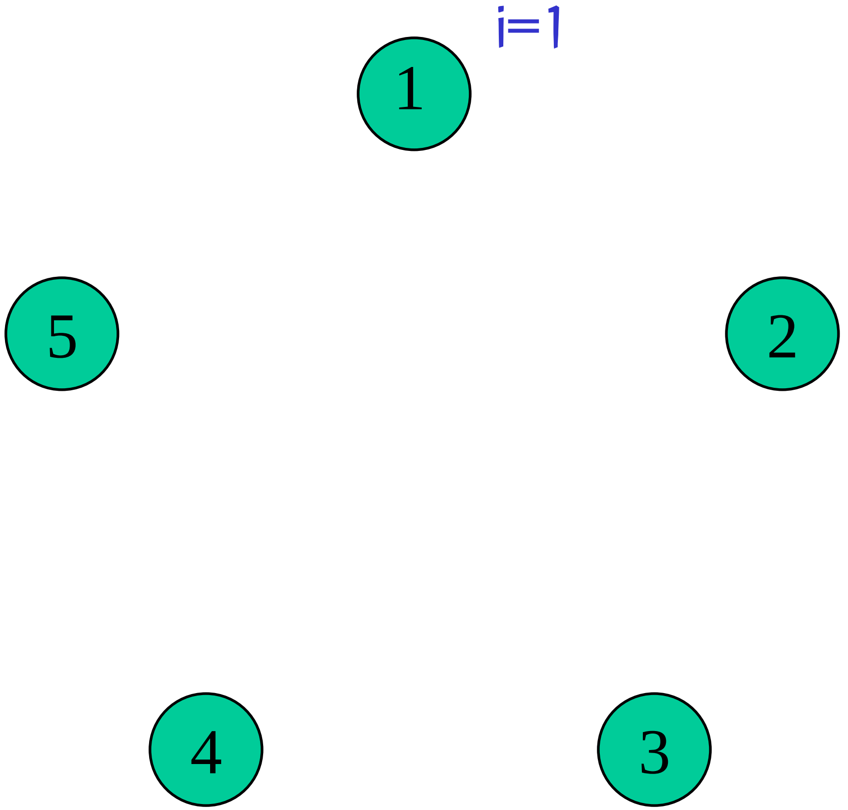
Fim-enquanto

Calcular o custo da solução e terminar.

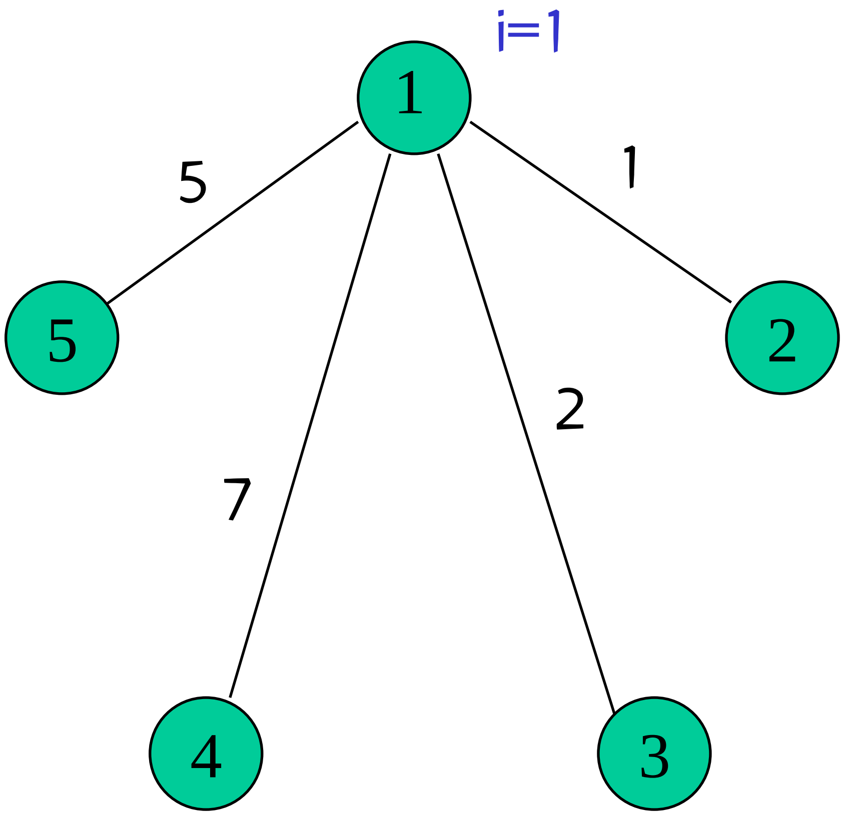
Algoritmos construtivos



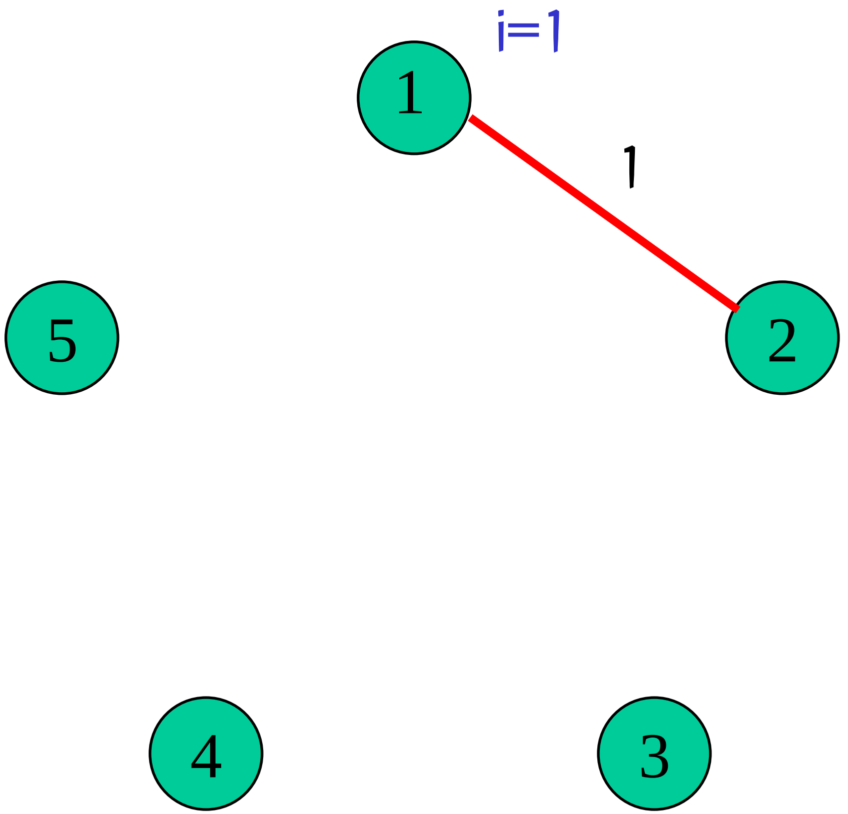
Algoritmos construtivos



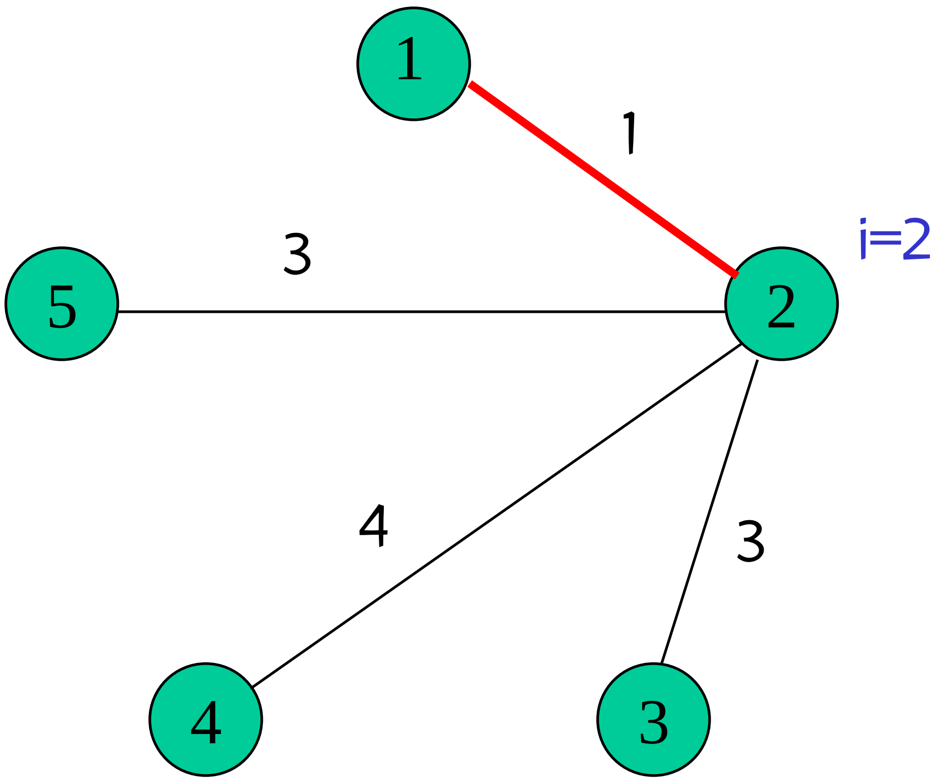
Algoritmos construtivos



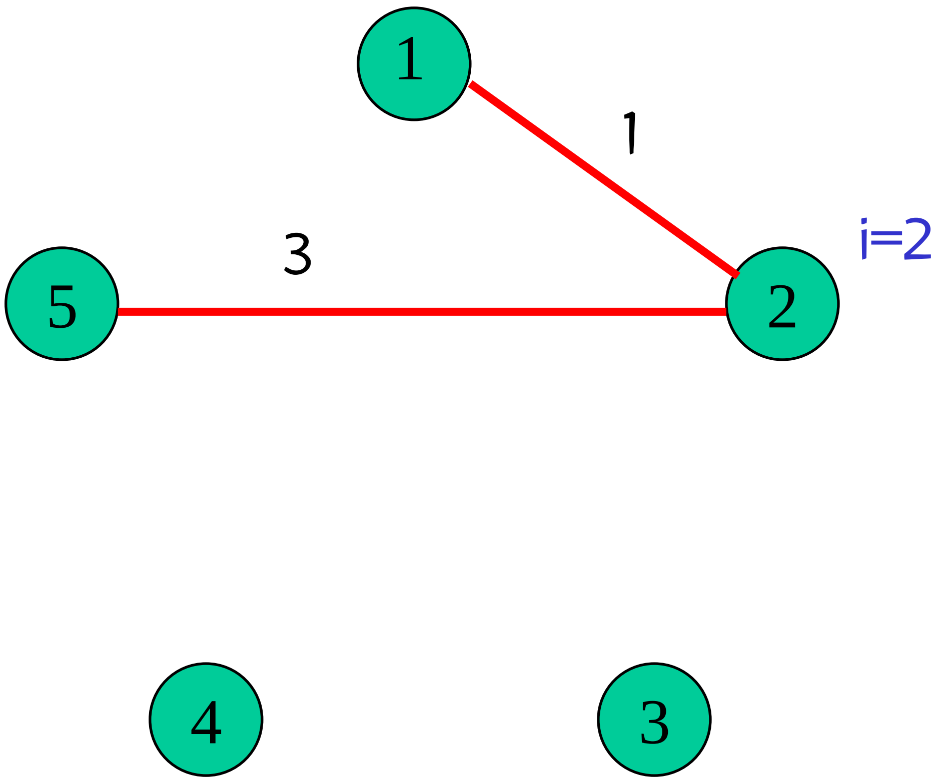
Algoritmos construtivos



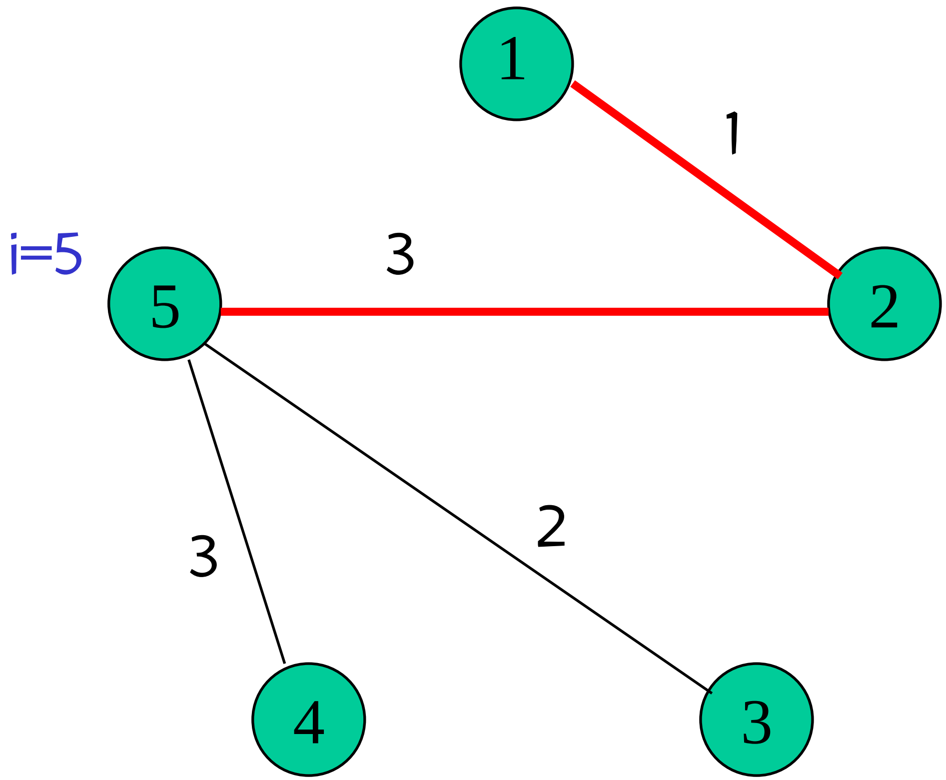
Algoritmos construtivos



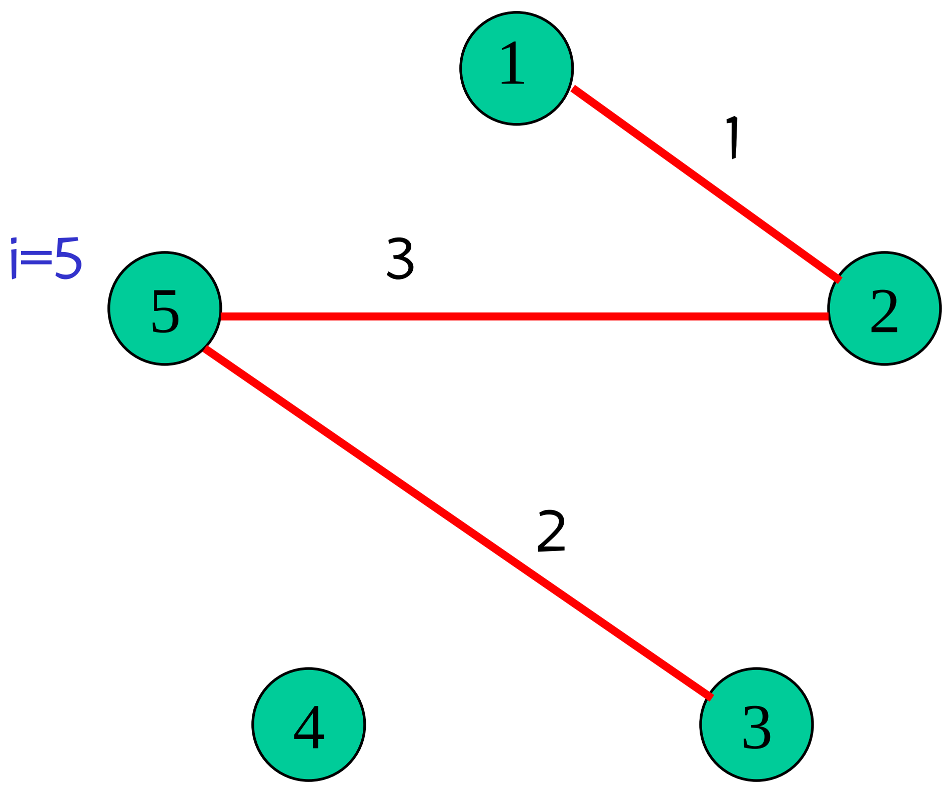
Algoritmos construtivos



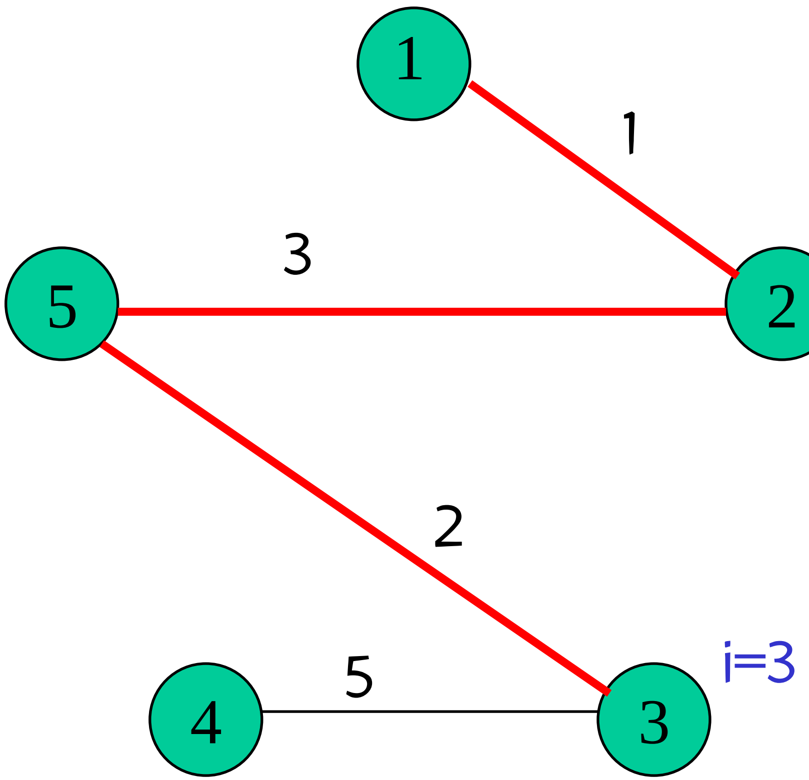
Algoritmos construtivos



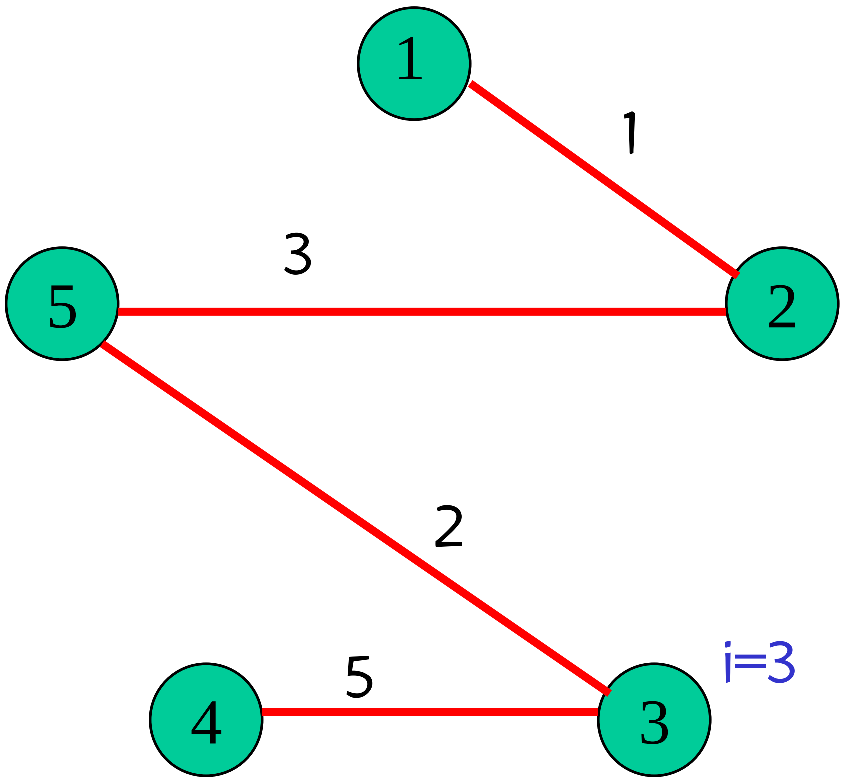
Algoritmos construtivos



Algoritmos construtivos

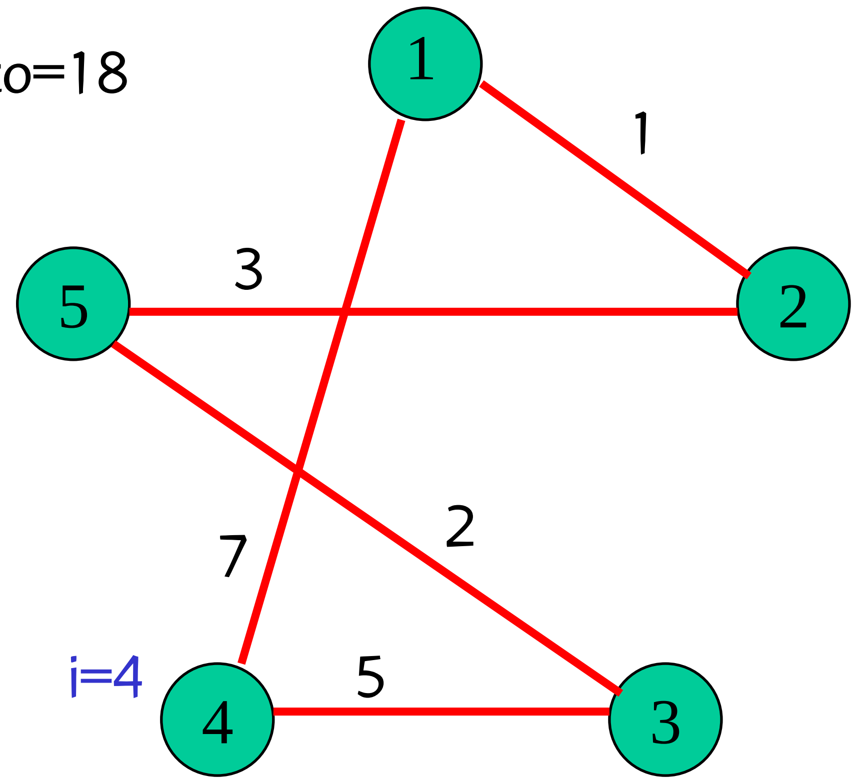


Algoritmos construtivos



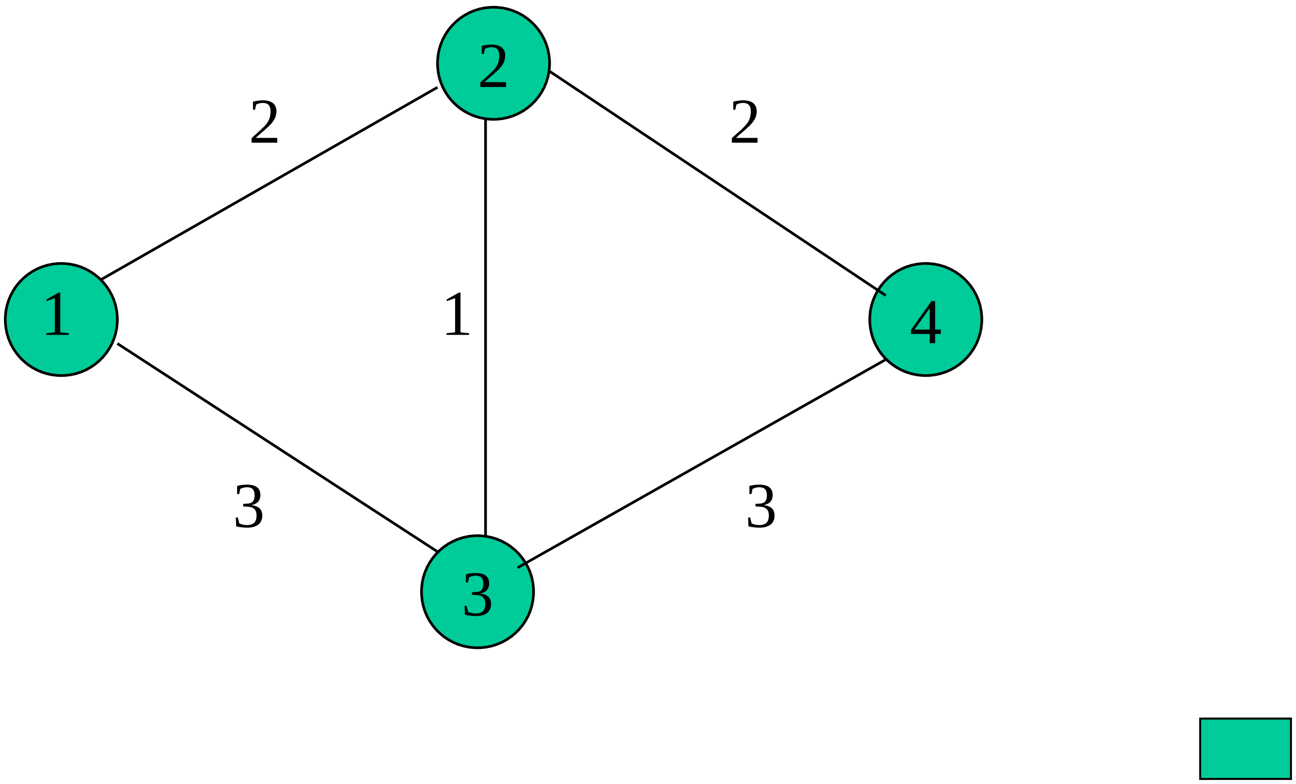
Algoritmos construtivos

comprimento=18



Algoritmos construtivos

- Podem falhar mesmo para casos muito simples!



Algoritmos gulosos

- Algoritmos gulosos:

  A construção de uma solução gulosa consiste em selecionar a cada passo o elemento de E ainda não utilizado que minimiza o incremento no custo da solução parcial sem torná-la inviável, terminando quando se obtém uma solução viável.

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

Algoritmos gulosos

- Algoritmo guloso de Kruskal para o problema da árvore geradora de peso mínimo
  - Sempre encontra a solução ótima: este problema pertence a uma classe particular de problemas onde um algoritmo guloso sempre encontra a solução ótima.
- Algoritmo do vizinho mais próximo para o PCV
  - Cuidado: nem sempre encontra a solução ótima exata, é portanto uma heurística para este problema!

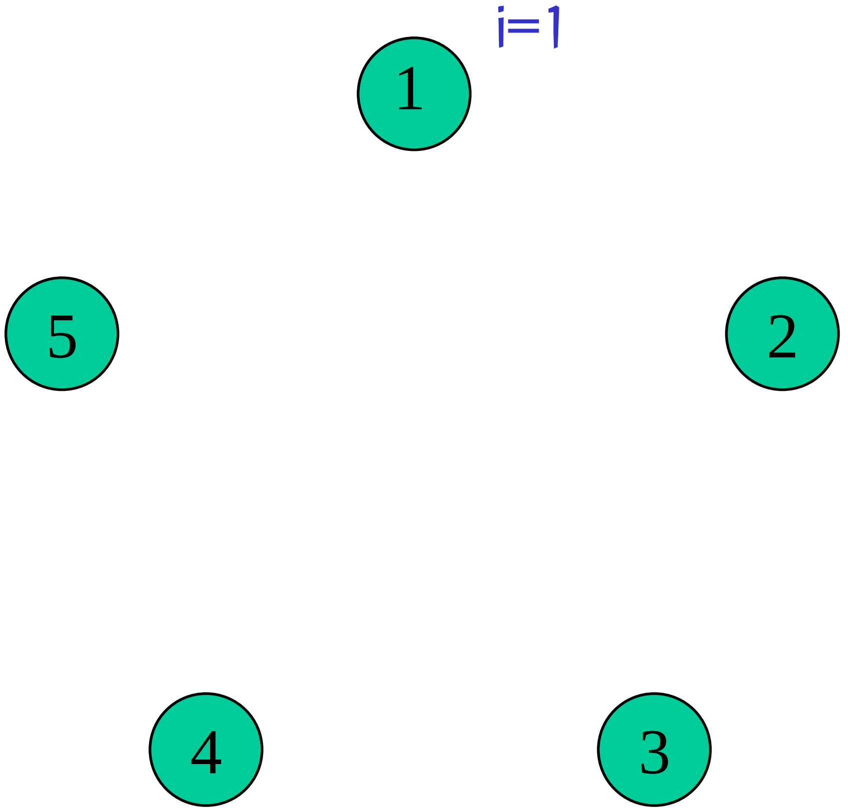
Algoritmos gulosos randomizados

- Um algoritmo guloso encontra sempre a mesma solução para um dado problema.

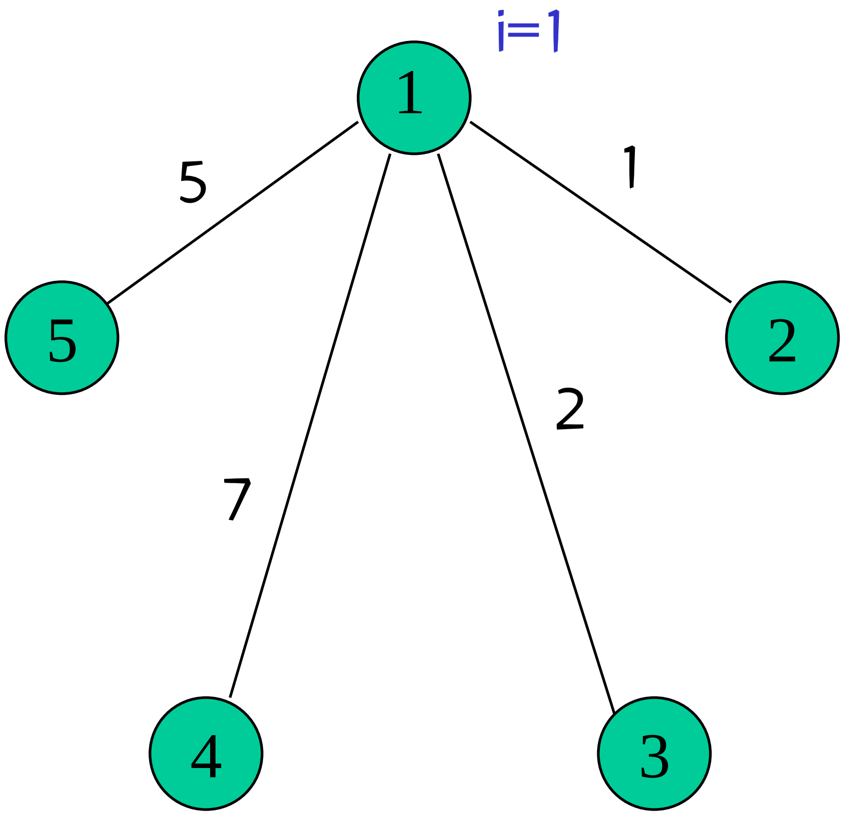
- Algoritmo guloso randomizado:
  - Criar uma lista de candidatos a cada iteração com os melhores elementos ainda não selecionados e forçar uma escolha aleatória a cada iteração

- Aplicar o algoritmo repetidas vezes, obtendo soluções diferentes a cada iteração.

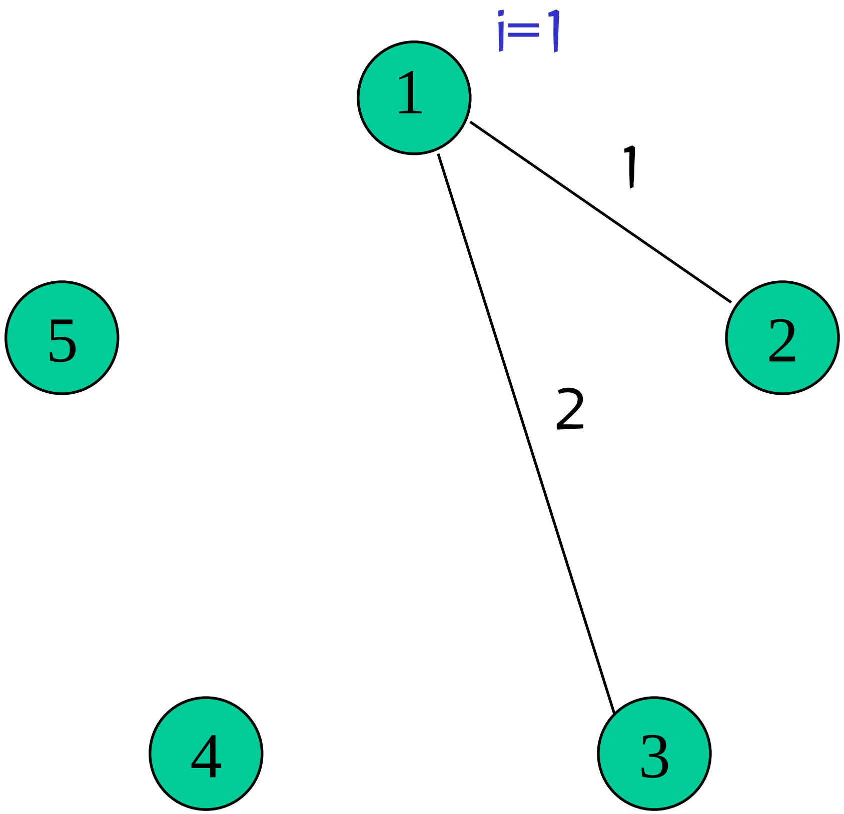
Algoritmos gulosos randomizados



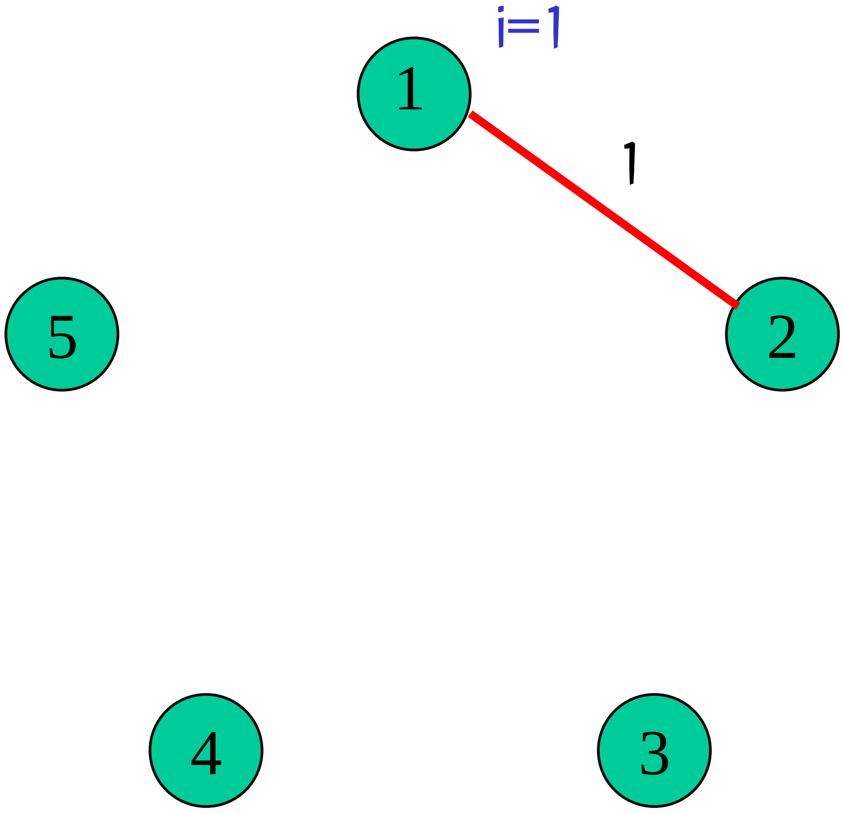
Algoritmos gulosos randomizados



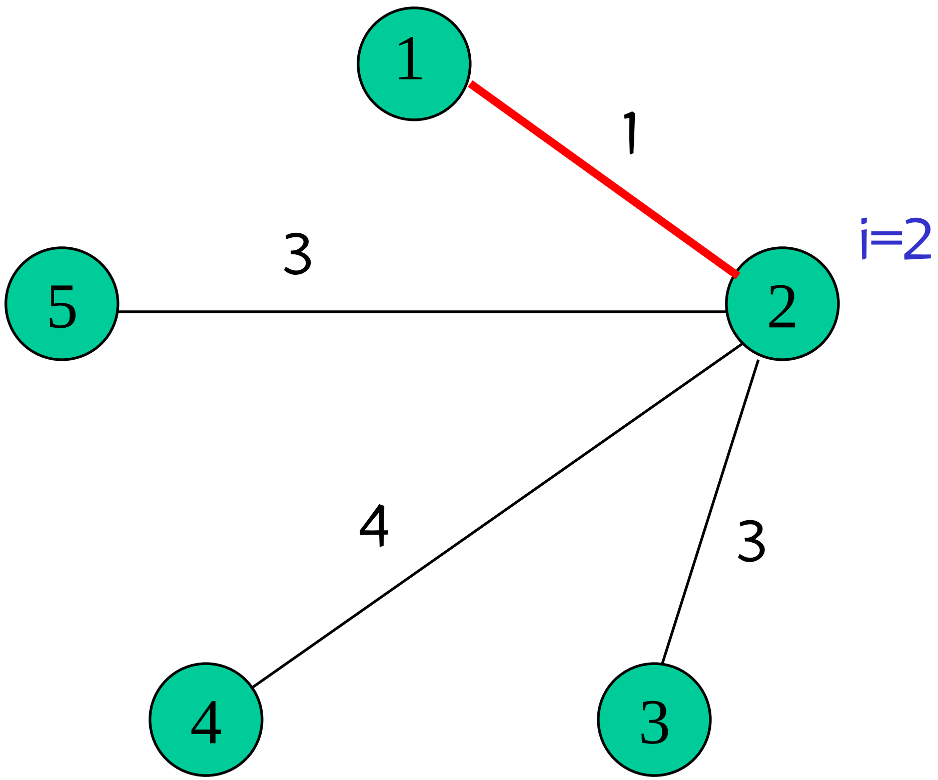
Algoritmos gulosos randomizados



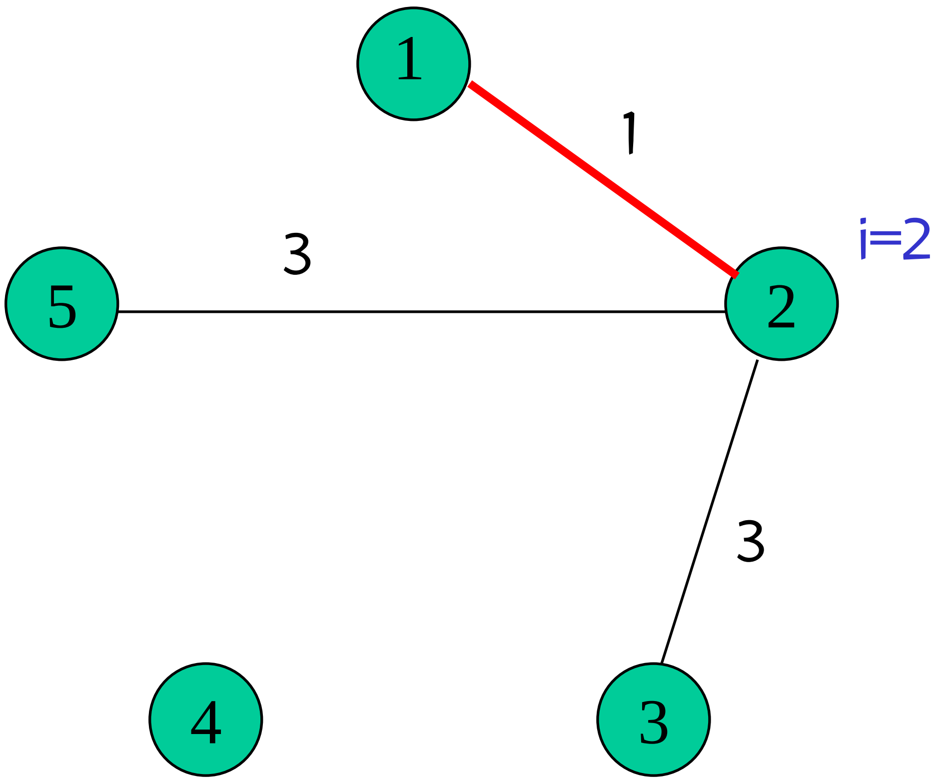
Algoritmos gulosos randomizados



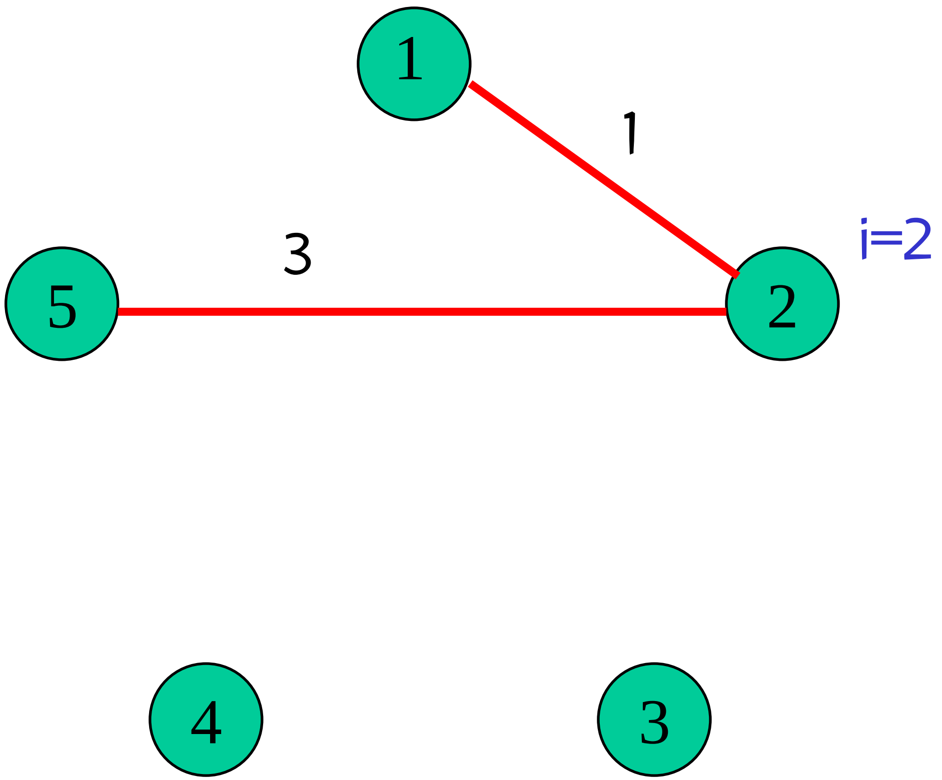
Algoritmos gulosos randomizados



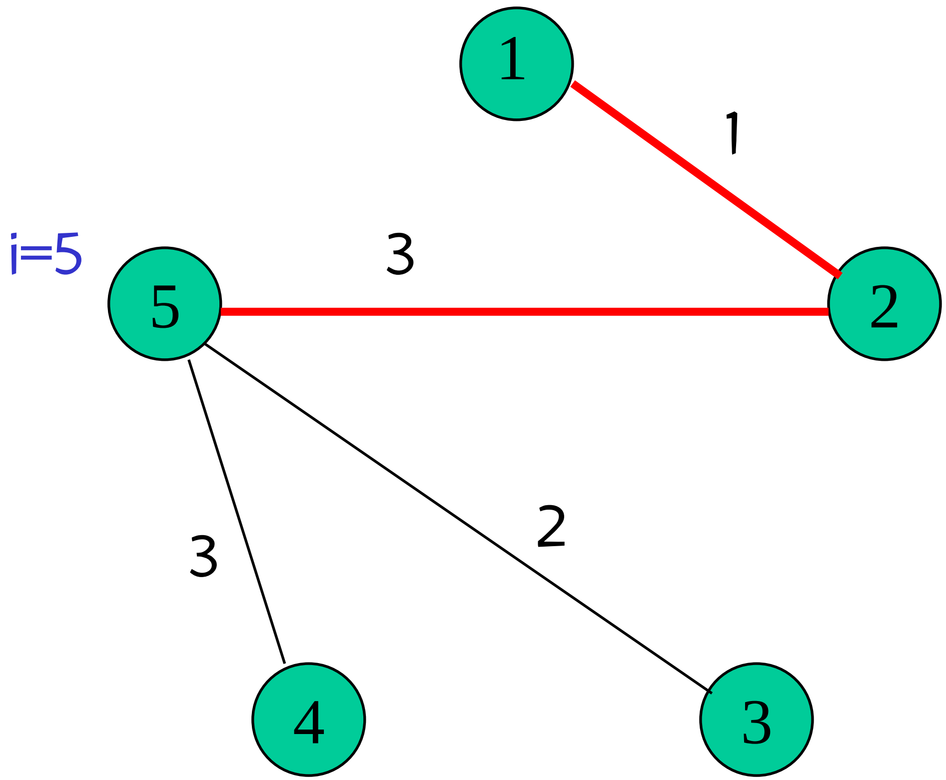
Algoritmos gulosos randomizados



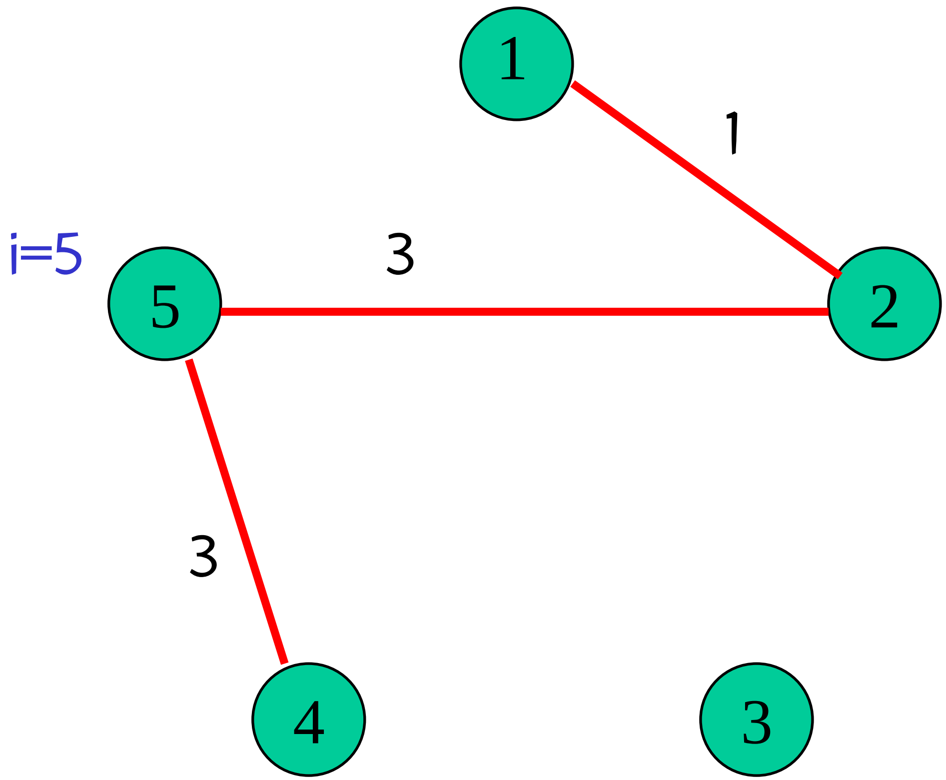
Algoritmos gulosos randomizados



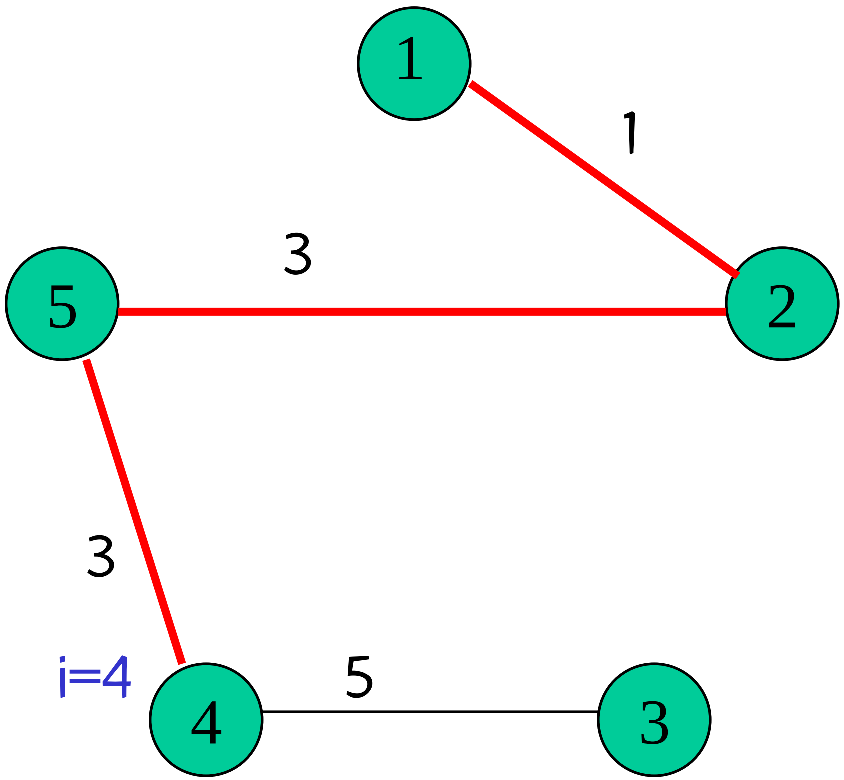
Algoritmos gulosos randomizados



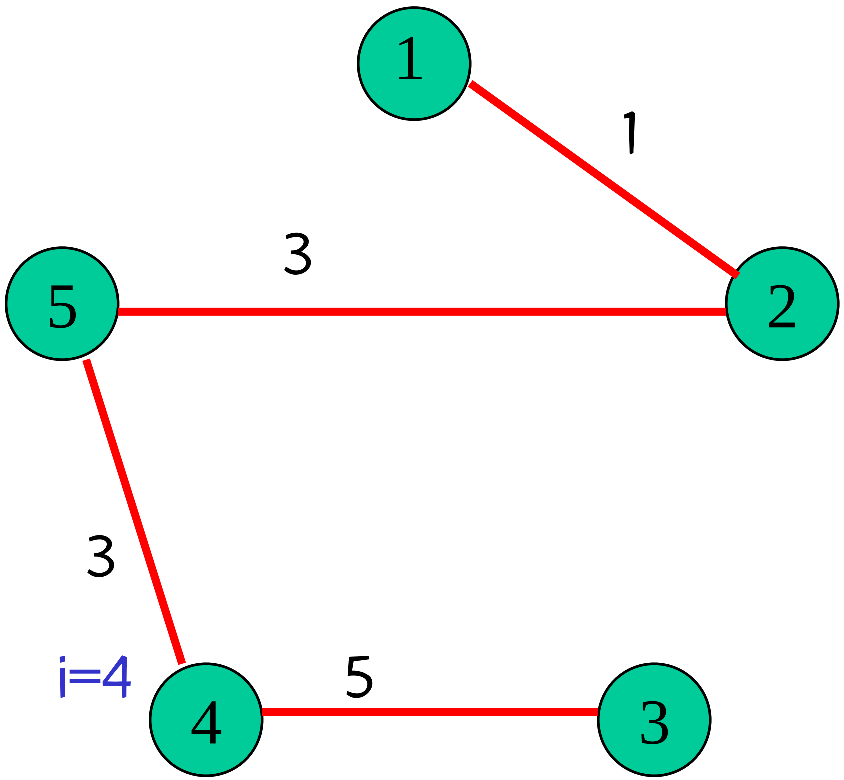
Algoritmos gulosos randomizados



Algoritmos gulosos randomizados

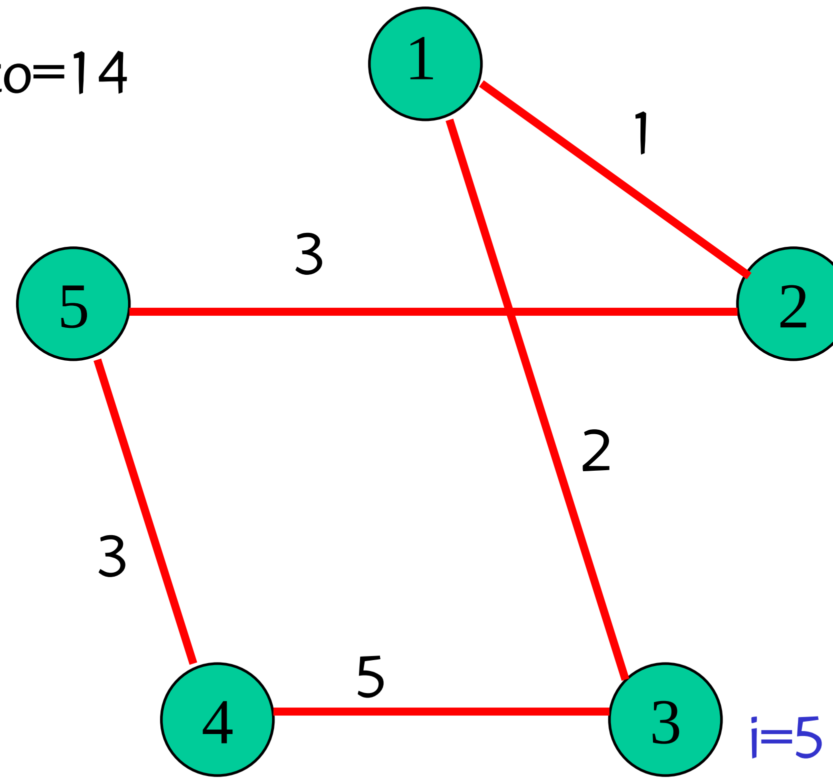


Algoritmos gulosos randomizados

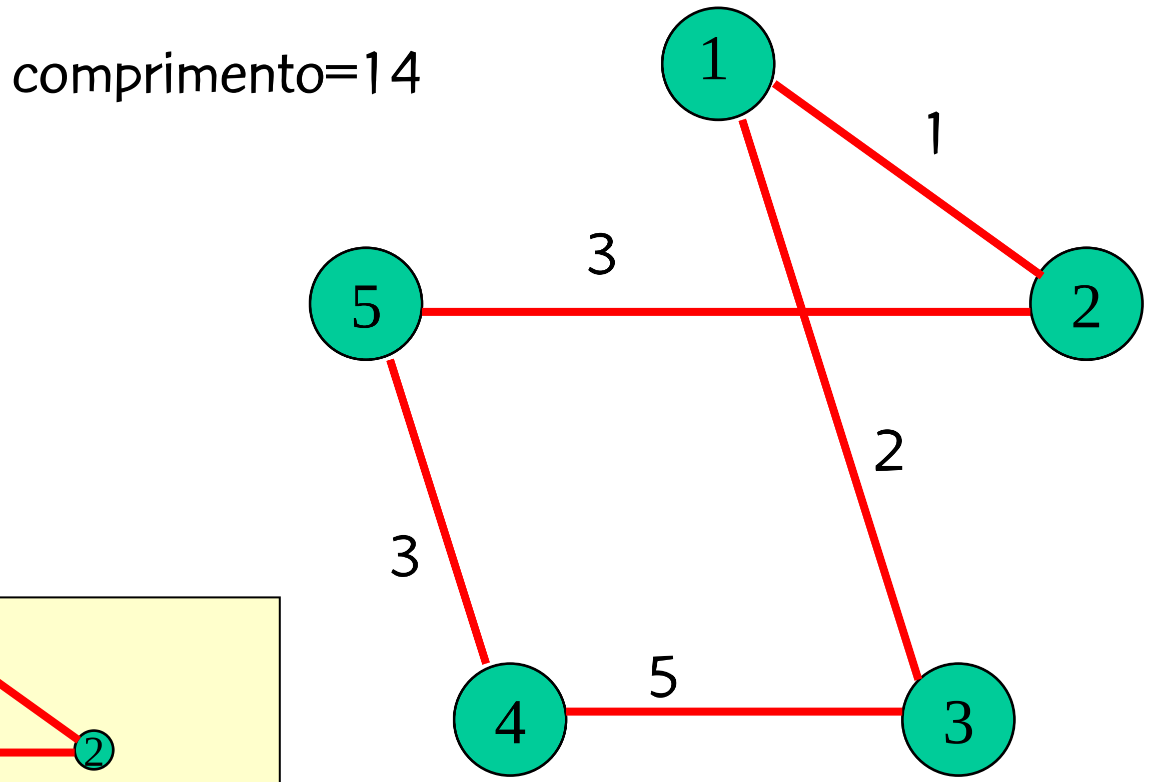


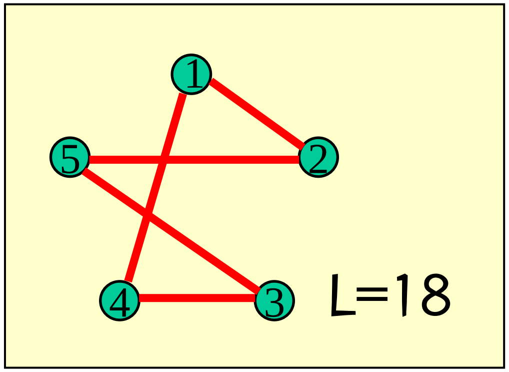
Algoritmos gulosos randomizados

comprimento=14



Algoritmos gulosos randomizados





Algoritmos gulosos randomizados

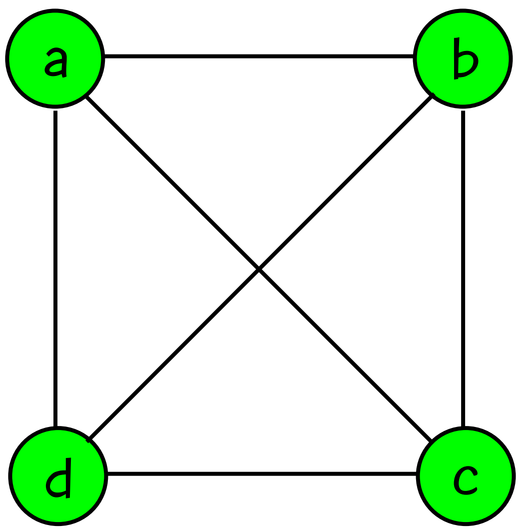
- A qualidade da solução obtida depende da qualidade dos elementos na lista de candidatos.
- A diversidade das soluções encontradas depende da cardinalidade da lista de candidatos.
- Casos extremos:
  - algoritmo guloso puro (o candidato único é o melhor)
  - solução gerada de forma completamente aleatória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

Vizinhanças

- Conjunto F de soluções viáveis formado por subconjuntos do conjunto suporte E que satisfazem determinadas condições.
- Representação de uma solução: indicar quais elementos de E estão presentes e quais não estão.
- Problema da mochila:
  itens j=1,...,n, peso máximo b, lucros c_j e pesos a_j
  Solução representada por um vetor 0-1 com n posições:
  $x_j = 1$ se o item j é selecionado, $x_j = 0$ caso contrário

Vizinhanças

- Problema do caixeiro viajante: representar cada solução pela ordem em que os vértices são visitados, como uma permutação circular dos n vértices (o primeiro vértice é arbitrário)
  - (a)bcd
  - (a)bdc
  - (a)cbd
  - (a)dcb
  - (a)cdb
  - (a)dbc



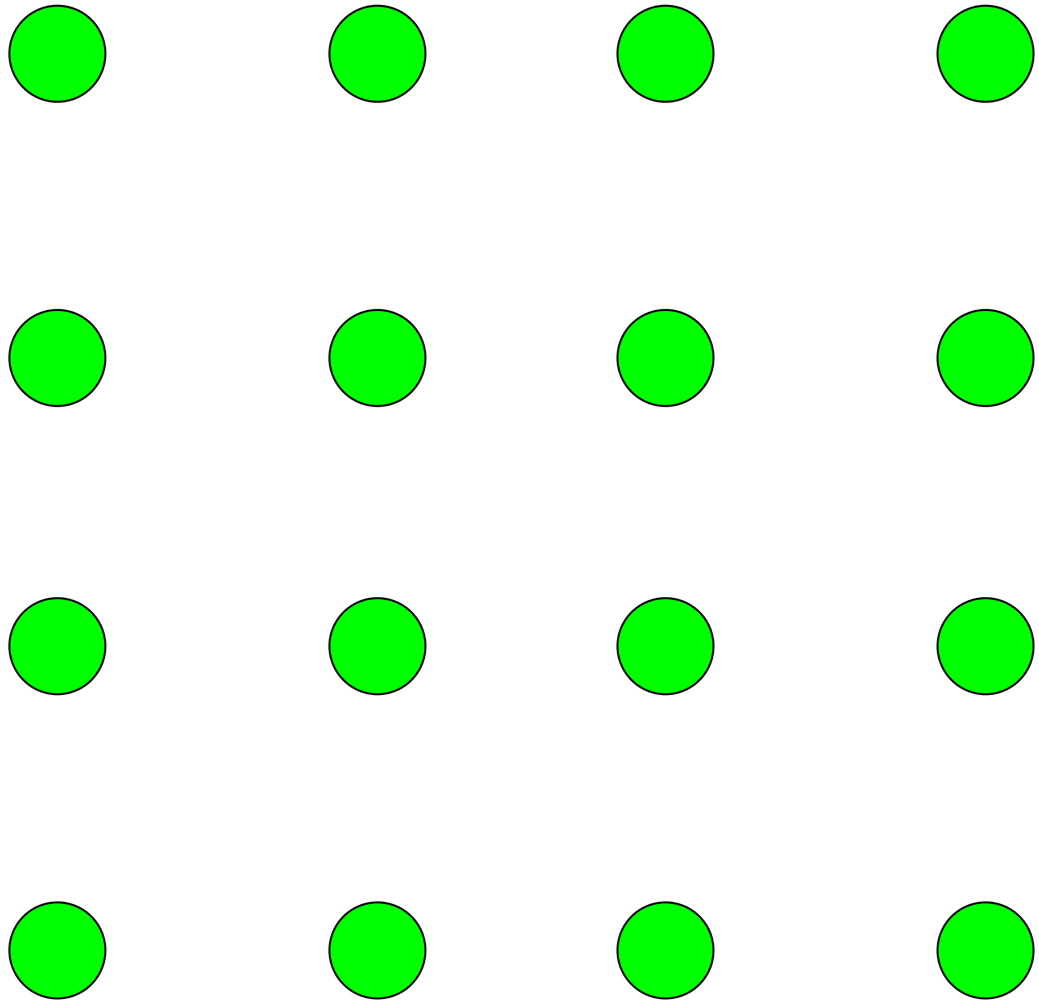
Vizinhanças

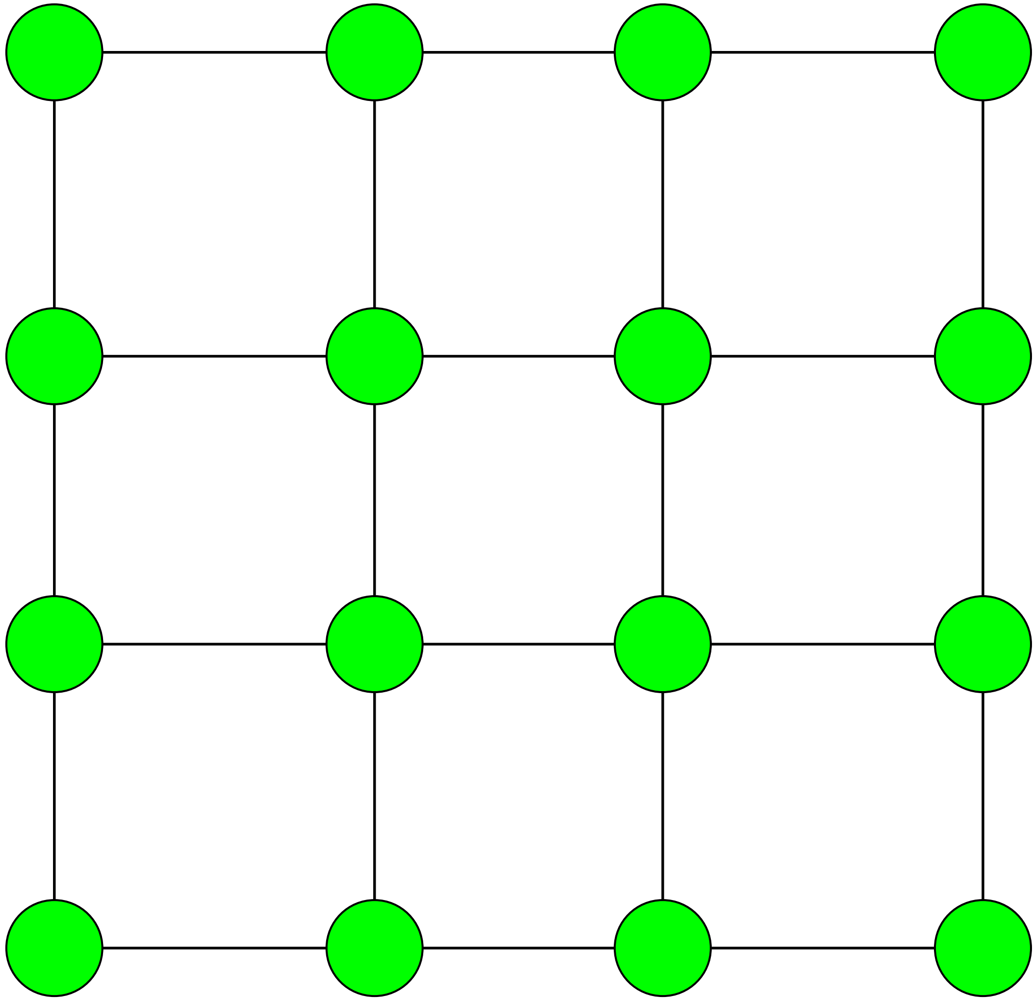
- Problema: $c(S^*) = \text{mínimo } \{ c(S): S \in F \subseteq 2^E \}$

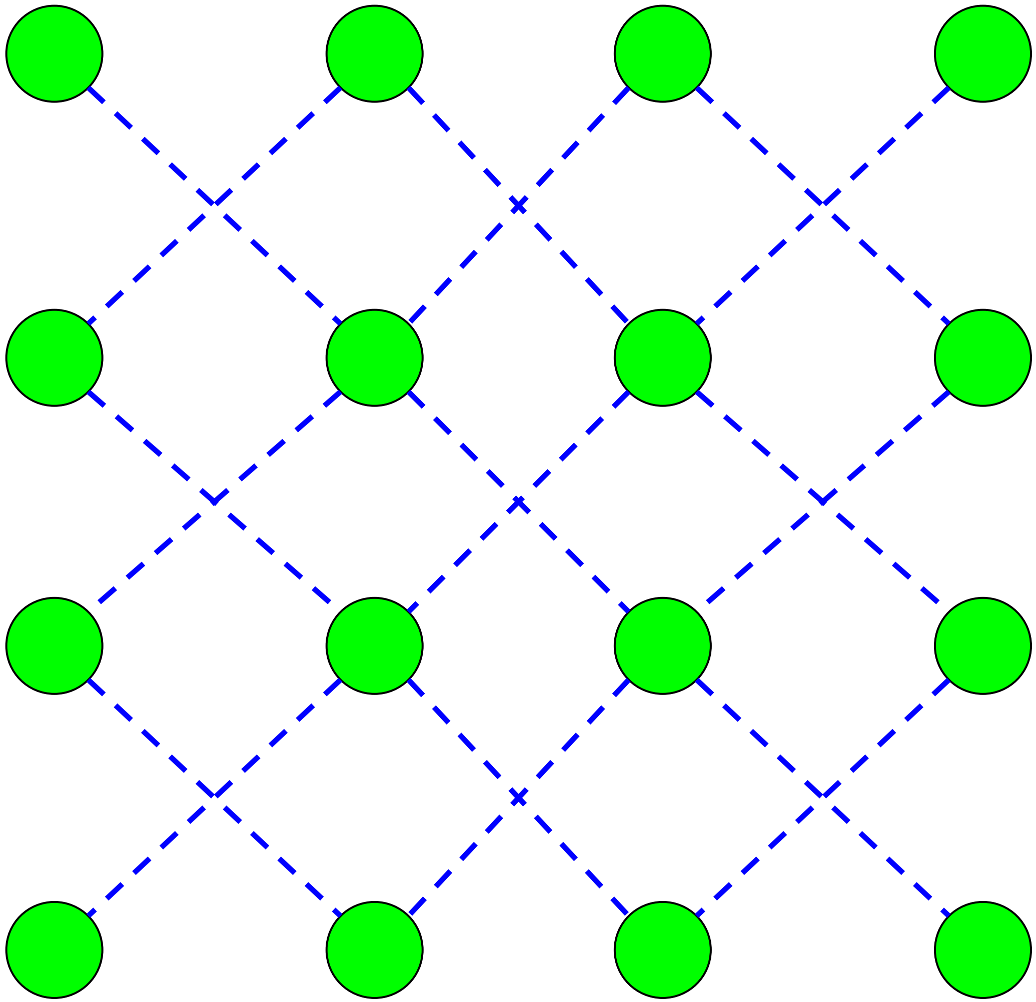
- Vizinhança: um conceito que introduz a noção de proximidade entre as soluções em F

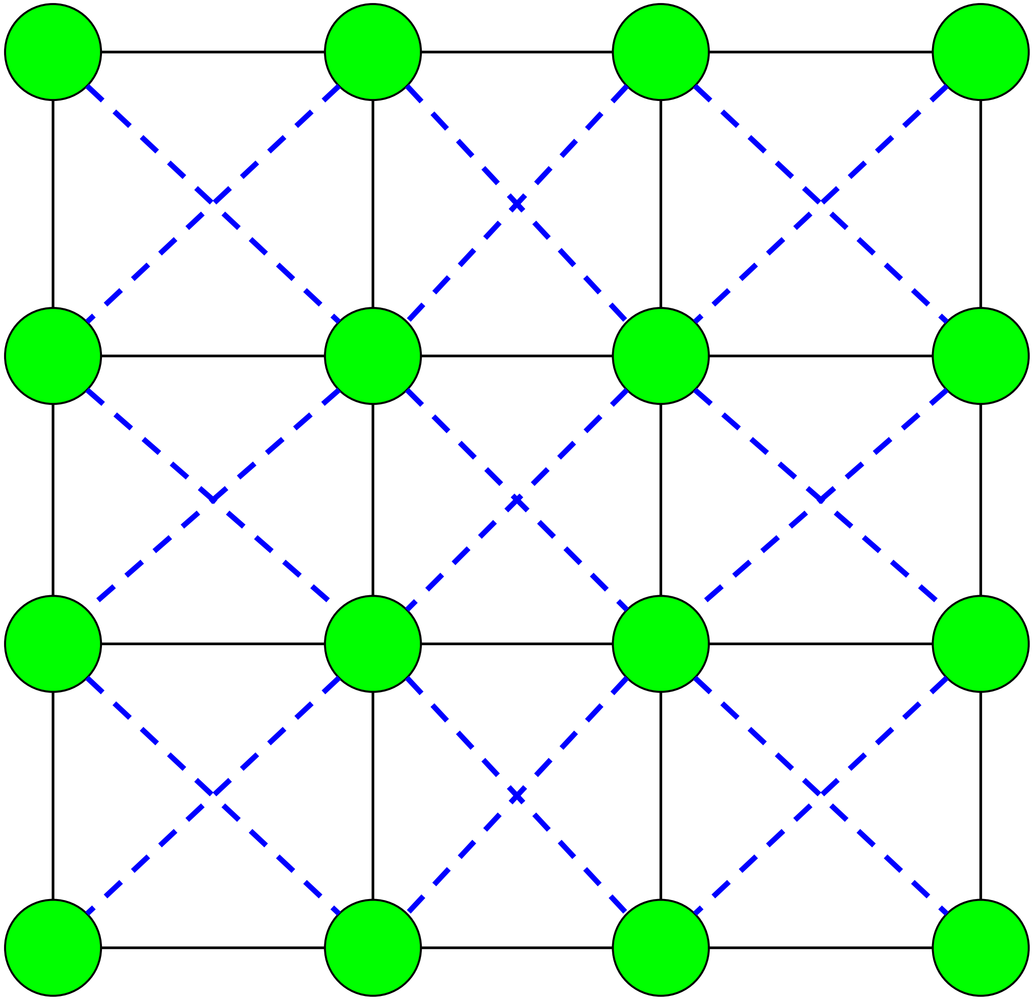
- Uma vizinhança é um mapeamento que associa cada solução a um conjunto de soluções (vizinhos)

  $N(S) = \{S_1, S_2, \ldots, S_k\}$ soluções vizinhas de S









Vizinhanças

- Boas representações permitem representar de forma compacta o conjunto de soluções vizinhas de uma solução S qualquer e percorrer de modo eficiente o conjunto de soluções.

Vizinhanças

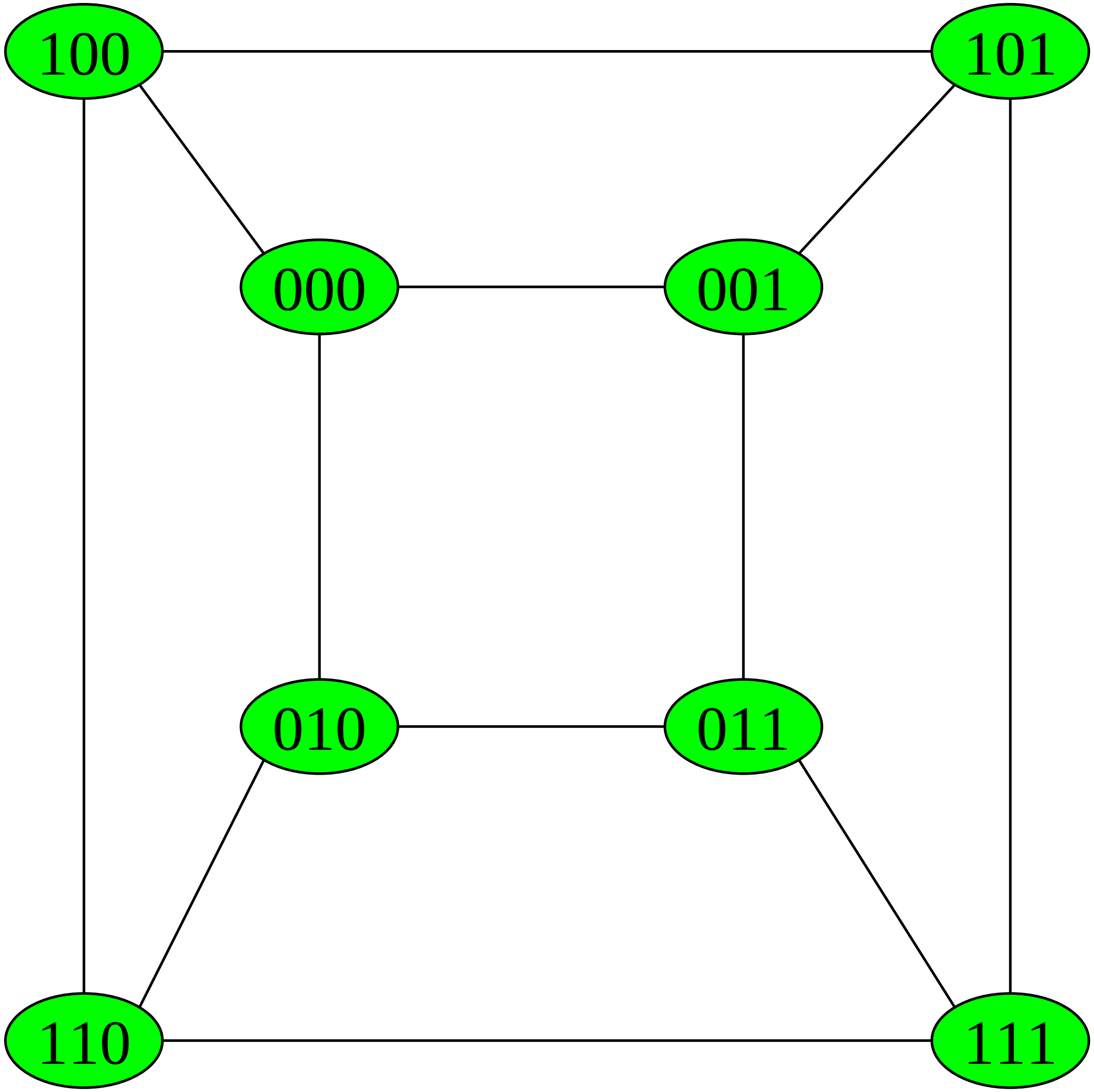
- Exemplo: problema da mochila

Solução $S = (x_1,\ldots,x_i,\ldots,x_n) \quad x_i \in \{0,1\},\ i=1,\ldots,n$

$N(S) = \{(x_1,\ldots,1\text{-}x_i,\ldots,x_n): i=1,..,n\}$

Vizinhos de (1,0,1,1) =
{(0,0,1,1), (1,1,1,1), (1,0,0,1), (1,0,1,0)}

Vizinhanças

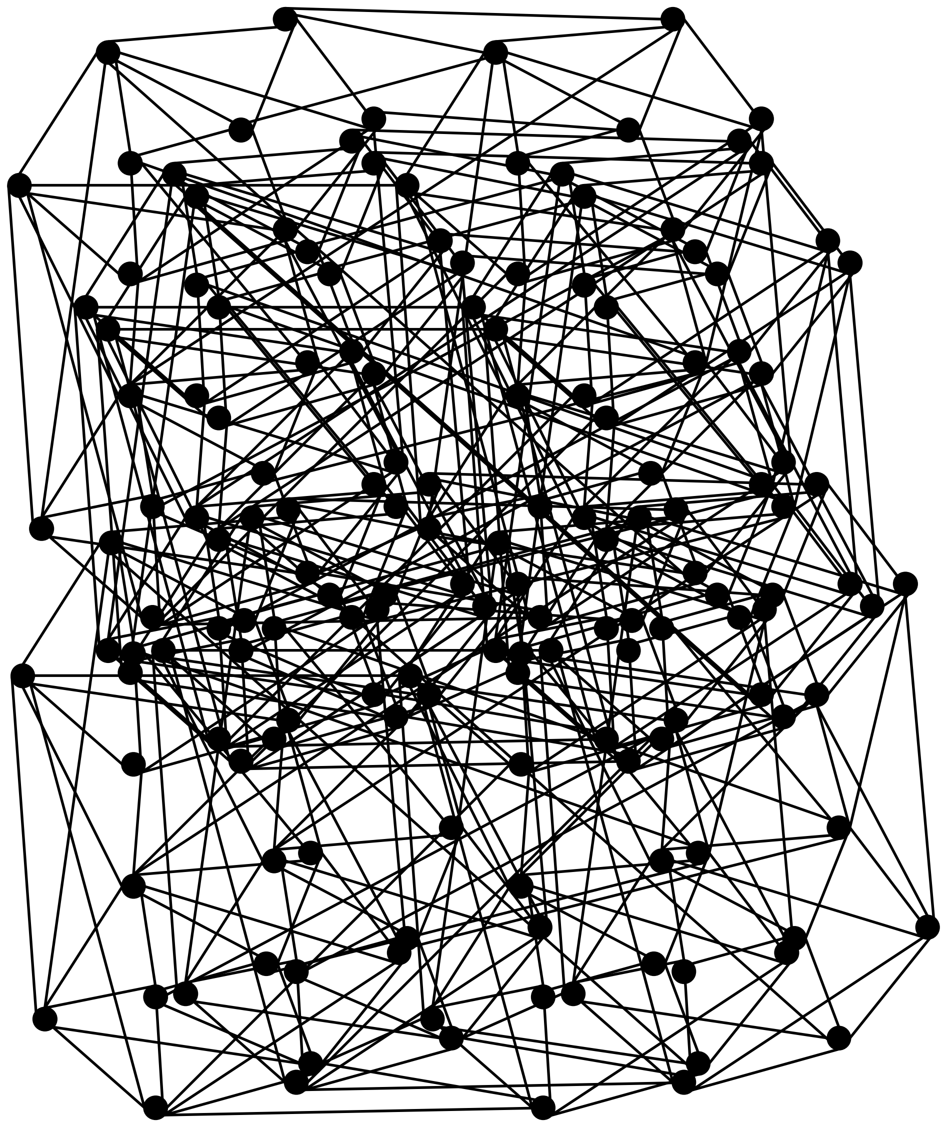


Vizinhanças

- Espaço de busca: grafo cujos vértices são as soluções, com arestas entre pares de vértices associados a soluções vizinhas.

- Caminho: seqüência de soluções, onde duas soluções consecutivas são vizinhas.

- Ótimo local: solução melhor ou tão boa quanto qualquer uma das soluções vizinhas



Busca local

- Algoritmos de busca local: estratégia de exploração do espaço de busca

1. Partida: solução inicial obtida através de um método construtivo
2. Iteração: melhoria da solução corrente através de uma busca na sua vizinhança
3. Parada: primeiro ótimo local encontrado (não existe solução vizinha aprimorante)

Busca local

- Melhoria iterativa: a cada iteração, selecionar qualquer (eventualmente a primeira) solução aprimorante na vizinhança

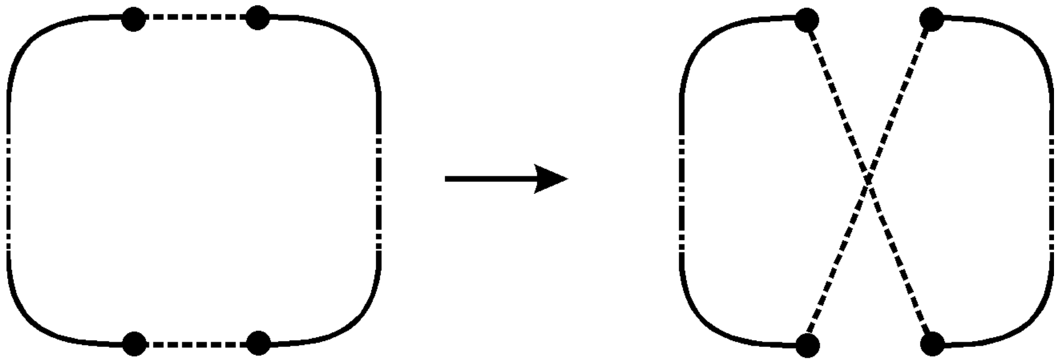
- Descida mais rápida: a cada iteração, selecionar a melhor solução aprimorante na vizinhança

Busca local

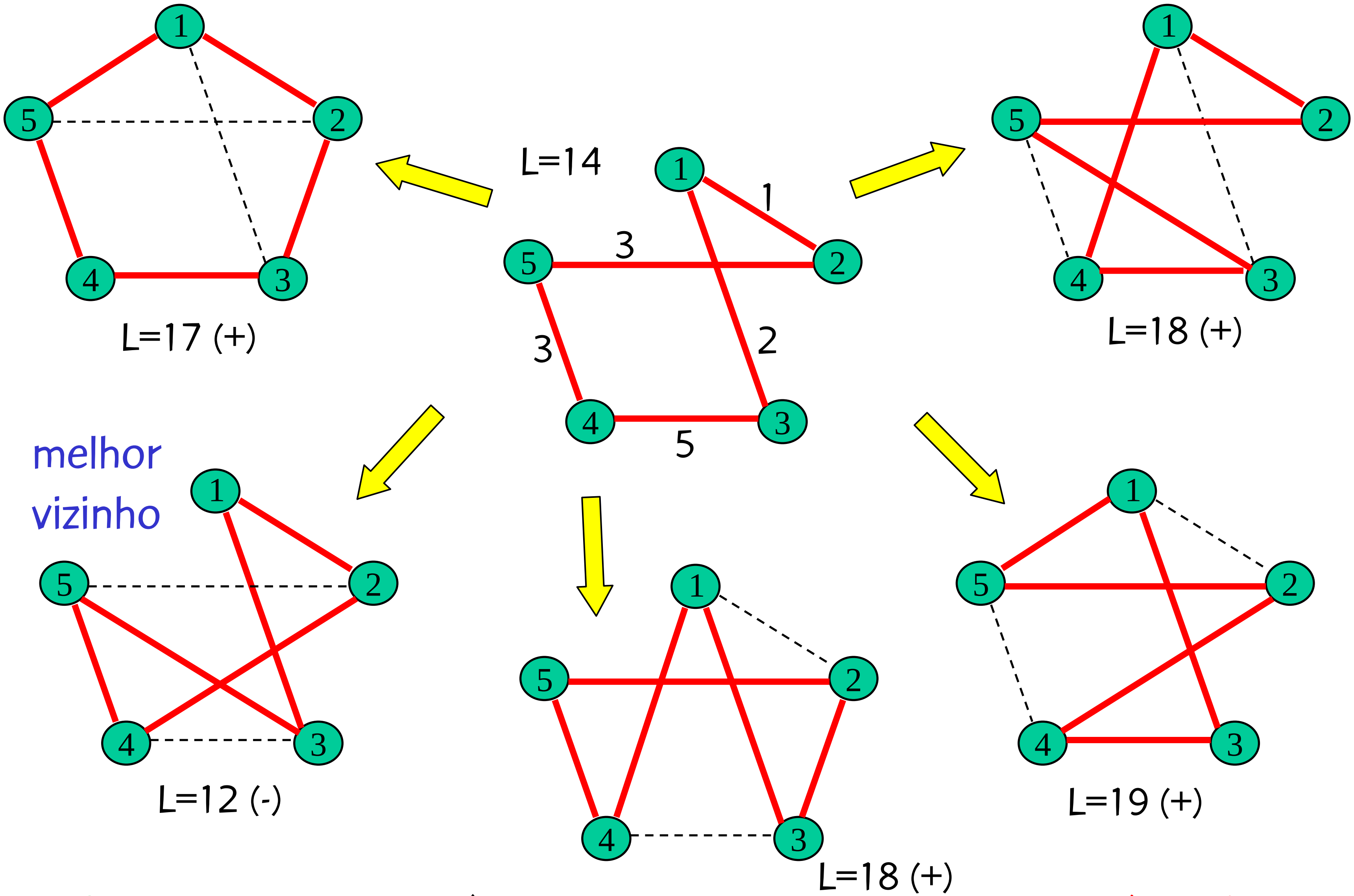
- Questões fundamentais:
  - Definição da vizinhança
  - Estratégia de busca na vizinhança
  - Complexidade de cada iteração
    - Proporcional ao tamanho da vizinhança
    - Eficiência depende da forma como é calculada a função objetivo para cada solução vizinha: algoritmos eficientes são capazes de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

Busca local

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

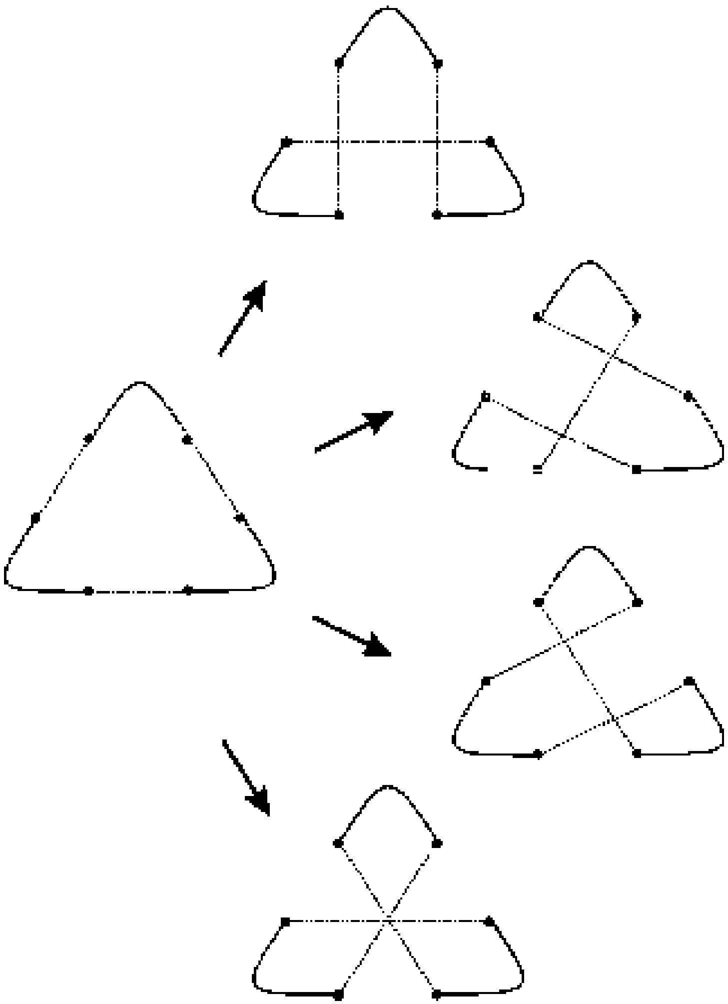


- Um vizinho para cada par de arestas: número de vizinhos é $O(n^2)$
- Custo de cada vizinho pode ser avaliado em $O(1)$: complexidade de cada iteração da busca local é $O(n^2)$


L=14
1
3
3
2
5
1
2
3
4
5
L=17 (+)
L=18 (+)
melhor
vizinho
L=12 (-)
L=18 (+)
L=19 (+)

Busca local

- Vizinhança 3-opt para o problema do caixeiro viajante
- Combinar diferentes vizinhanças no mesmo algoritmo de busca local



Busca local

- Um vizinho para cada tripla de arestas: número de vizinhos é $O(n^3)$
- Custo de cada vizinho pode ser avaliado em $O(1)$: complexidade de cada iteração da busca local é $O(n^3)$
- Vizinhança (k+1)-opt inclui as soluções de k-opt.
- Extensão até n-opt corresponderia a uma busca exaustiva do espaço de soluções!
- A complexidade de cada iteração aumenta com k, enquanto o ganho possível diminui.

Busca local

- Diferentes aspectos do espaço de busca influenciam o desempenho da busca local
- Conexidade: deve existir um caminho entre qualquer par de soluções no espaço de busca
- Distância entre duas soluções: número de soluções visitadas ao longo do caminho mais curto entre elas
- Diâmetro: distância entre as duas soluções mais afastadas (diâmetros reduzidos)

Busca local

- Dificuldades:
  - 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?

Metaheurísticas

- *Simulated annealing*
- Busca tabu
- GRASP
- VNS *(Variable Neighborhood Search)*
- Algoritmos genéticos
- *Scatter search*
- Colônias de formigas