Algoritmos Construtivos

Slides adaptados de material didático dos professores Adriana Alvim (UniRio), Celso Ribeiro (UFF) e Diego Luchi (egresso Doutorado UFES)

Laboratório de Otimização - UFES

Sumário

- Motivação
- Métodos de busca heurística
  - construtivos
  - busca local
  - metaheurísticas

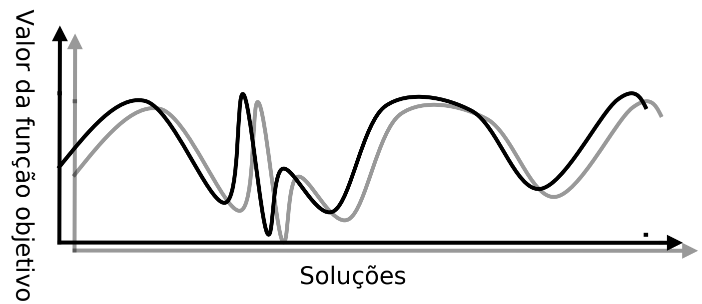
Motivação

- O fato de determinado problema ser caracterizado como NP-Completo é aceito como forte evidência contra a existência de algoritmos polinomiais e, consequentemente, como justificativa para a utilização de heurísticas
  - com intuito de se obter soluções aproximadas de boa qualidade

Métodos de busca

- Abordagens para resolver problemas combinatórios difíceis são em geral caracterizadas como métodos de busca
- Nesse contexto, pode-se definir mecanismos genéricos de busca de soluções:
  - sistemática (exata)
  - heurística
    - construtiva
    - busca local
    - meta-heurística

Espaço de soluções



Métodos de busca

- A ideia fundamental nos métodos de busca é
  - gerar e avaliar soluções candidatas de forma iterativa
    - problemas de decisão
      - avaliação = testa se é solução
    - problemas de otimização
      - avaliação = verifica valor da função objetivo

Métodos de busca

- Tipicamente avaliar soluções candidatas é mais barato do que encontrar soluções ótimas
  - para uma dada instância do PCV, uma solução candidata corresponde a um caminho de ida e volta que visita cada vértice do respectivo grafo uma única vez
  - *A*valiar a solução equivale a calcular o valor da função objetivo somando os pesos associados a cada arco do respectivo caminho

Métodos de busca

- A principal diferença entre os algoritmos de busca está na forma de gerar soluções candidatas

Soluções candidatas

- Soluções candidatas são compostas de
  - componentes da solução
    - Exemplo: PCV
      - Cada trecho entre duas cidades (ou dois vértices do grafo) que estão em sequencia representa uma componente da solução

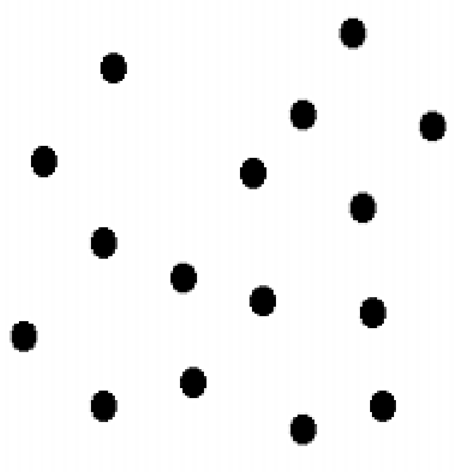
- Solução candidata parcial
  - faltam um ou mais componentes da solução
    - Exemplo: PCV
      - caminhos que visitam um subconjunto dos vértices do grafo

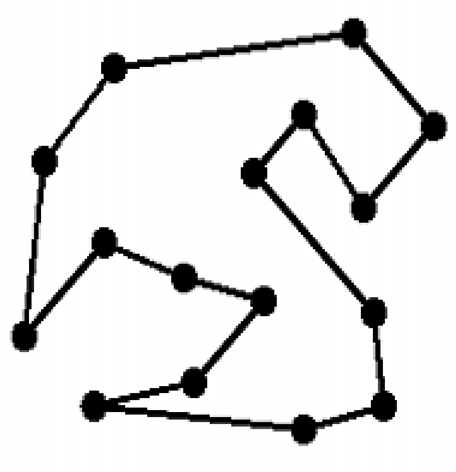
Métodos construtivos

- Método construtivo
  - espaço de busca = solução candidata parcial
  - passo da busca = extensão com um ou mais componentes da solução
  - objetivo
    - criar uma boa solução candidata

PCV X Ciclo Hamiltoniano

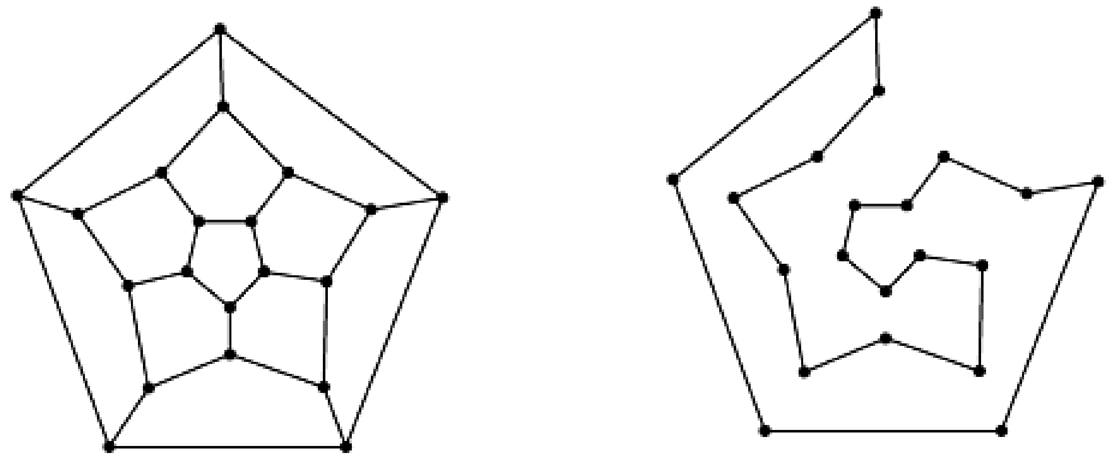
- Problema do caixeiro viajante (PCV)
  - dados um conjunto de n cidades, e uma distância para cada par de cidades, encontre o caminho mais curto ligando todas as cidades, sem visitar nenhuma cidade duas vezes
  - Entrada: $G=(V,E)$ grafo completo, e c_{ij} custo da aresta que liga os vértices i e j





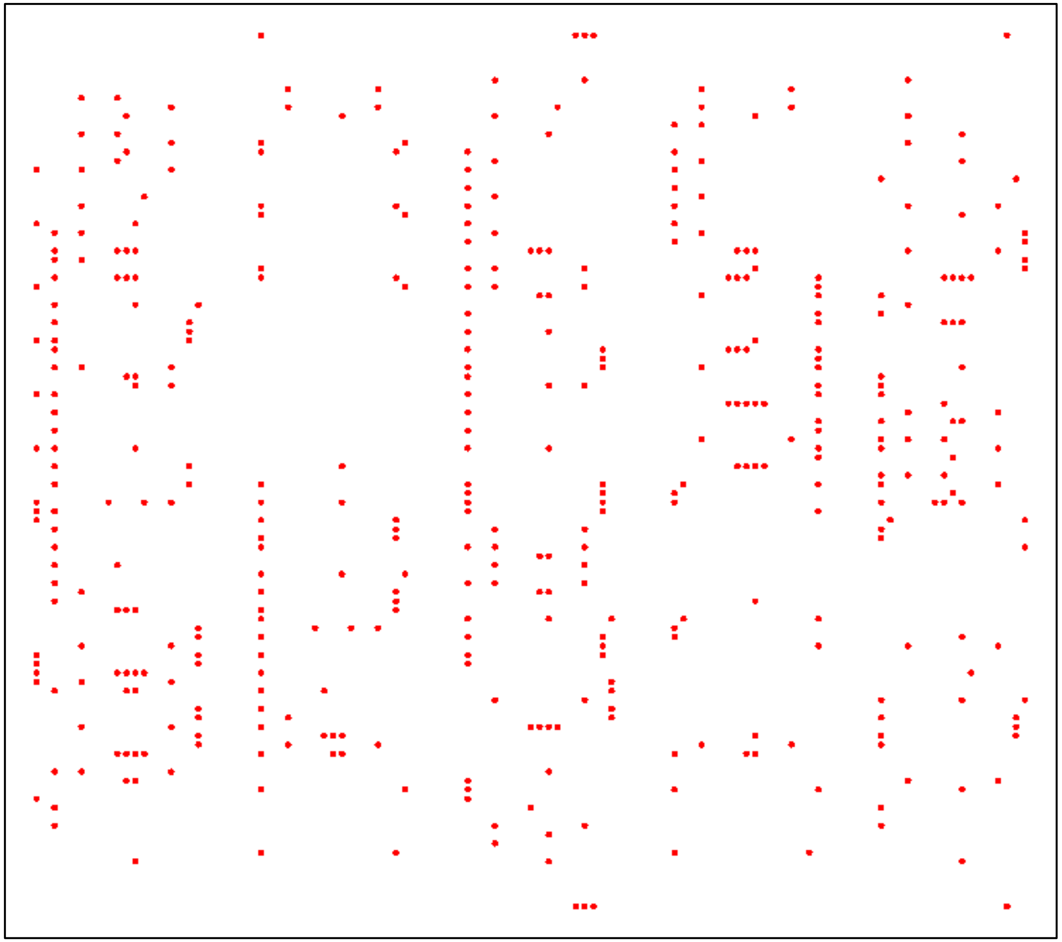
PCV X Ciclo Hamiltoniano

- Ciclo Hamiltoniano
  - Entrada G=(V,E) grafo completo
  - Encontre uma ordem para os vértices tal que cada vértice é visitado apenas uma vez



- O PCV equivale a encontrar no grafo G um Ciclo Hamiltoniano (CH) de custo mínimo

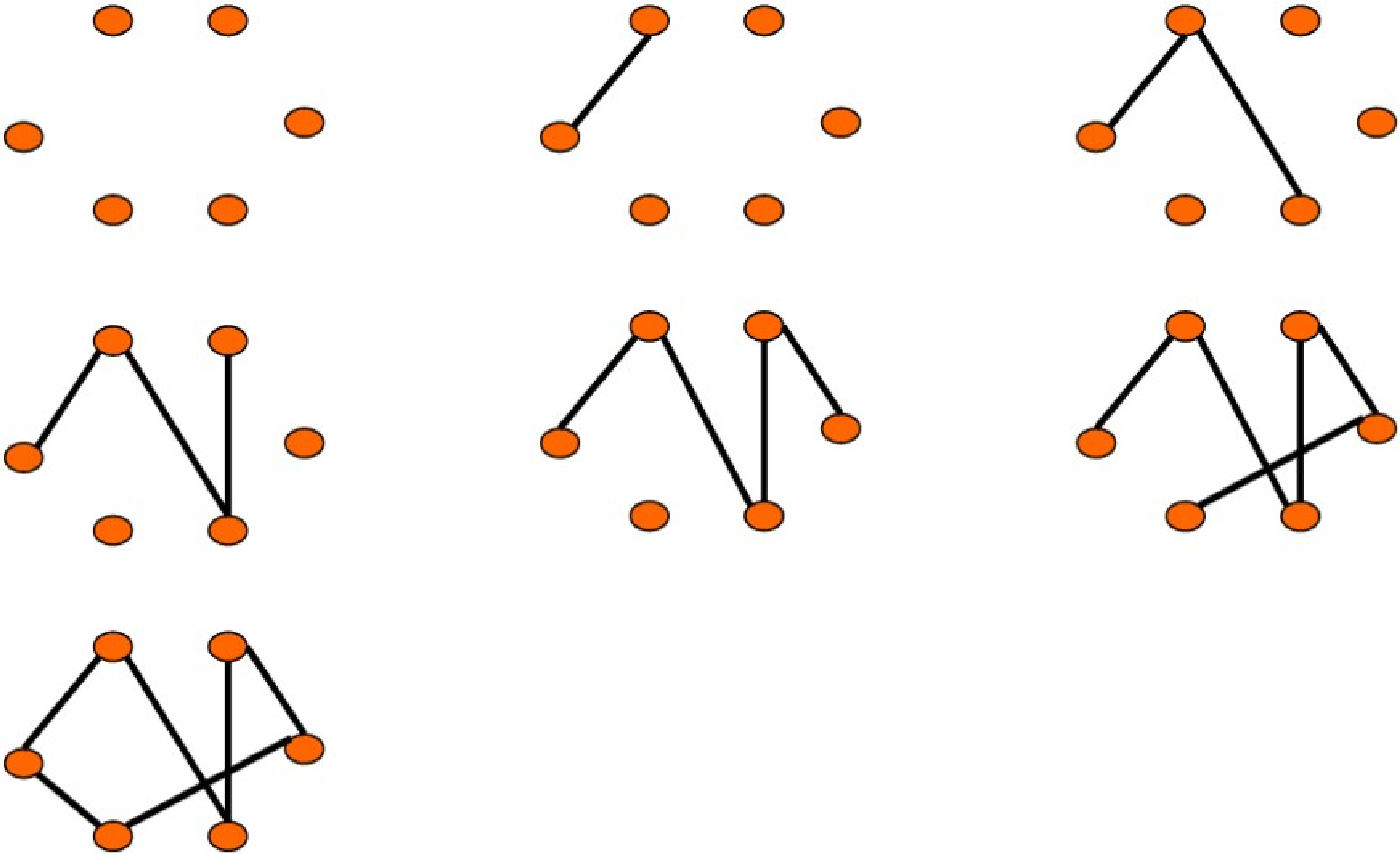
PCV – 423 cidades



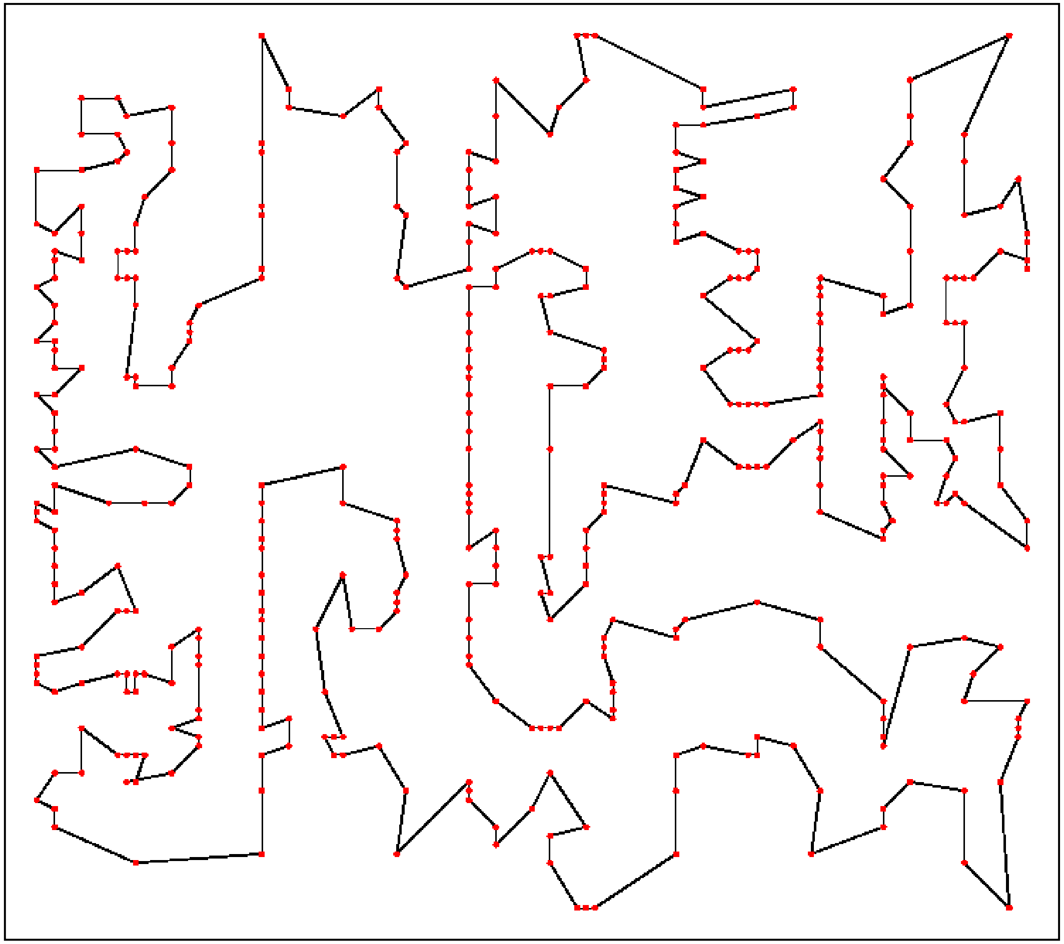
Métodos construtivos

- Algoritmo do vizinho mais próximo para o PCV
  - Nearest Neighbour Heuristic (NNH)

1. inicie com um nó arbitrário
2. procure o nó mais próximo do último nó adicionado que não esteja no caminho e adicione ao caminho a aresta que liga estes dois nós
3. quando todos os nós estiverem no caminho, adicione uma aresta conectando o nó inicial ao último nó adicionado



PCV – 423 cidades



Algoritmo do vizinho mais próximo

Entrada: grafo completo $G(N,E)$

Passo 0

$H \leftarrow \{1\}$; $N \leftarrow N-1$; $L \leftarrow 0$; $i \leftarrow 1$

Passo 1

if $N = \oslash$ fazer $L \leftarrow L + c_{i1}$ e terminar;

Passo 2

obter $j \in N : c_{ij} = \min_{k \in N} \{c_{ik}\}$;

Passo 3

$H \leftarrow H \cup \{j\}$;

$L \leftarrow L + c_{ij}$;

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

$i \leftarrow j$;

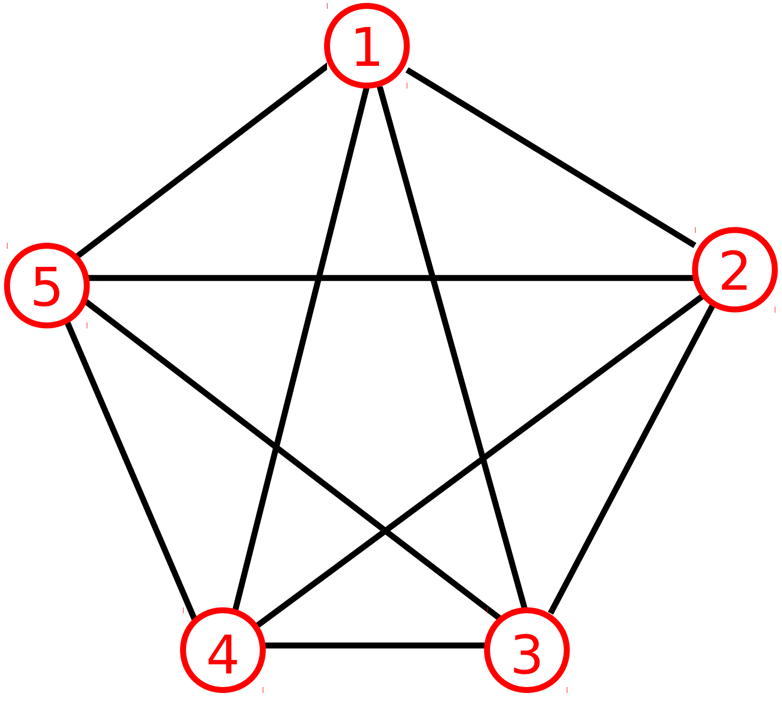
Retornar ao passo 1;

Algoritmo do vizinho mais próximo

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



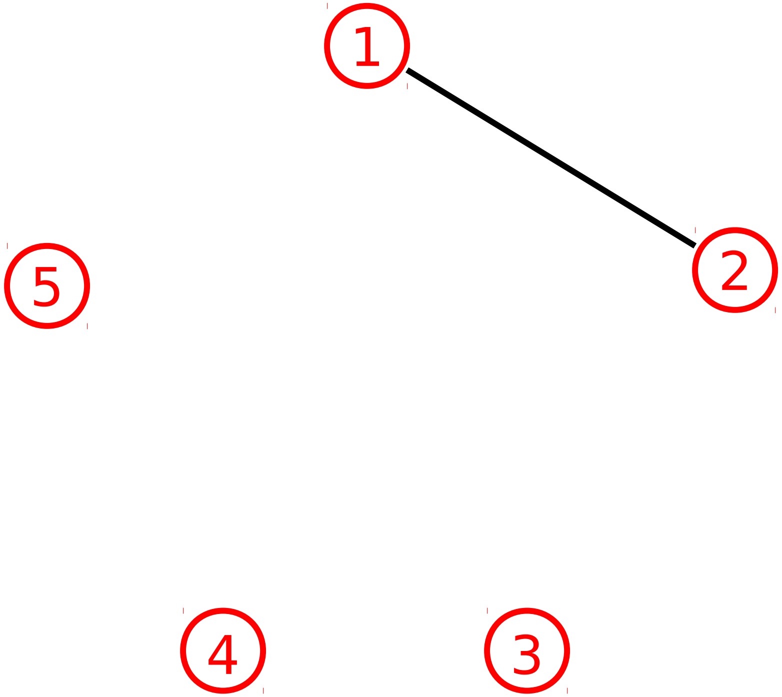
L=0, H={1}

Algoritmo do vizinho mais próximo

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



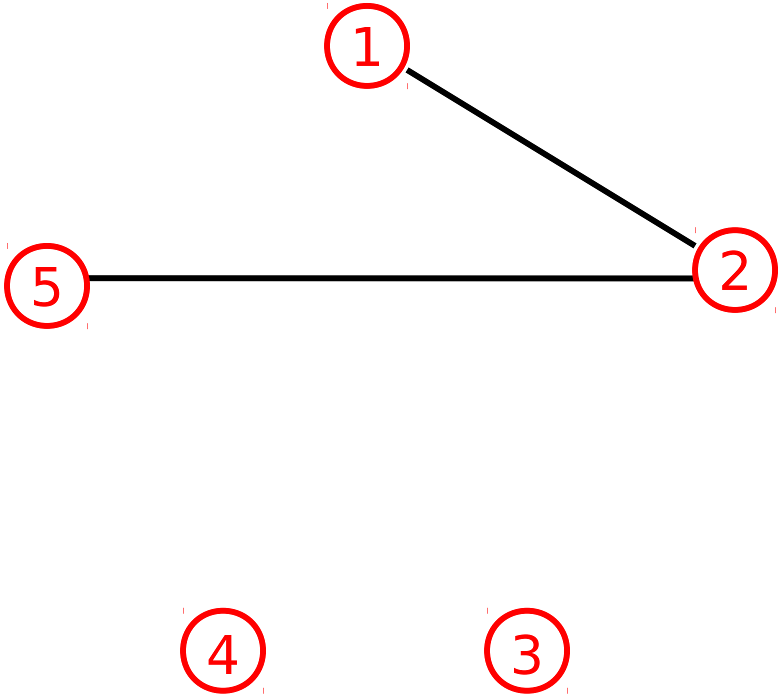
L=1, H={1,2}

Algoritmo do vizinho mais próximo

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



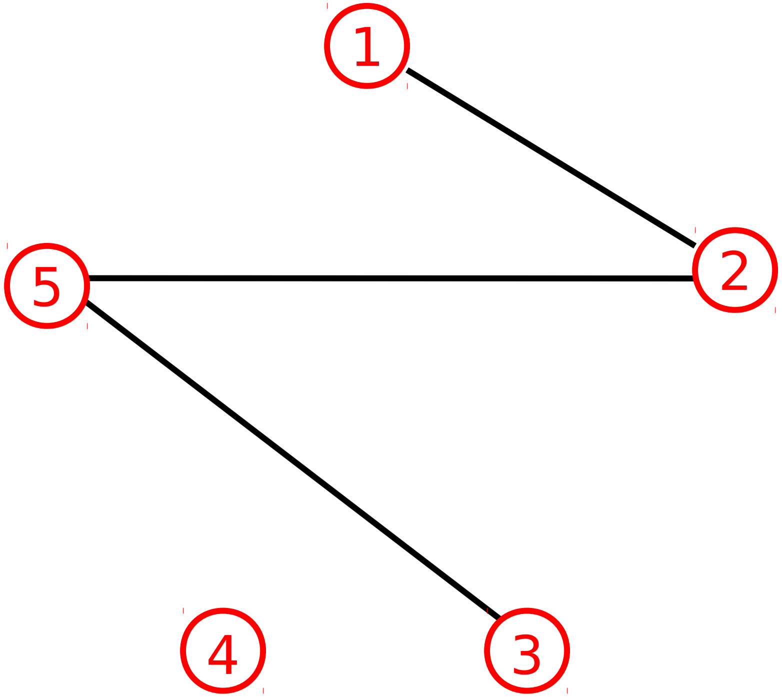
L=4, H={1,2,5}

Algoritmo do vizinho mais próximo

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



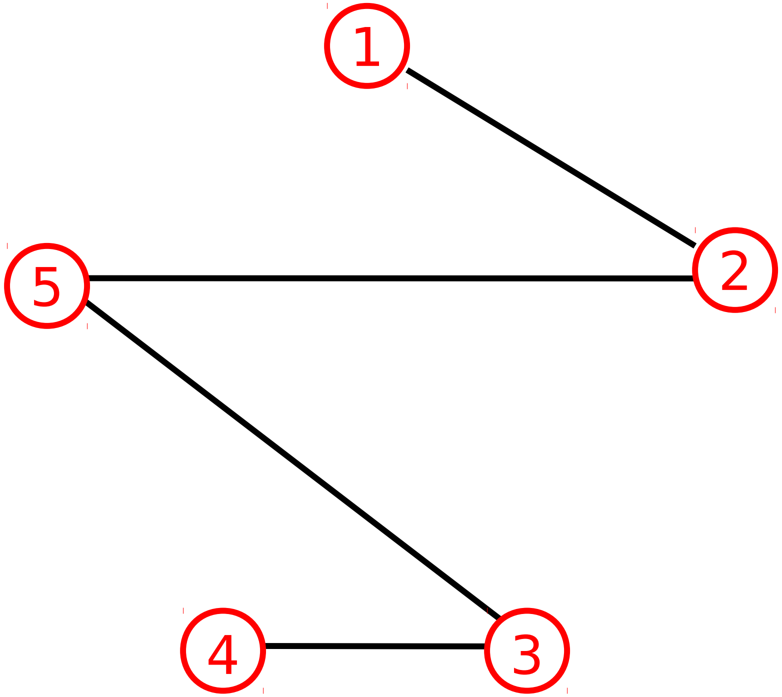
L=6,
H={1,2,5,3}

Algoritmo do vizinho mais próximo

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



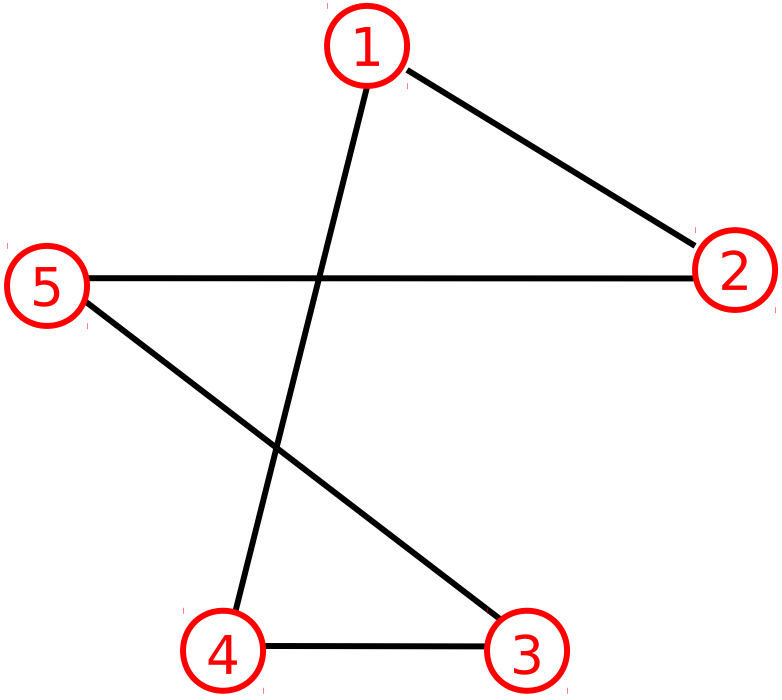
L=11,
H={1,2,5,3,4}

Algoritmo do vizinho mais próximo

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



L=18, H={1,2,5,3,4,1}

Algoritmo do vizinho mais próximo

- Este algoritmo tem complexidade $O(n^2)$
- O ciclo Hamiltoniano obtido sofre forte influência da escolha do nó inicial
- Para eliminar esta influência, basta aplicar o procedimento a partir de cada nó
  - o algoritmo resultante terá complexidade $O(n^3)$
- Aspecto negativo
  - embora todas as arestas escolhidas localmente sejam localmente mínimas, a aresta final pode ser bastante grande

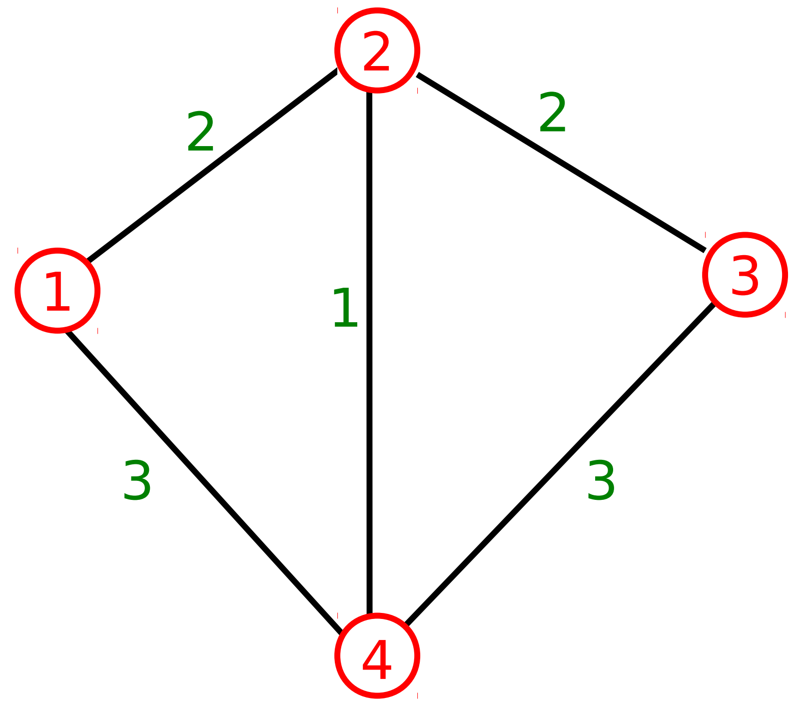
Algoritmo do vizinho mais próximo

- Embora o algoritmo do vizinho mais próximo não encontre a solução ótima
  - a obtida está bem próxima do ótimo
- Entretanto, é possível encontrar instâncias em que a solução obtida pode ser muito ruim
  - pode mesmo ser arbitrariamente ruim, uma vez que a aresta final pode ser muito longa

Algoritmo do vizinho mais próximo

- Se o grafo não for completo o algoritmo pode falhar ou terminar sem obter um tour (ciclo hamiltoniano)

- O grafo termina com L={1,2,4,3,1} que não é um tour

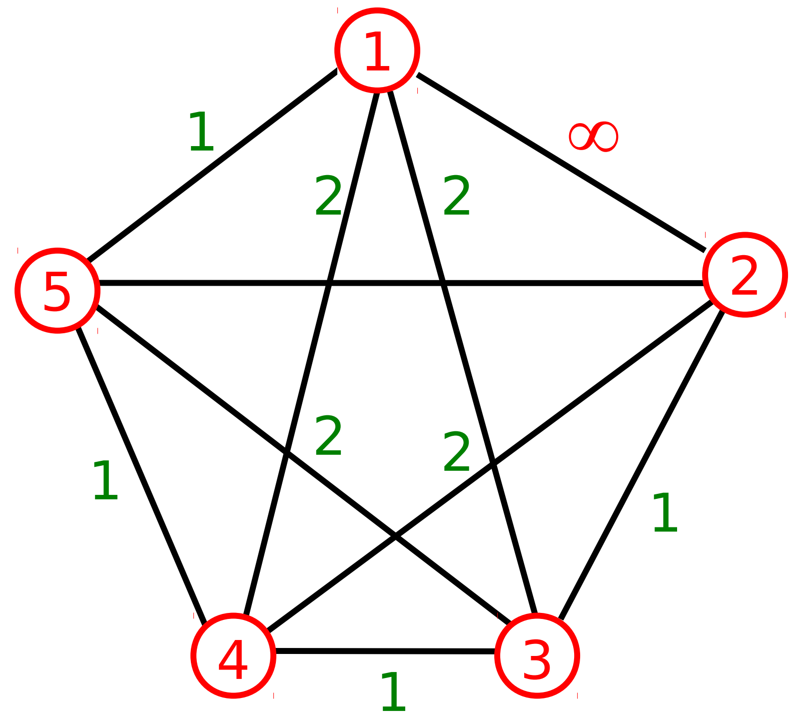


H={1,2,4,3,1}

Algoritmo do vizinho mais próximo

- Para problemas em que a desigualdade triangular não se verifica
  - problemas em que a desigualdade triangular se verifica, isto é

    $c_{ij} + c_{jk} \geq c_{ik} \ \forall\, i, j, k \in N$
  - possibilidade de obter soluções arbitrariamente ruins



H={1,5,4,3,2,1}

Algoritmo do vizinho mais próximo

- Tipicamente o algoritmo do vizinho mais próximo não encontra soluções de alta qualidade
  - porém, frequentemente é usado com sucesso em combinação com outros métodos de busca

Método construtivo e busca sistemática

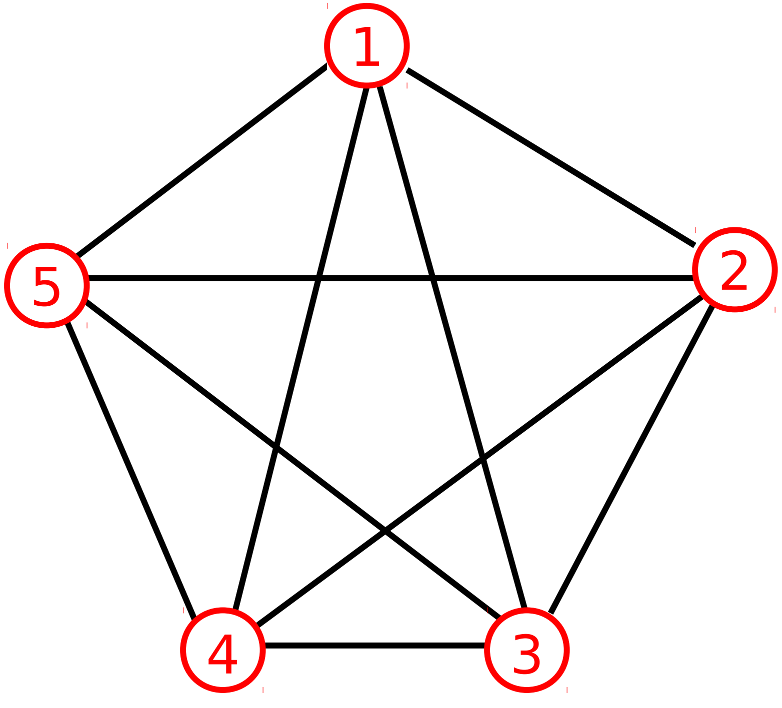
- Exemplo: Heurística do Vizinho mais Próximo para o PCV combinada com backtracking
  - em cada ponto do algoritmo construtivo onde pode ser feita uma escolha (incluindo o vértice inicial), guarda-se uma lista de todas as alternativas ainda não visitadas
  - quando um caminho completo tiver sido criado
  - o processo de busca backtracks para a escolha mais recente onde existe alguma alternativa ainda não explorada
  - a busca construtiva é finalizada usando um vértice diferente neste ponto
    - e possivelmente criando novas listas de alternativas

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



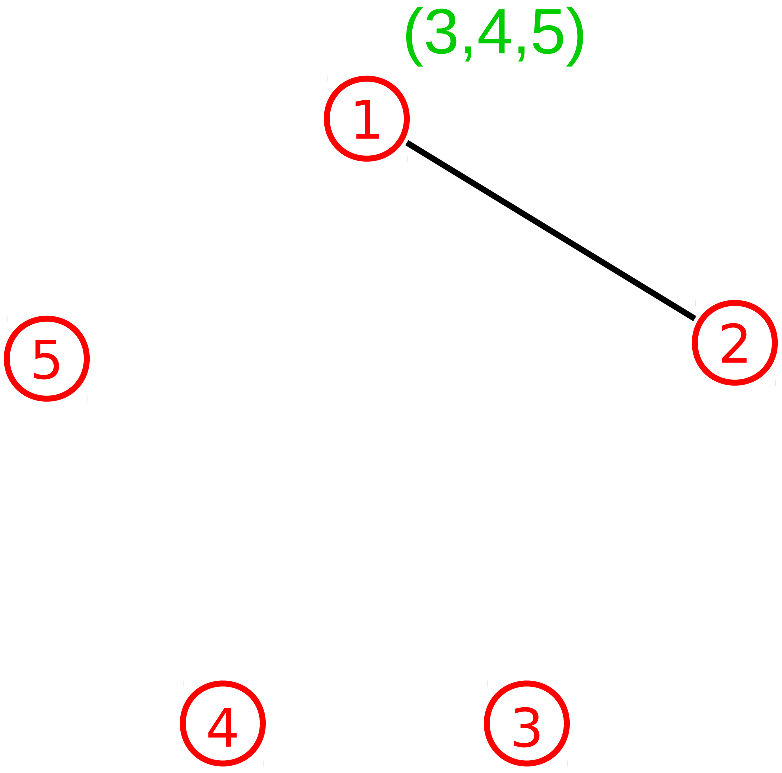
L=0, H={1}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



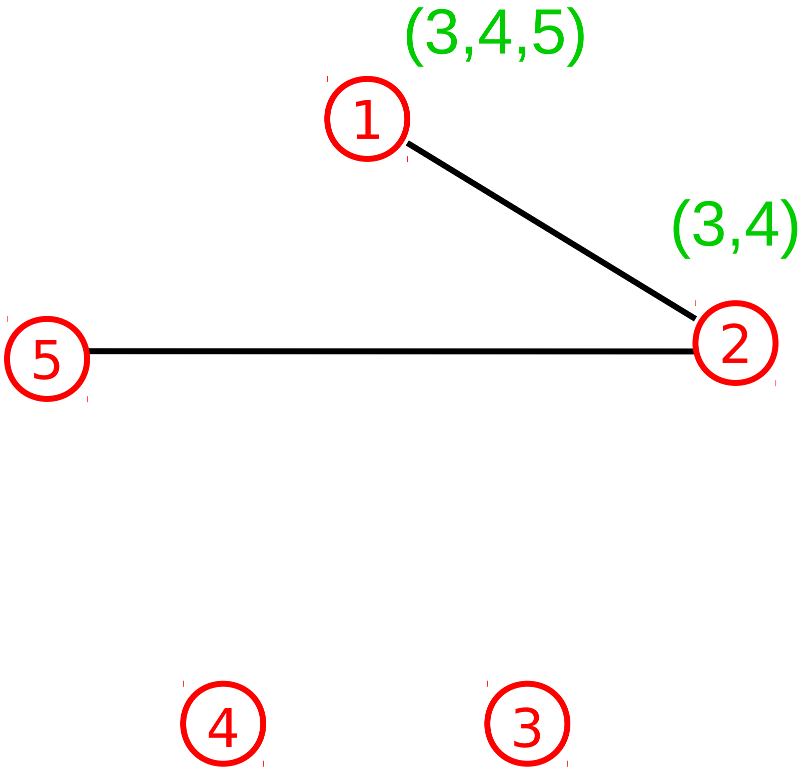
L=1, H={1,2}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



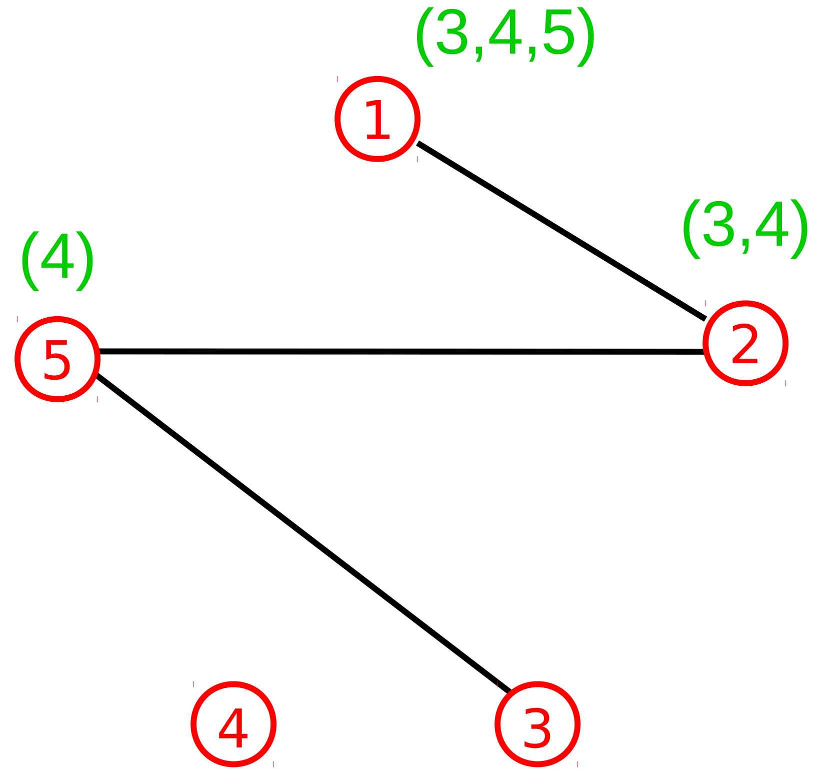
L=4, H={1,2,5}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



L=6,
H={1,2,5,3}

Vizinho mais próximo com backtrack

- Soluções visitadas

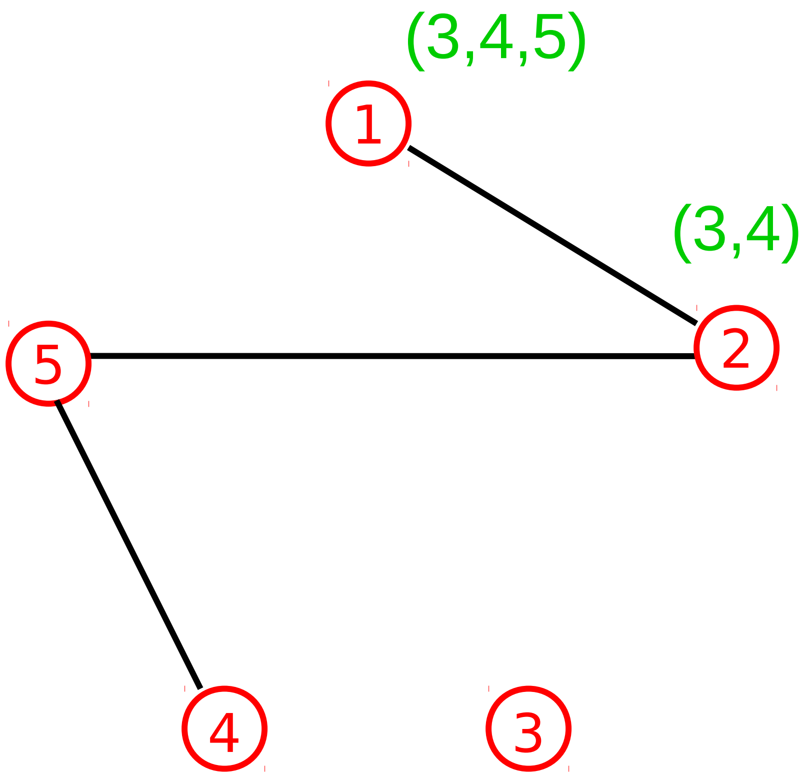
1. L=18, H={1,2,5,3,4,1}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



L=7,
H={1,2,5,4}

Vizinho mais próximo com backtrack

- Soluções visitadas

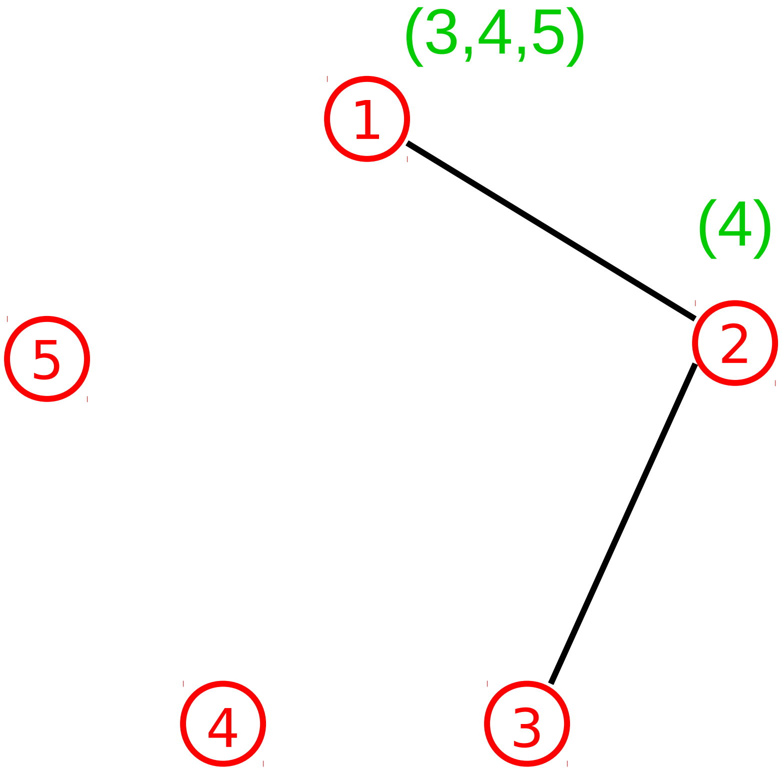
1. L=18, H={1,2,5,3,4,1}
2. L=14, H={1,2,5,4,3,1}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



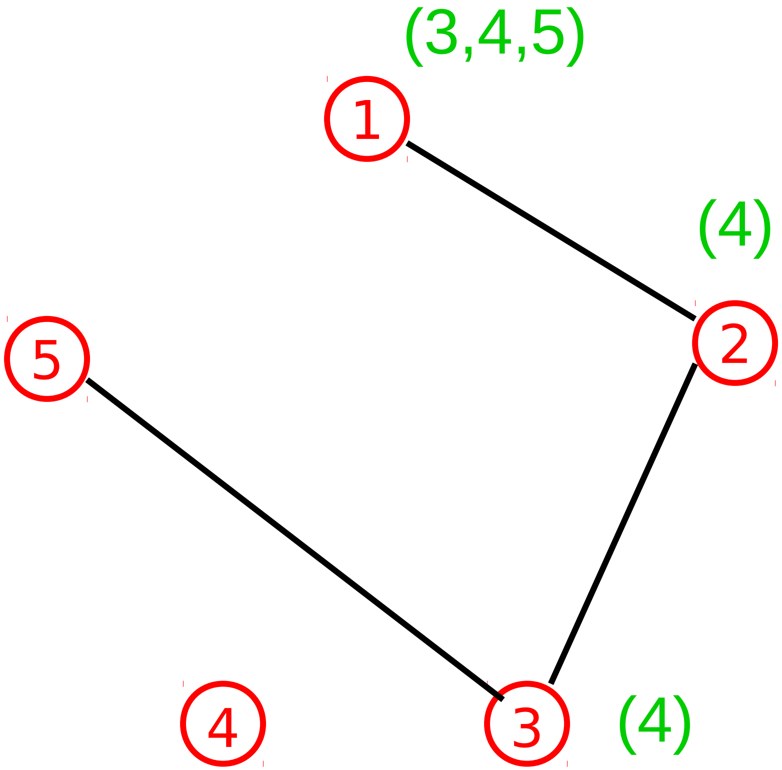
L=4, H={1,2,3}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



L=6,
H={1,2,3,5}

Vizinho mais próximo com backtrack

- Soluções visitadas

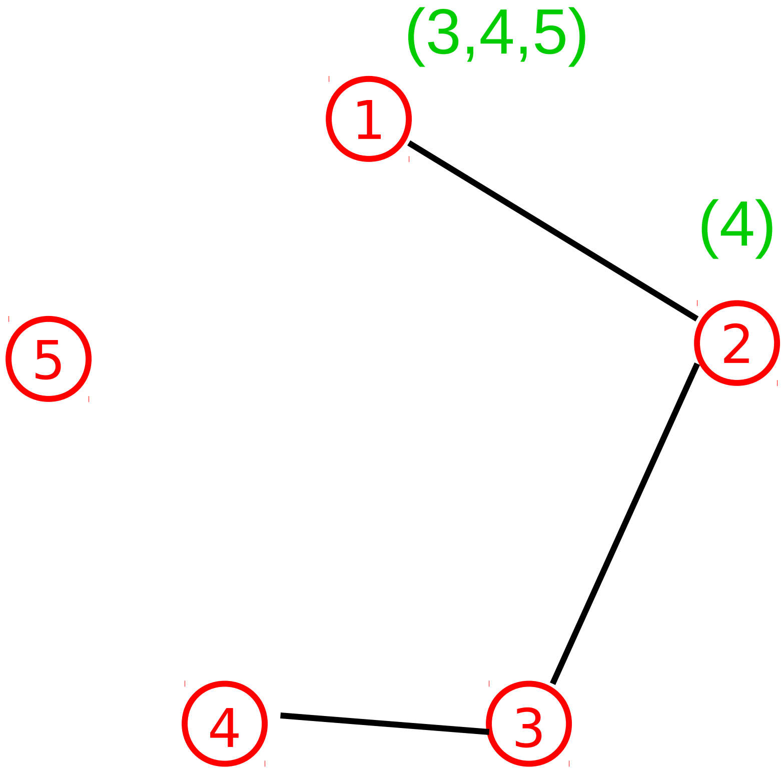
1. L=18, H={1,2,5,3,4,1}
2. L=14, H={1,2,5,4,3,1}
3. L=16, H={1,2,3,5,4,1}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



L=9,
H={1,2,3,4}

Vizinho mais próximo com backtrack

- Soluções visitadas

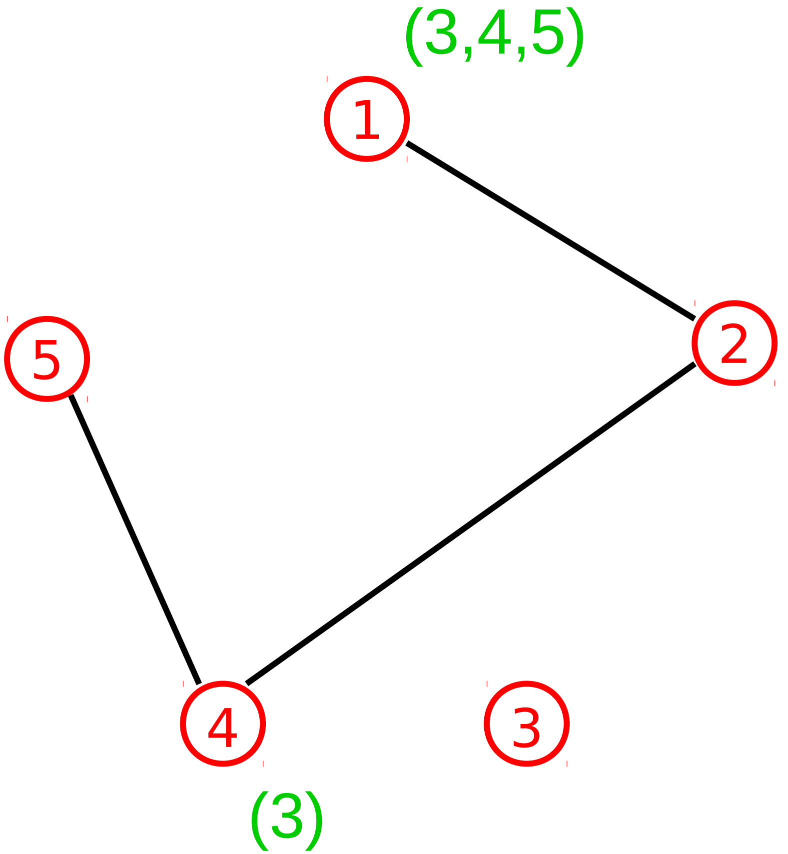
1. L=18, H={1,2,5,3,4,1}
2. L=14, H={1,2,5,4,3,1}
3. L=16, H={1,2,3,5,4,1}
4. L=17, H={1,2,3,4,5,1}

Vizinho mais próximo com backtrack

Exemplo com n=5

matriz de distâncias

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



L=8,
H={1,2,4,5}

Vizinho mais próximo com backtrack

- Soluções visitadas

1. L=18, H={1,2,5,3,4,1}
2. L=14, H={1,2,5,4,3,1}
3. L=16, H={1,2,3,5,4,1}
4. L=17, H={1,2,3,4,5,1}
5. L=12, H={1,2,4,5,3,1}
6. L=17, H={1,2,4,3,5,1}, etc...
    - 24 soluções considerando caminhos simétricos

Método construtivo e busca sistemática

- Visitar todas as soluções candidatas leva a um algoritmo exponencial no tamanho da entrada do problema
  - em várias situações é possível melhorar a eficiência desse método eliminando escolhas que não levam a solução ótima
  - exemplo: no caso do PCV a busca em uma direção (ramo da árvore de busca) pode ser terminada se o tamanho do caminho parcial corrente mais um limite inferior para o caminho restante for maior do que o valor da melhor solução conhecida até o momento
  - esse tipo de algoritmo é conhecido como branch and bound

Algoritmo de inserção mais próxima

Passo 0

Iniciar com um ciclo de comprimento 3

Passo 1

Encontrar o vértice k fora do ciclo corrente cuja aresta de menor comprimento que o liga a este ciclo é mínima

Passo 2

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

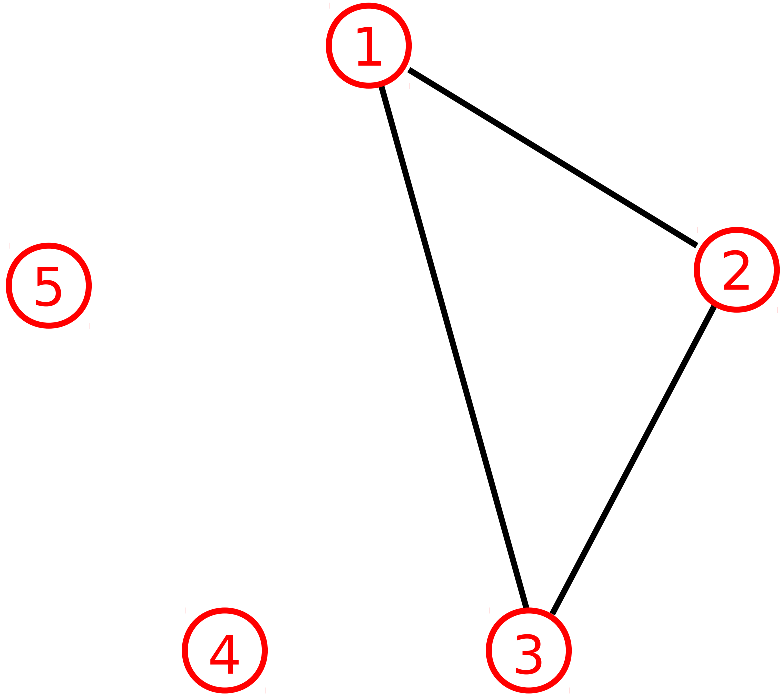
Inserir as arestas (i,k) e (k,j) e retirar a aresta (i,j)

Passo 3

Retornar ao passo 1

Algoritmo de inserção mais próxima

- Iniciar com um ciclo de comprimento 3

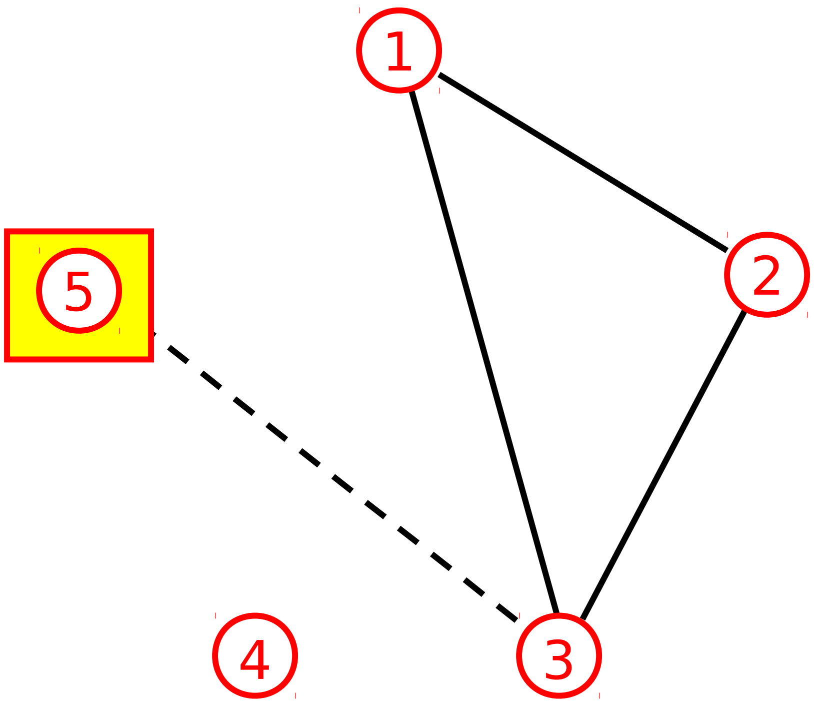


-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

L=1+3+2=6

Algoritmo de inserção mais próxima

- Encontrar o vértice k fora do ciclo corrente cuja aresta de menor comprimento que o liga a este ciclo é mínima



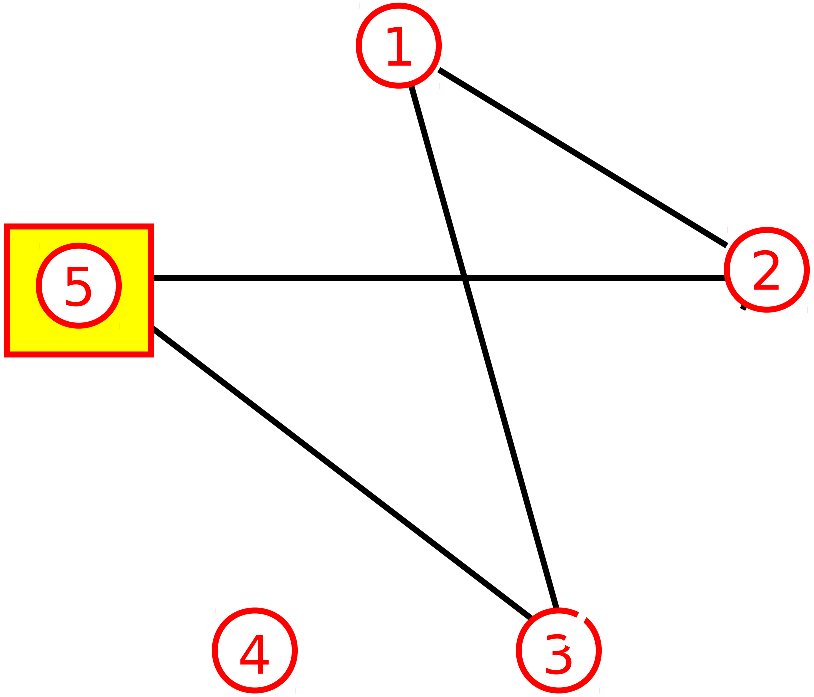
-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

$L=1+3+2=6$

Algoritmo de inserção mais próxima

- Encontrar o par de arestas (i,k) e (k,j) que ligam o vértice k ao ciclo minimizando $c_{ik} + c_{kj} - c_{ij}$
- Inserir as arestas (i,k) e (k,j) e retirar a aresta (i,j)

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



$\Delta 12(5)=5+3-1=7$

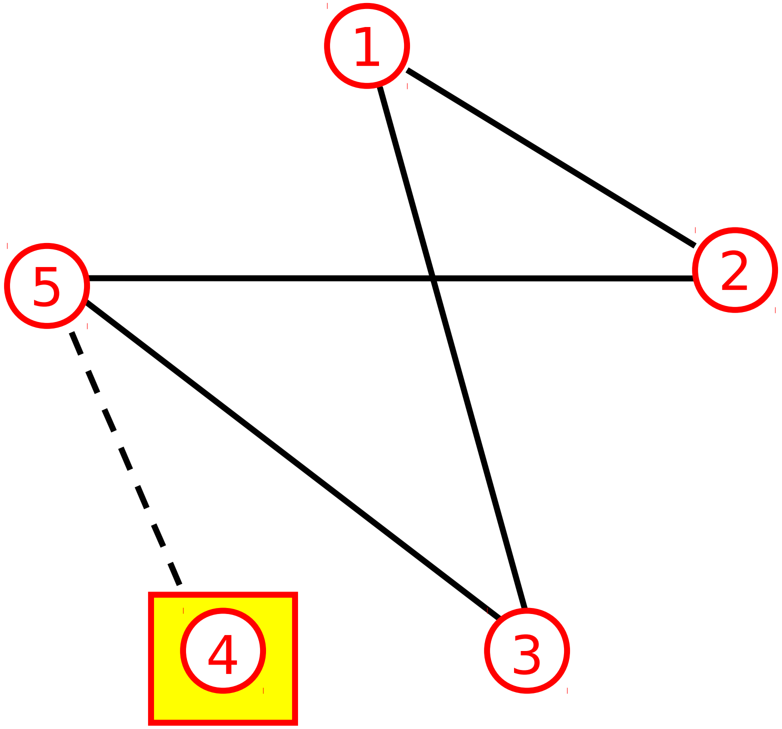
$\Delta 23(5)=3+2-3=2$

$\Delta 13(5)=5+2-2=5$

Algoritmo de inserção mais próxima

- Encontrar o vértice k fora do ciclo corrente cuja aresta de menor comprimento que o liga a este ciclo é mínima

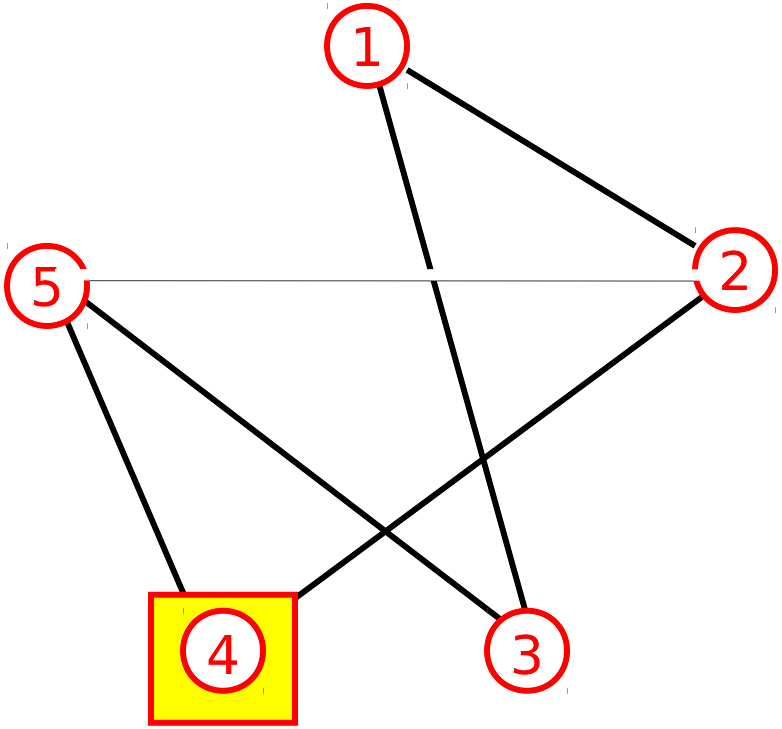
-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



Algoritmo de inserção mais próxima

- Encontrar o par de arestas (i,k) e (k,j) que ligam o vértice k ao ciclo minimizando $c_{ik} + c_{kj} - c_{ij}$
- Inserir as arestas (i,k) e (k,j) e retirar a aresta (i,j)

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



Δ12(4)=7+4-1=10

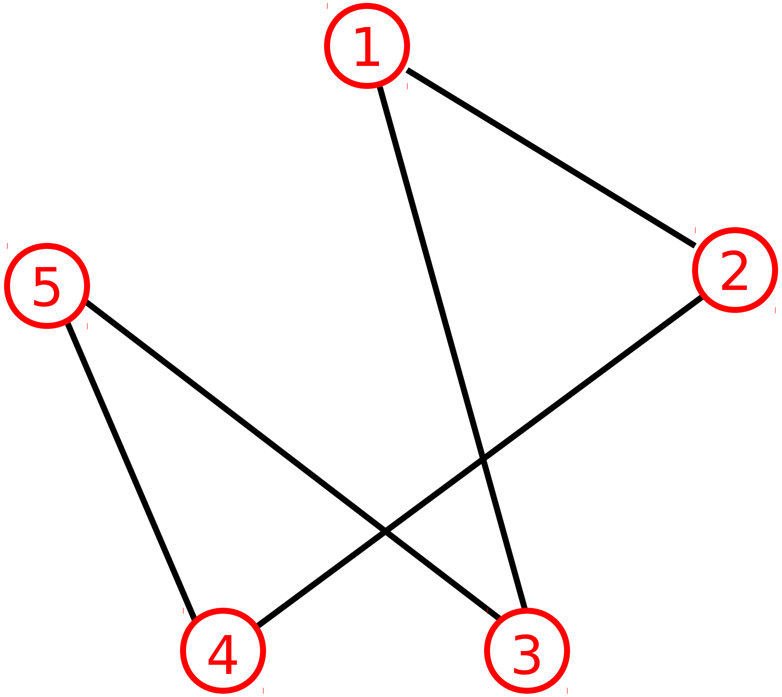
Δ13(4)=7+5-2=10

Δ25(4)=3+4-3=4

Δ35(4)=5+3-2=6

Algoritmo de inserção mais próxima

- L=1+2+4+2+3 =12



-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

Algoritmo de inserção mais distante

Passo 0

Iniciar com um ciclo de comprimento 3

Passo 1

Encontrar o vértice k fora do ciclo corrente cuja aresta de menor comprimento que o liga a este ciclo é máxima

Passo 2

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

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

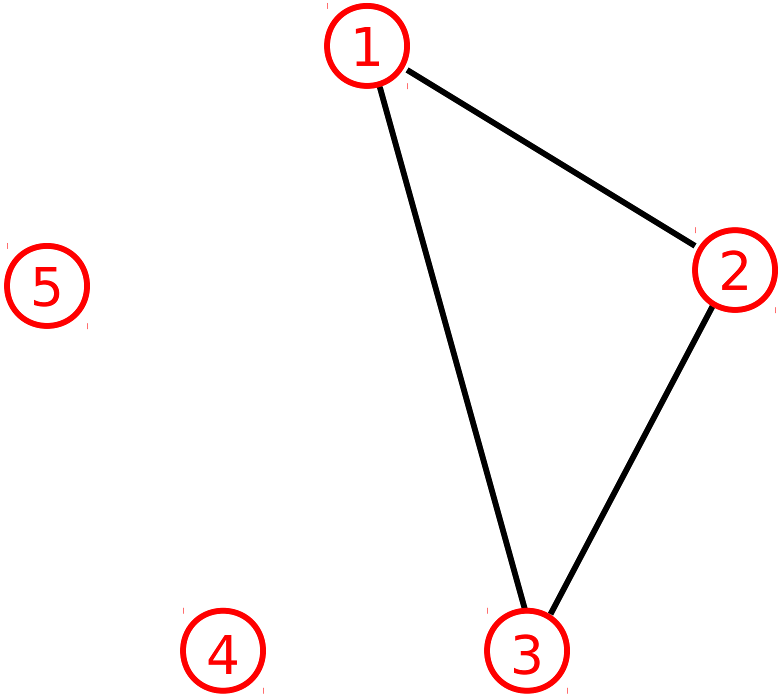
Passo 3

Retornar ao passo 1

Algoritmo de inserção mais distante

- Iniciar o ciclo com apenas um vértice

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

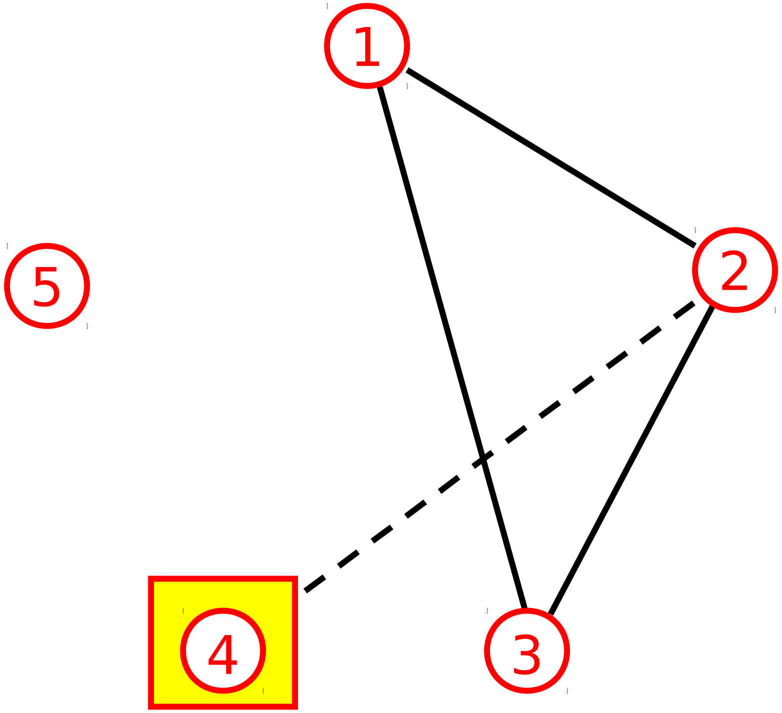


L=1+2+3=6

Algoritmo de inserção mais distante

- Encontrar o vértice k fora do ciclo corrente cuja aresta de menor comprimento que o liga a este ciclo é máxima

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

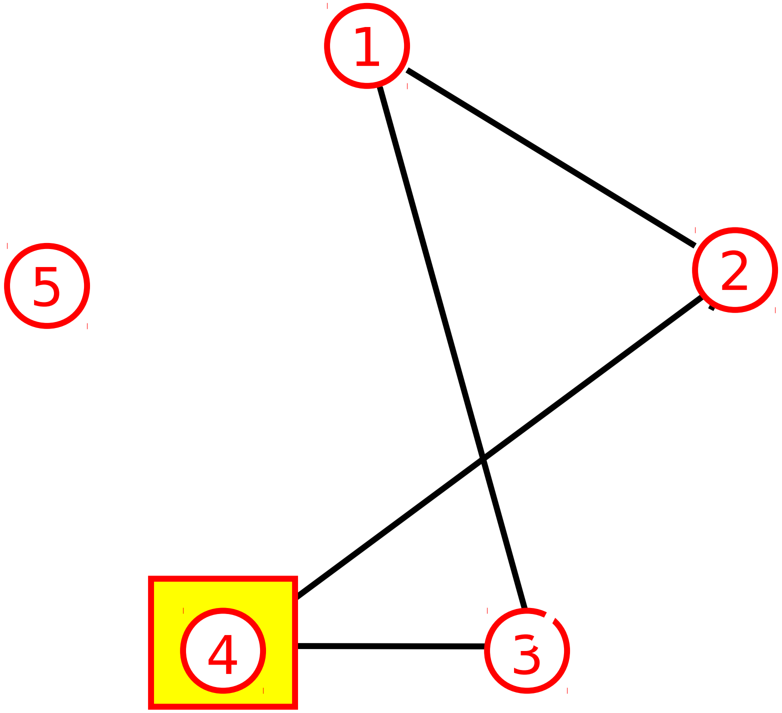


L=6

Algoritmo de inserção mais distante

- Encontrar o par de arestas (i,k) e (k,j) que ligam o vértice k ao ciclo minimizando $c_{ik} + c_{kj} - c_{ij}$
- Inserir as arestas (i,k) e (k,j) e retirar a aresta (i,j)

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



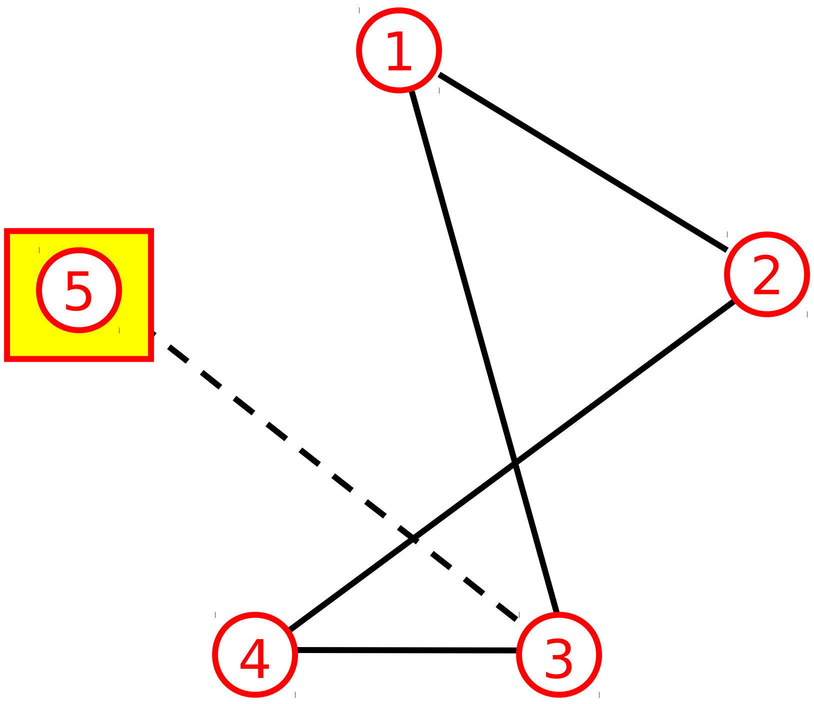
Δ12(4)=7+4-1=10

Δ23(4)=4+5-3=6

Δ13(4)=7+5-2=10

Algoritmo de inserção mais distante

- Encontrar o vértice k fora do ciclo corrente cuja aresta de menor comprimento que o liga a este ciclo é máxima

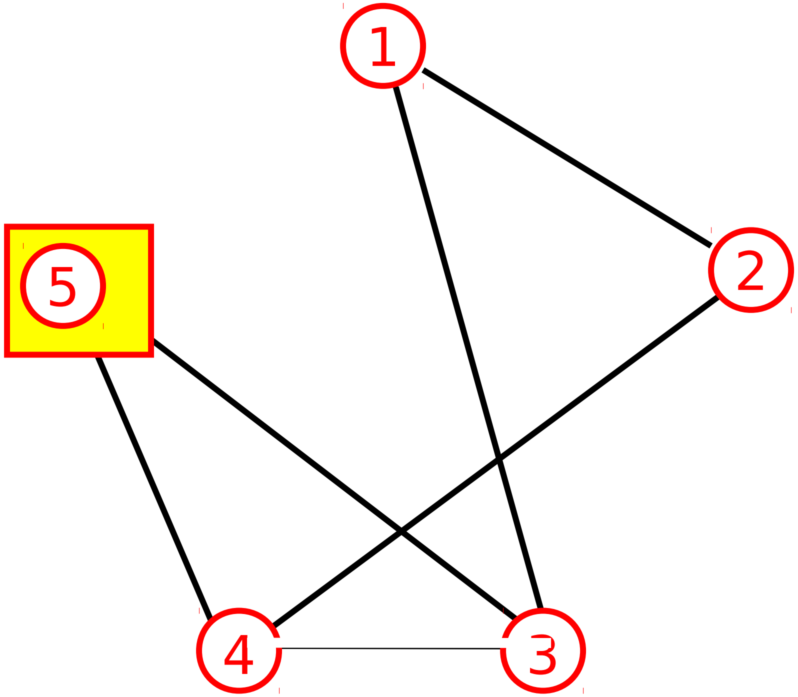


-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

Algoritmo de inserção mais distante

- Encontrar o par de arestas (i,k) e (k,j) que ligam o vértice k ao ciclo minimizando $c_{ik} + c_{kj} - c_{ij}$
- Inserir as arestas (i,k) e (k,j) e retirar a aresta (i,j)

-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-



Δ12(5)=5+3-1=7

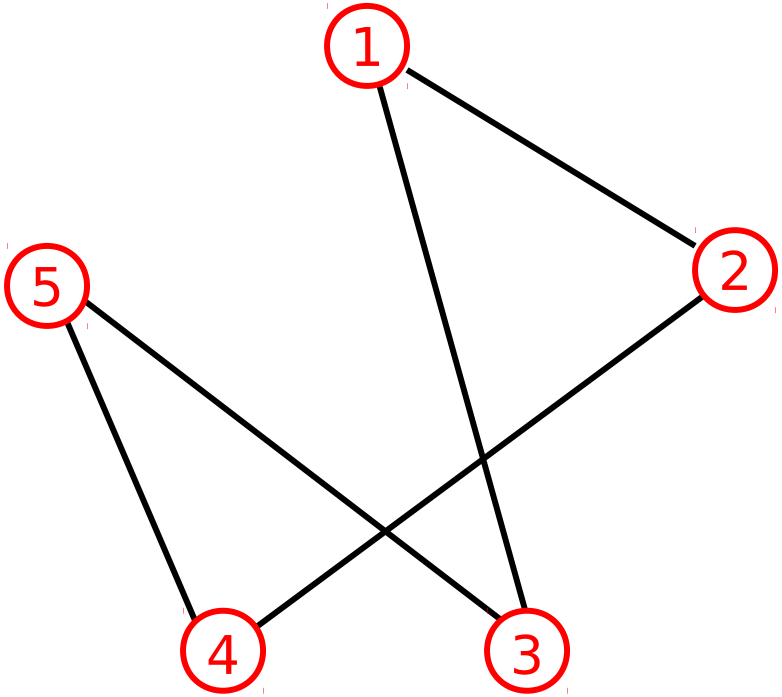
Δ13(5)=5+2-2=5

Δ24(5)=3+3-4=2

Δ34(5)=2+3-5=0

Algoritmo de inserção mais distante

- $L = 1 + 4 + 3 + 2 + 2 = 12$



-	1	2	7	5
1	-	3	4	3
2	3	-	5	2
7	4	5	-	3
5	3	2	3	-

Algoritmo de inserção mais barata

Passo 0

Iniciar com um ciclo de comprimento 3

Passo 1

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

Passo 2

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

Passo 3

Retornar ao passo 1

Algoritmo de economias - exercício

1. Escolher um vértice base inicial
   - por exemplo o vértice 1
   - construir ciclos de comprimento 2 deste vértice a cada um dos n -1 vértices restantes
2. A cada passo e enquanto existirem mais de um ciclo
   - dois ciclos são combinados eliminando dois arcos e adicionando um arco de ligação
   - os ciclos são escolhidos para o processo de combinação de forma a maximizar a distância economizada em relação à configuração anterior
3. Repete-se o procedimento até obter um Ciclo Hamiltoniano

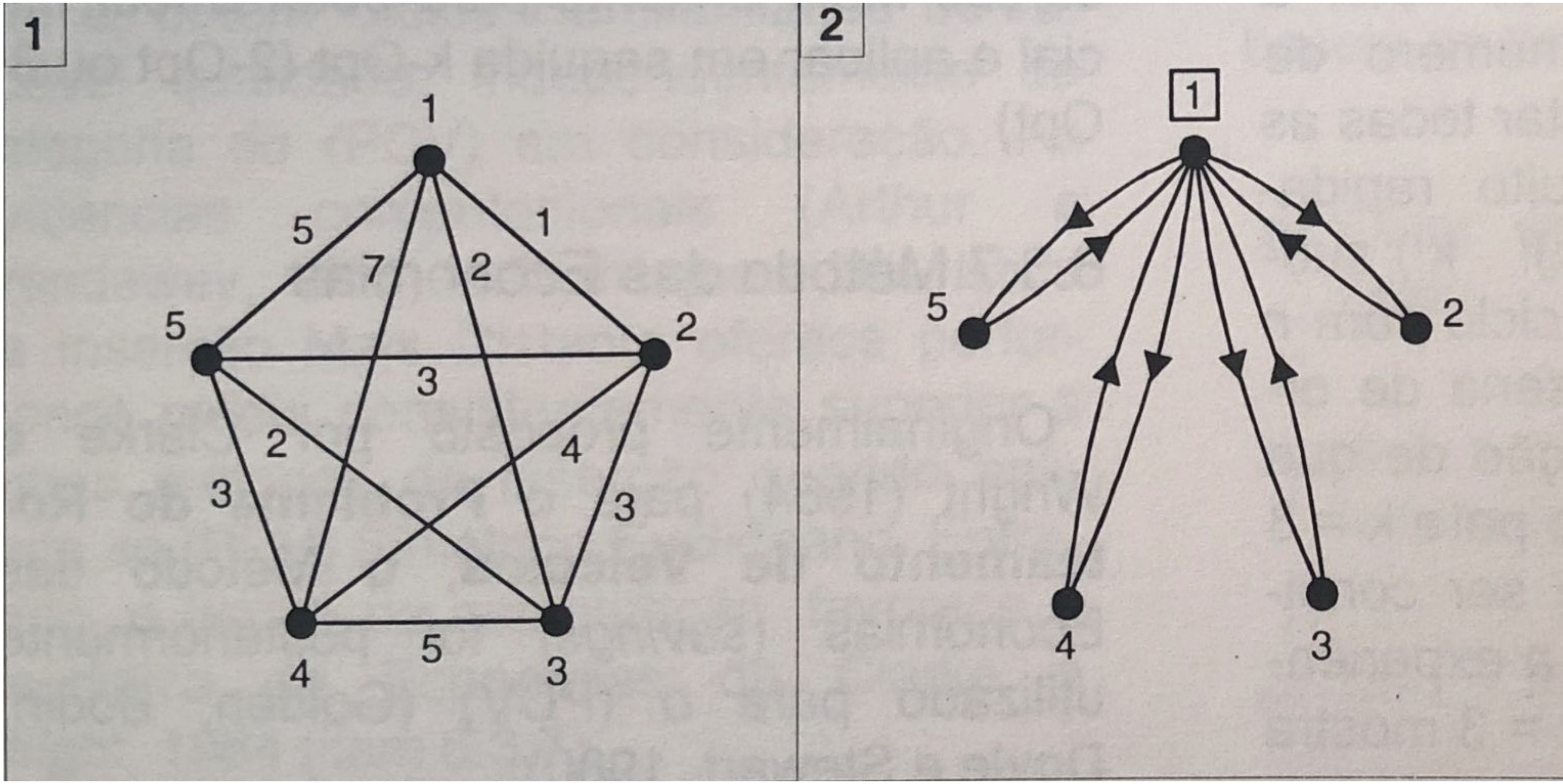
Algoritmo de economias

1. Selecionar qualquer vértice como referência (o vértice 1)
2. Obter $s_{ij} = c_{1i} + c_{j1} - c_{ij}$ e onde s_{ij} é a economia alcançada se o vértice i for ligado diretamente a j
3. Ordenar as economias em ordem decrescente
4. Percorrer sequencialmente a lista de economias e realizar as trocas que mantêm uma rota iniciando no vértice 1 e passando pelos demais nós, até obter um ciclo hamiltoniano

Algoritmo de economias

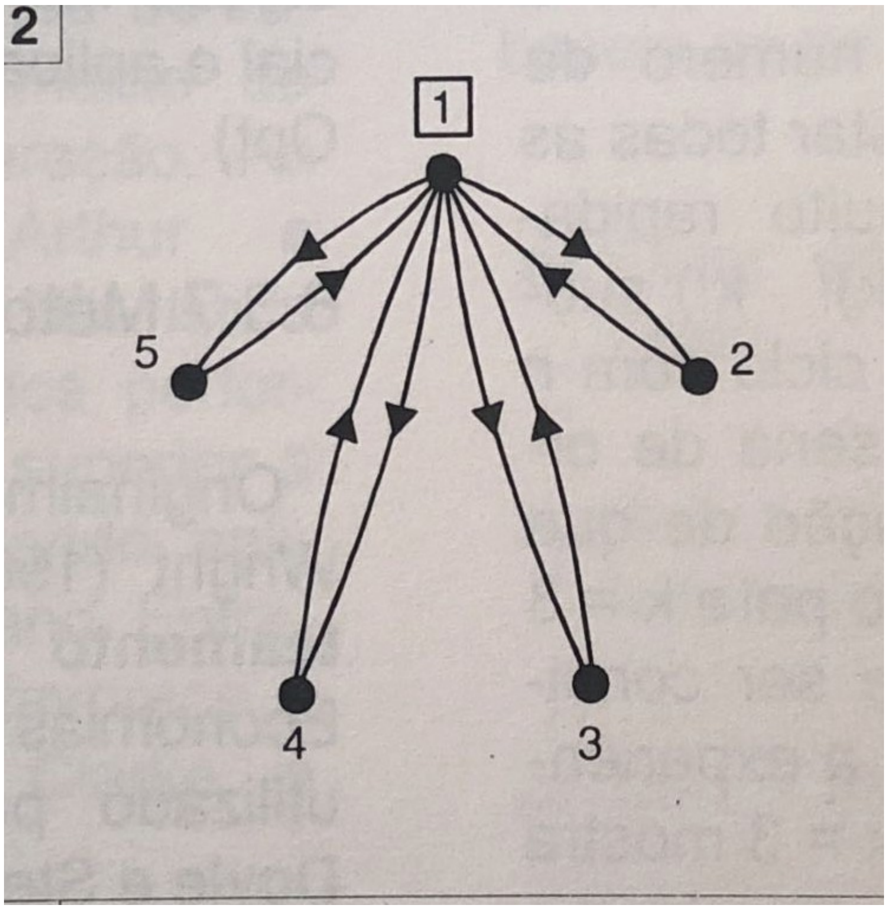
1. Percorrer sequencialmente a lista de economias e realizar as trocas que mantêm uma rota iniciando no vértice 1 e passando pelos demais nós, até obter um ciclo hamiltoniano (continuação)
   - tentar a ligação correspondente ao primeiro sij da lista
   - se a inserção da aresta (i,j) e a retirada das arestas (1,i) e (j,1) resultar em uma rota iniciando em 1 e passando pelos demais vértices, eliminar sij da lista
   - caso contrário, tentar a ligação seguinte na lista
   - repetir até obter o ciclo hamiltoniano

Algoritmo de economias



fonte: [1]

Algoritmo de economias



Lista de economias

$S_{45} = 5 + 7 - 3 = 9$

$S_{35} = 5 + 2 - 2 = 5$

$S_{34} = 7 + 2 - 5 = 4$

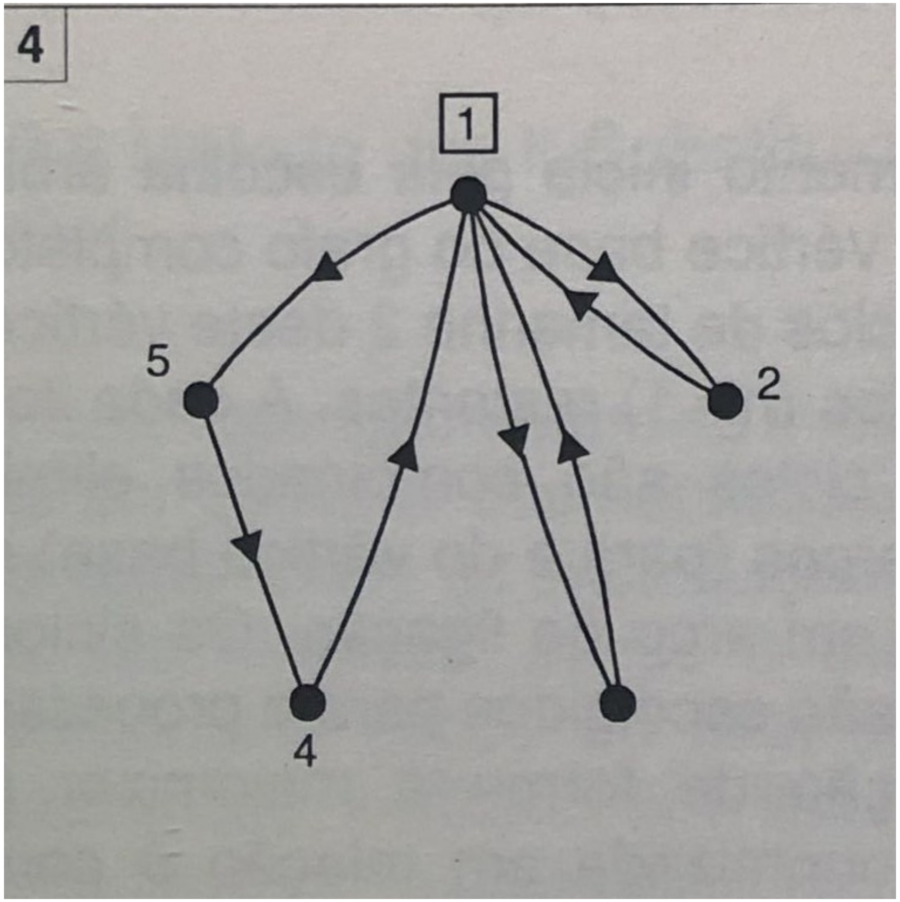
$S_{24} = 7 + 1 - 4 = 4$

$S_{25} = 5 + 1 - 3 = 3$

$S_{23} = 2 + 1 - 3 = 0$

fonte figura: [1]

Algoritmo de economias



Lista de economias

$S_{45} = 5 + 7 - 3 = 9$

$S_{35} = 5 + 2 - 2 = 5$

$S_{34} = 7 + 2 - 5 = 4$

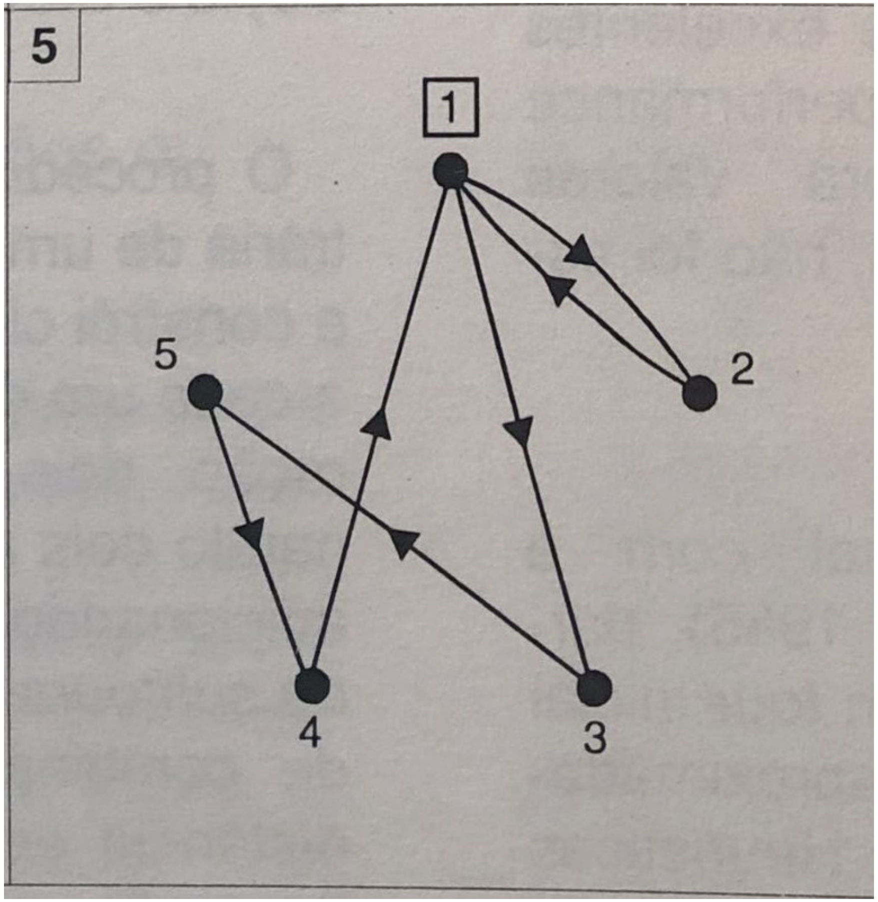
$S_{24} = 7 + 1 - 4 = 4$

$S_{25} = 5 + 1 - 3 = 3$

$S_{23} = 2 + 1 - 3 = 0$

fonte figura: [1]

Algoritmo de economias



Lista de economias

$S_{45} = 5 + 7 - 3 = 9$

$S_{35} = 5 + 2 - 2 = 5$

$S_{34} = 7 + 2 - 5 = 4$

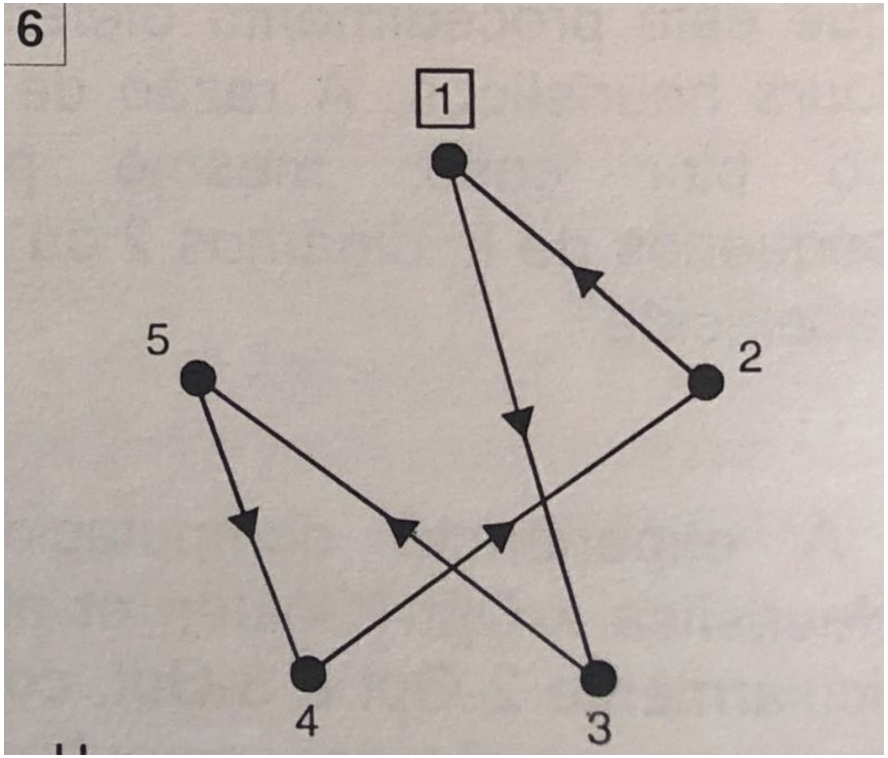
$S_{24} = 7 + 1 - 4 = 4$

$S_{25} = 5 + 1 - 3 = 3$

$S_{23} = 2 + 1 - 3 = 0$

fonte figura: [1]

Algoritmo de economias



Lista de economias

$S_{45} = 5 + 7 - 3 = 9$

$S_{35} = 5 + 2 - 2 = 5$

$S_{34} = 7 + 2 - 5 = 4$

$S_{24} = 7 + 1 - 4 = 4$

$S_{25} = 5 + 1 - 3 = 3$

$S_{23} = 2 + 1 - 3 = 0$

fonte figura: [1]

Algoritmo de economias

- O algoritmo de economias é dominado pelo processo de construção da lista ordenada de economias e tem complexidade de $O(n^2 \log n)$
- Se quisermos eliminar a influência da escolha do vértice base, repetimos o procedimento para cada um dos (n-1) vértices restantes como ponto de referência
- O algoritmo resultante passa a ter complexidade $O(n^3 \log n)$

Problema de otimização combinatória

- Dados

  um conjunto finito $E = \{1, 2, ..., n\}$,

  uma coleção de subconjuntos $F \in 2^E$

  e uma função de custo $c: 2^{|E|} \rightarrow R$ que associa um custo à cada solução,

  encontrar

  $S^* \in F$ tal que $c(S^*) \leq c(S)$ para todo $S \in F$

  onde $F \in 2^E$ é o conjunto de soluções viáveis do problema

- Tem-se um Problema Linear de Otimização Combinatória [1]

Problema de otimização combinatória

- Função de otimização
  - custos, comprimentos, quantidades
- Objetivo
  - minimização ou maximização
- Conjunto F discreto de soluções viáveis com um número finito de elementos
  - o conjunto de soluções geralmente é representado por subconjuntos de um conjunto $E = \{1, 2, ..., n\}$ de elementos que satisfazem determinadas condições

Métodos construtivos

- Construção de uma solução
  - selecionar sequencialmente elementos de E,
  - eventualmente descartando alguns já selecionados,
  - de tal forma que ao final se obtenha uma solução viável,
  - i.e. pertencente a F

Métodos construtivos

- Heurísticas gulosas
  - uma das técnicas de construção de algoritmos heurísticos mais explorada é tentar obter uma boa solução (eventualmente ótima), considerando a cada iteração a melhor decisão um passo à frente
    - ou seja utilizando um critério de otimização meramente local
  - por esta razão heurísticas deste tipo são chamadas míopes ou gulosas
  - Exemplos:
    - algoritmo guloso para determinação da árvore geradora mínima (solução ótima)
    - algoritmo vizinho mais próximo (PCV)

Métodos construtivos

- Heurísticas gulosas
  - selecionar sequencialmente o elemento de E que minimiza o incremento no custo da solução parcial,
  - eventualmente descartando alguns já selecionados, de tal forma que ao final se obtenha uma solução viável
  - o incremento no custo da solução parcial é chamado de função gulosa
  - por escolher a cada passo considerando apenas a próxima decisão, chama-se também de algoritmo míope, pois enxerga somente o que está mais próximo

Métodos construtivos

- Heurísticas gulosas

Algoritmo genérico

início

entrada: $E = \{1, 2,. ...,n\}$, c, F

ordenar os elementos de E de modo que

$$c(1) \leq c(2) \leq \ldots \leq c(n)$$

$S \leftarrow \varnothing$

para i de 1 a n faça

se $S \cup \{ i \} \in F$ então $S \leftarrow S \cup \{ i \}$

fim

Escalonamento de tarefas em processadores

- Escalonamento de tarefas em processadores paralelos idênticos
- Descrição da entrada
  - um conjunto de n tarefas independentes com durações $t_1, \ldots, t_n$ e m processadores idênticos
- Descrição do problema
  - distribuir as n tarefas pelos m processadores de forma a minimizar o tempo de término (makespan) da última tarefa

Escalonamento de tarefas em processadores

- Heurísticas para este problema adotam em geral a estratégia de organizar as tarefas em uma lista e no momento em que um dos processadores fica ocioso uma tarefa ainda não processada é retirada da lista e alocada a este processador
- A avaliação da qualidade da heurística é feita através da comparação com um limite inferior para a duração ótima

$$L = \lceil 1/m \, \Sigma_{i=1 \text{ a } n} (t_i) \rceil$$

Escalonamento de tarefas em processadores

- Longest processing time (LPT)
  - sempre que um processador ficar ocioso, alocamos a tarefa de maior duração ainda não processada
    - empates são resolvidos arbitrariamente
  - é possível implementar em $O(n \log n)$
    - tempo de execução é dominado pela etapa de ordenação das tarefas segundo durações não-crescentes

Escalonamento de tarefas em processadores

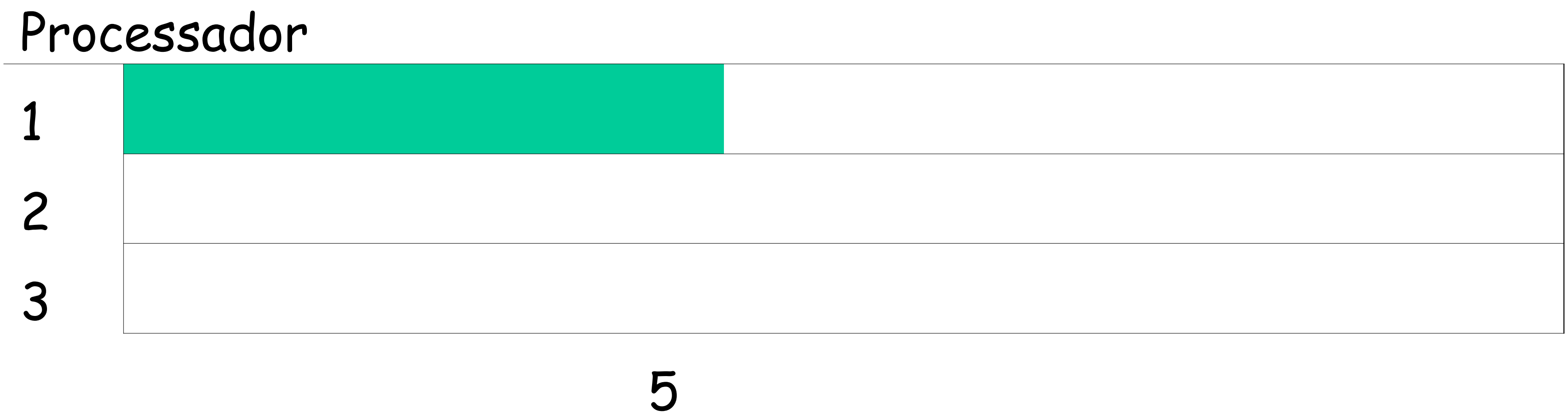
- Exemplo

$m = 3$

$n = 11$

$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$

$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$



Escalonamento de tarefas em processadores

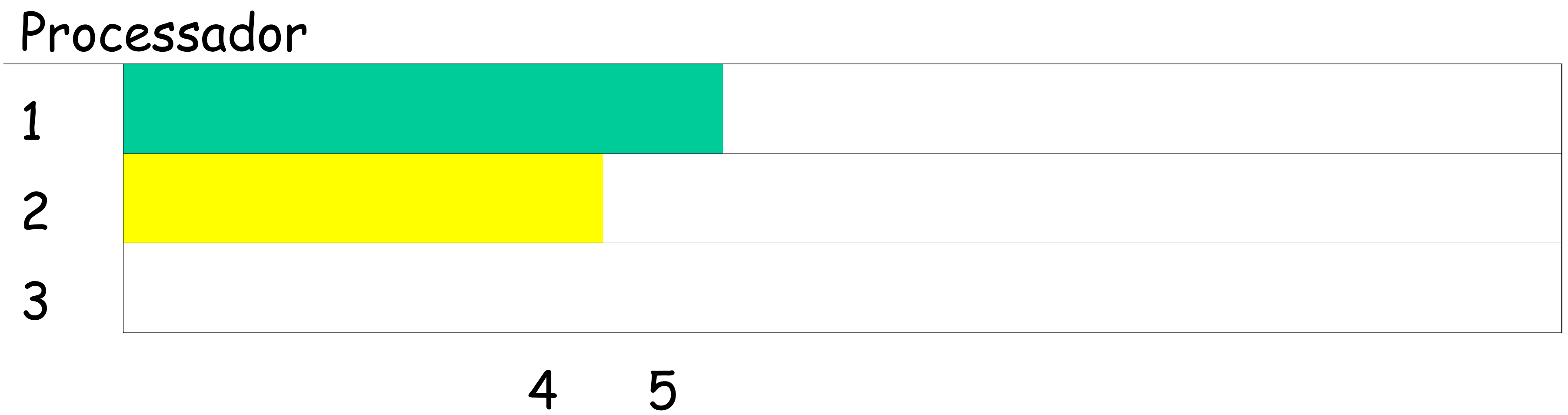
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Escalonamento de tarefas em processadores

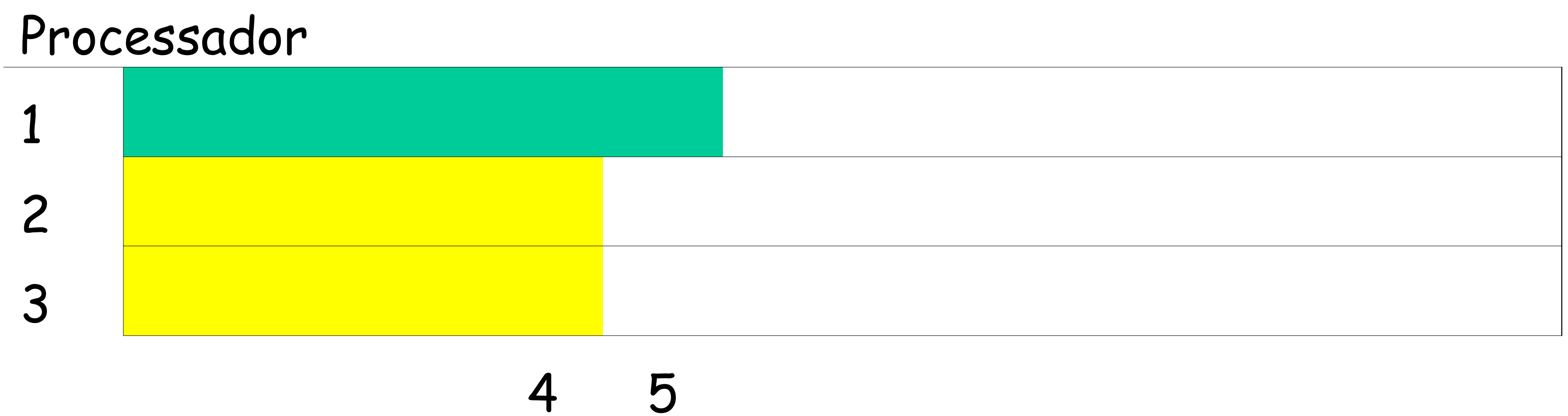
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Escalonamento de tarefas em processadores

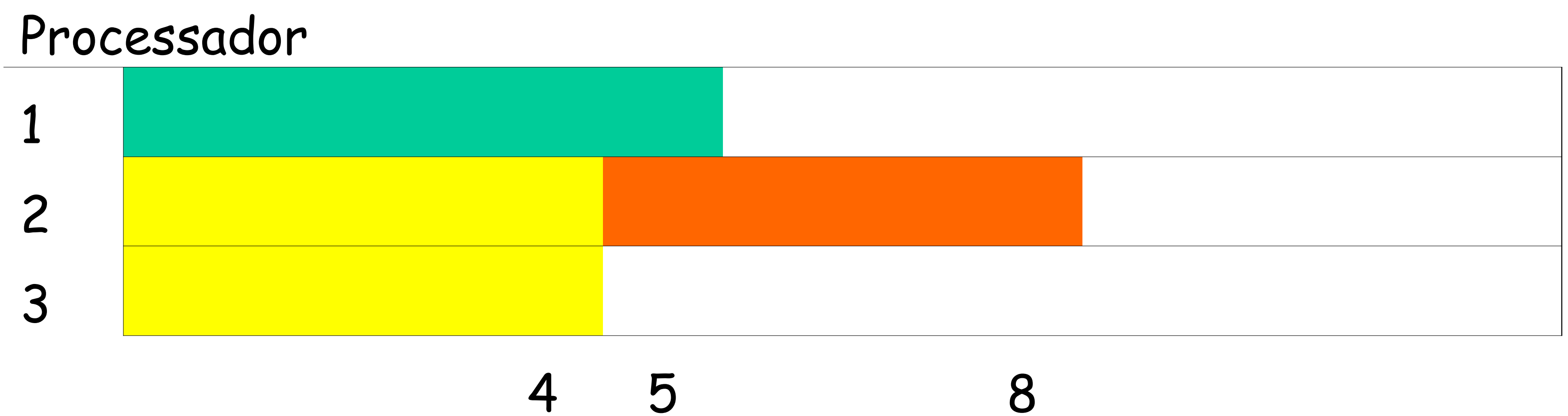
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Escalonamento de tarefas em processadores

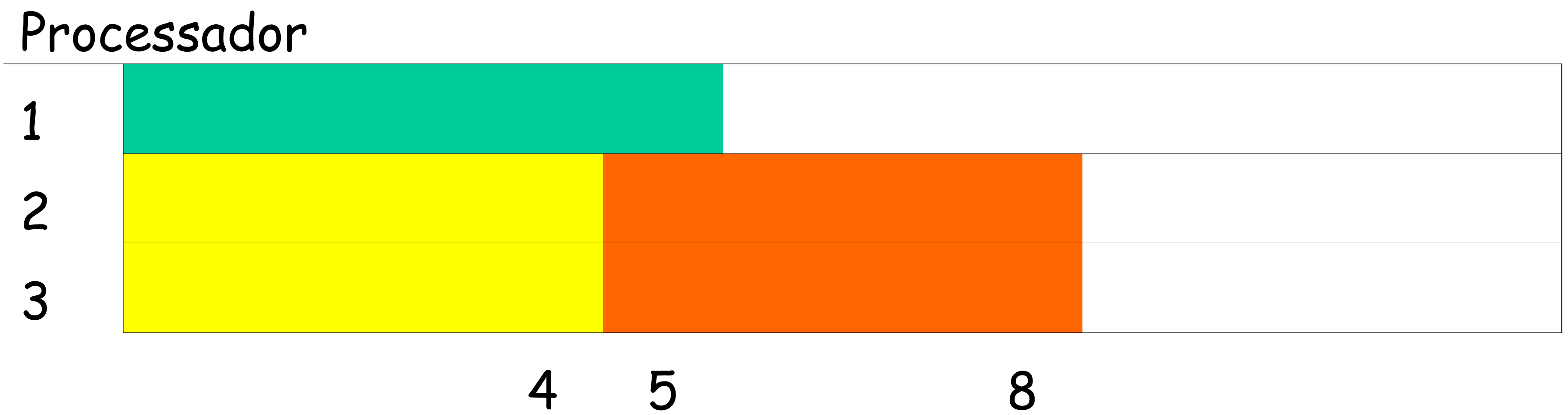
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Escalonamento de tarefas em processadores

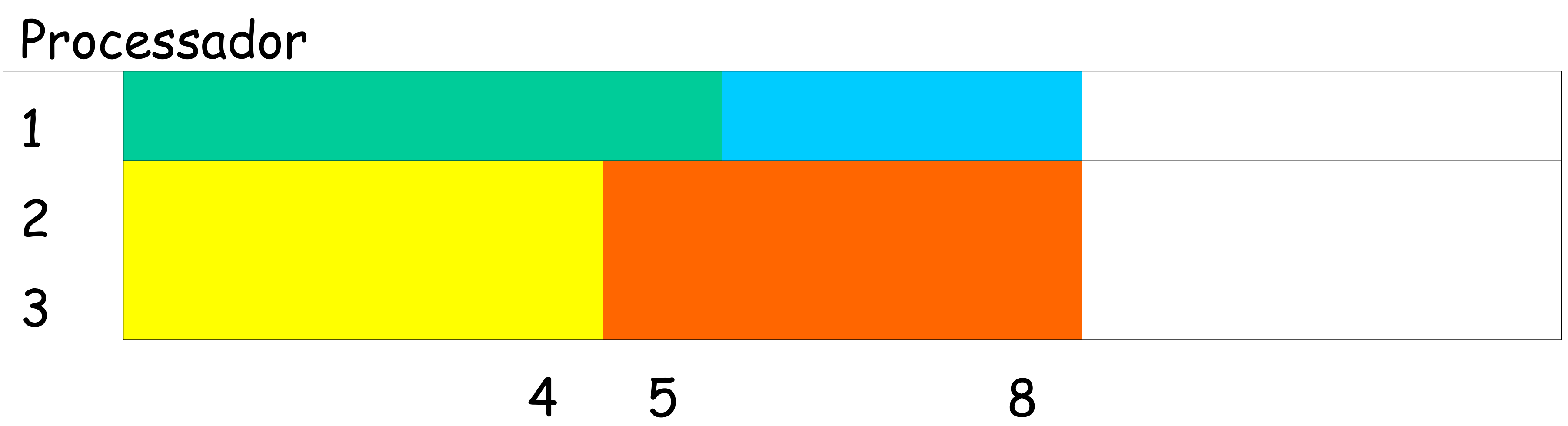
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Escalonamento de tarefas em processadores

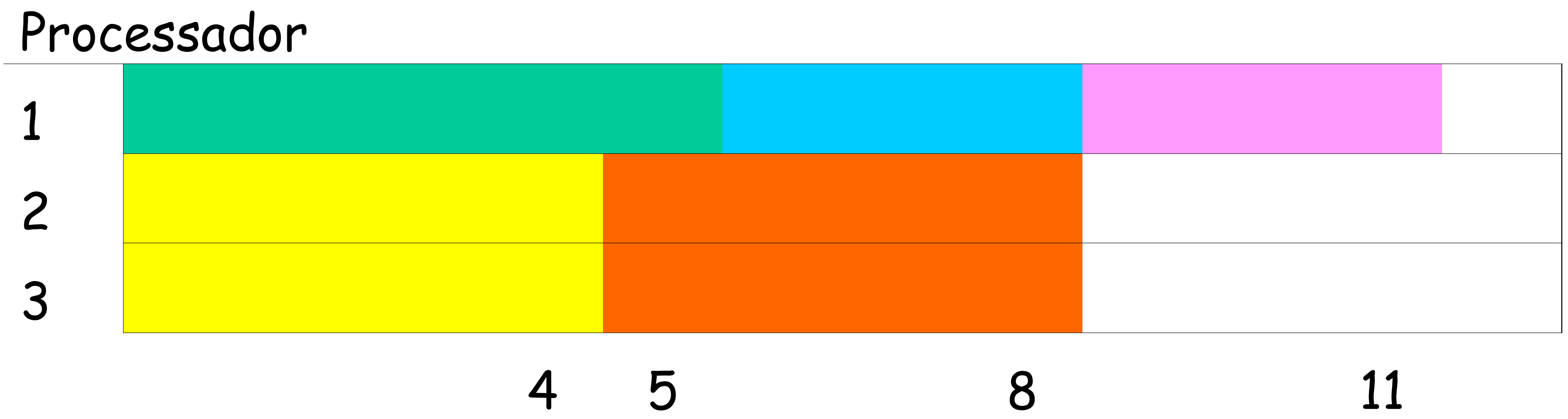
- Exemplo

$m = 3$

$n = 11$

$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$

$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$



Escalonamento de tarefas em processadores

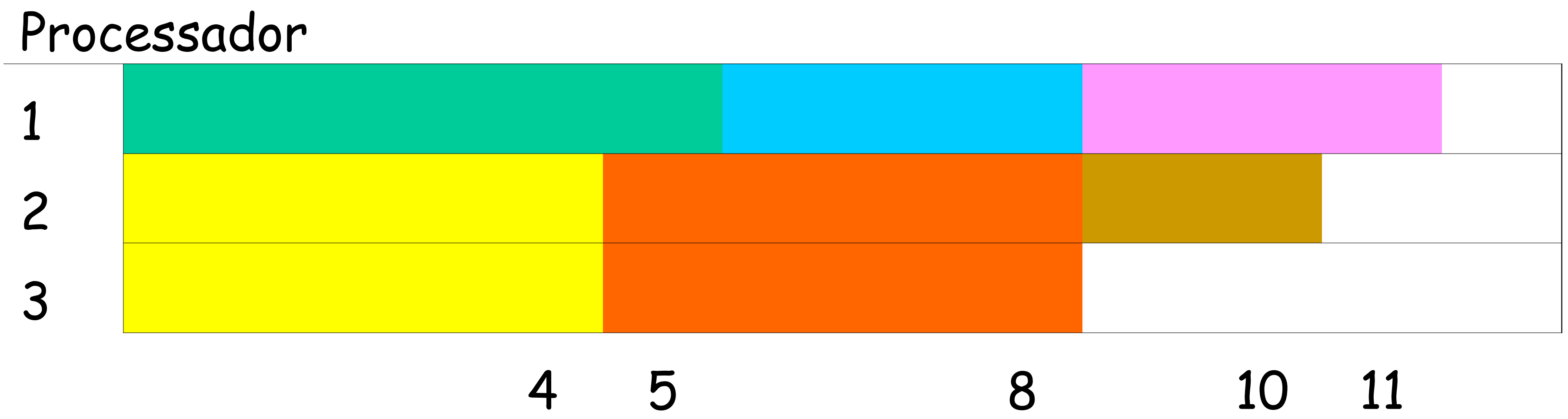
- Exemplo

$m = 3$

$n = 11$

$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$

$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$



Escalonamento de tarefas em processadores

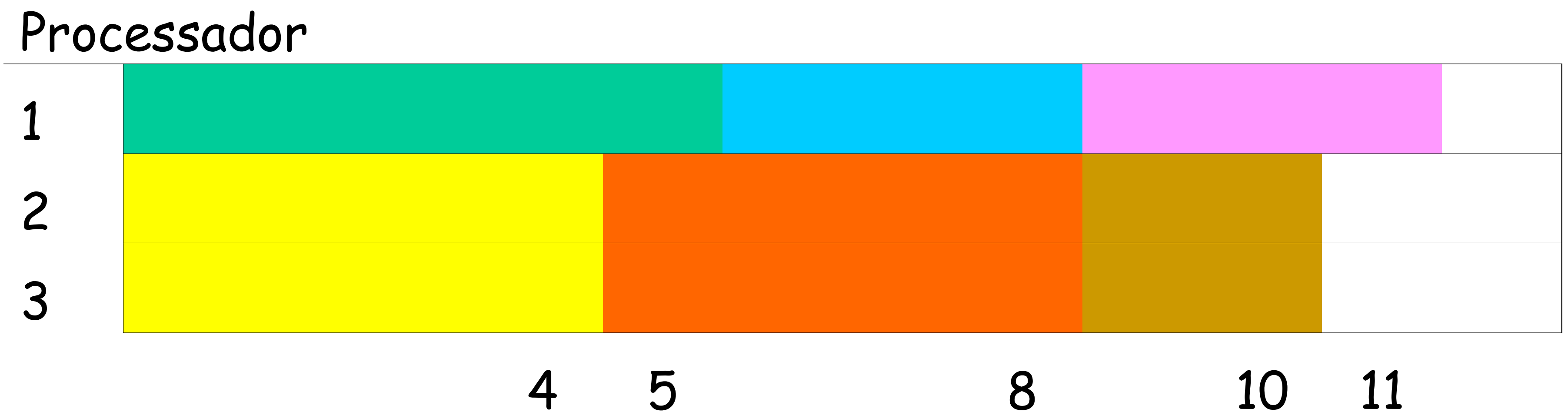
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Escalonamento de tarefas em processadores

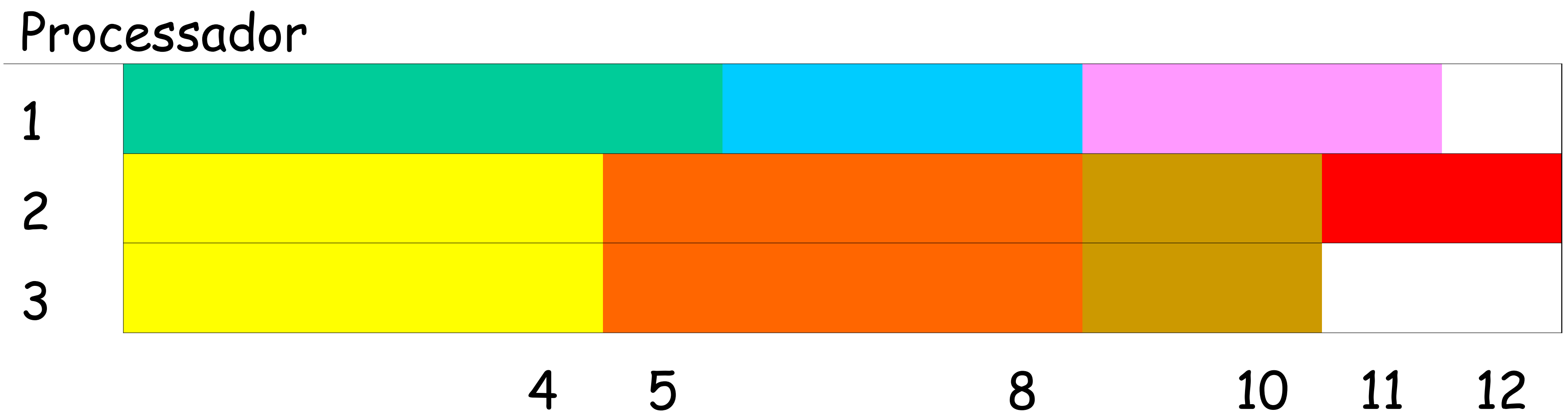
- Exemplo

$m = 3$

$n = 11$

$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$

$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$



Escalonamento de tarefas em processadores

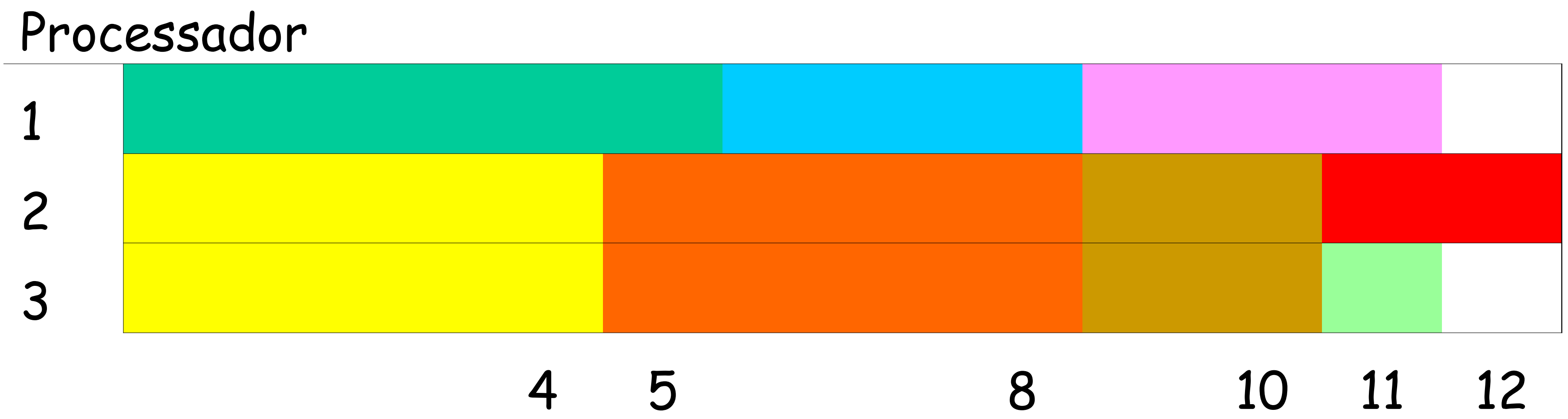
- Exemplo

$$m = 3$$

$$n = 11$$

$$(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}) =$$

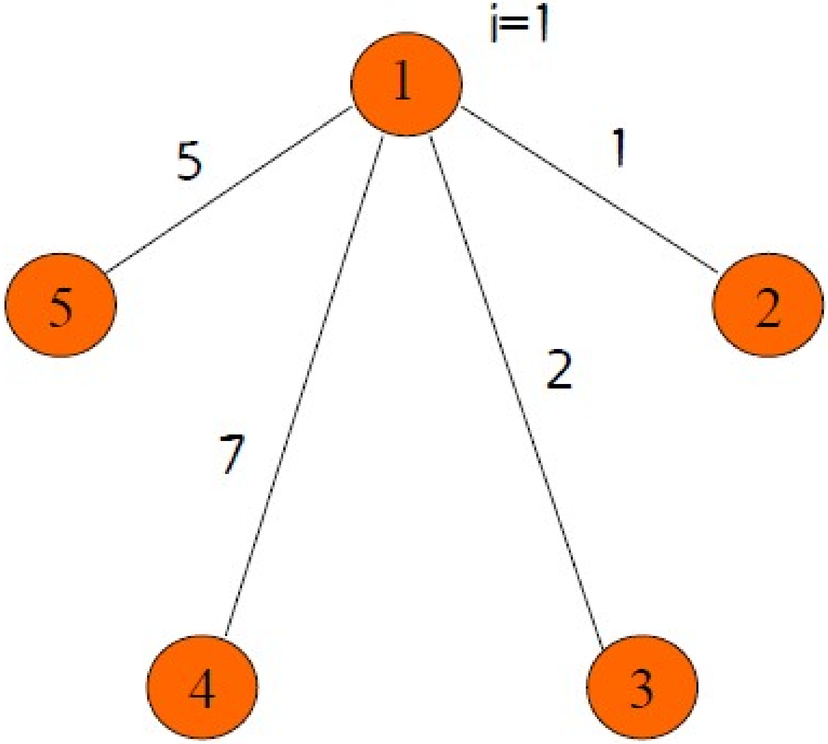
$$= (5, 4, 4, 4, 4, 3, 3, 2, 2, 2, 1)$$



Algoritmos gulosos randomizados

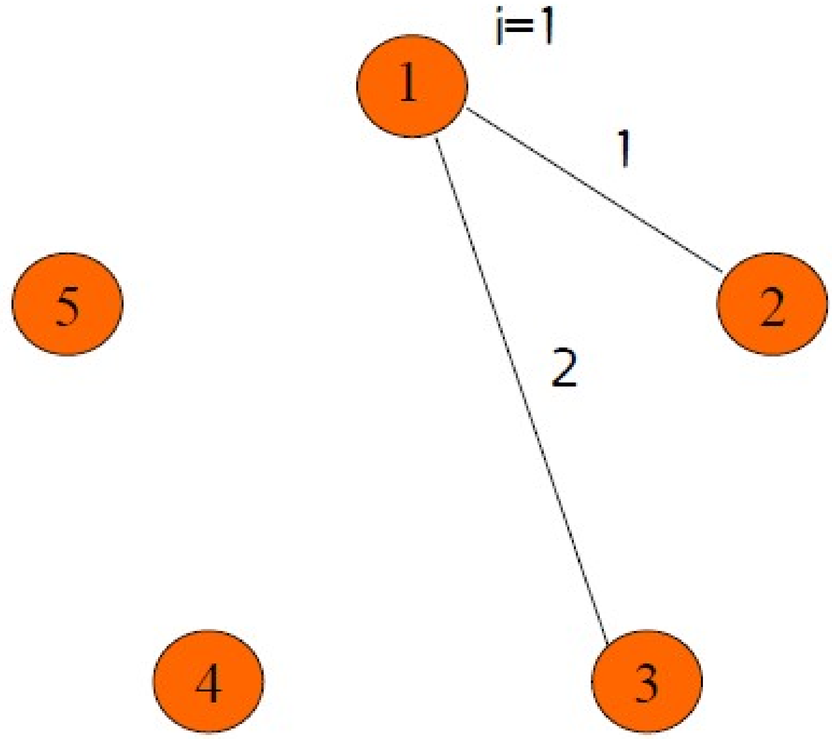
- Um algoritmo guloso encontra sempre a mesma solução para um dado problema.
- Randomização das escolhas gulosas permite alcançar diversidade nas soluções encontradas, se o algoritmo for aplicado diversas vezes.
- Algoritmo guloso randomizado:
  - Criar uma lista de candidatos a cada iteração com os 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

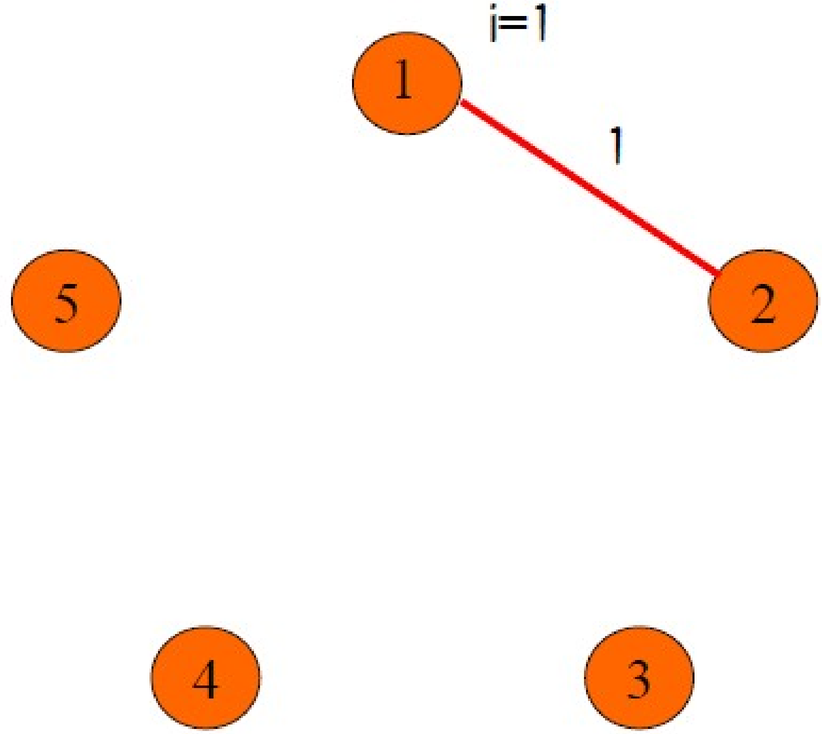


Laboratório de Otimização - UFES

Problema do caixeiro viajante



Problema do caixeiro viajante



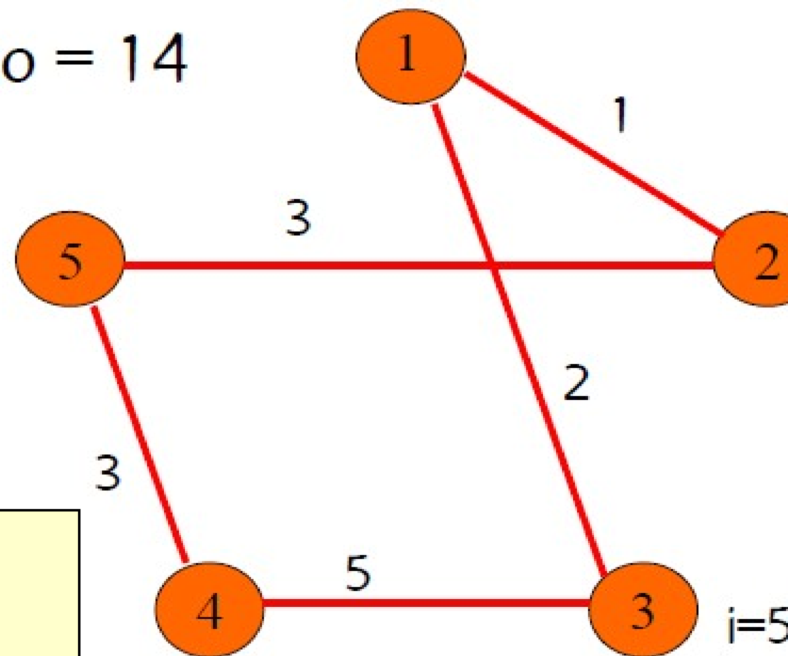
labtim
Laboratório de Otimização - UFES

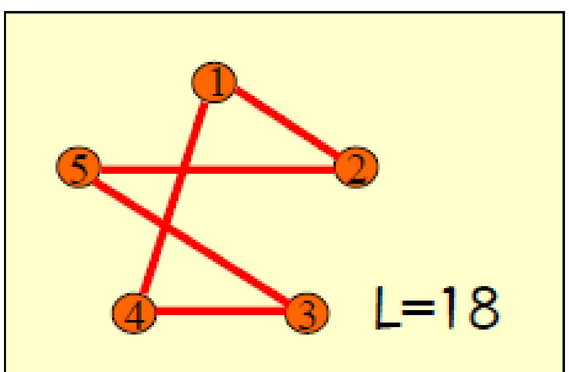
Problema do caixeiro viajante

E assim.....continuando em todos os nós.....

Problema do caixeiro viajante

comprimento = 14





Laboratório de Otimização - UFES

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

Referências

1. R.E. Campello e N. Maculan, Algoritmos e Heurísticas, Editora da Universidade Federal Fluminense, Niterói, 1994.
2. T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press & McGraw-Hill, 1991
3. J. Dréo, A. Pétrowski, P. Siarry and E. Taillard. "Metaheuristics for Hard Optimization: Methods and Case Studies"
4. H. H. Hoos e T. Stützle, 'Stochastic Local Search: Foundations e Applications' de (Morgan Kaufmann, 2004), www.sls-book.net
5. C. Ribeiro, Metaheurísticas, XI Escola Brasileira de Computação, 1998
    1. http://www-di.inf.puc-rio.br/~celso/grupo_de_pesquisa.htm
6. The Stony Brook Algorithm Repository.
    1. http://www.cs.sunysb.edu/\%7Ealgorith/
7. Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C , Pioneira Thomson Learning, 1993
    1. (http://www.dcc.ufmg.br/~nivio)