Heuristics and Local Search

Version 2

José Fernando Oliveira, Maria Antónia Carravilla – FEUP

Approximate methods to solve combinatorial optimization problems

Heuristics

Aim to efficiently generate very good solutions. They do not find the optimal solution, or at least do not guarantee the optimality of the found solutions.

Heuristics characteristics

- “Short” running times
- Easy to implement
- Flexible
- Simple

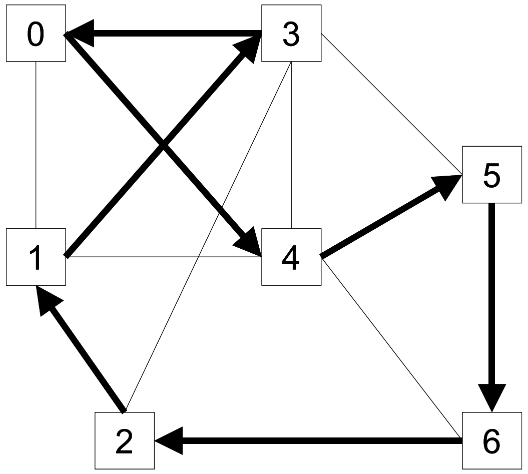
Types of heuristics

- **Constructive** – Build a solution, step by step, according to a set of rules defined before-hand.
- **Improvement** – Start form a feasible solution (any one) and improve it by applying successive small changes.
- **Compound** – First have a constructive phase and then an improvement phase.

These type of heuristics will be illustrated using the Traveling Salesperson Problem.

The Traveling Salesperson Problem (*TSP*)

The goal is to find the shortest path for a salesperson that leaves a city, visits n other cities and goes back to the initial city, without repeating any city.



The Traveling Salesperson Problem
Dantzig-Fulkerson-Johnson formulation

TSP formulation as a binary programming model over a graph $G = (V, A)$, where V is the set of vertices (cities) and A is the set of edges (direct paths between any two cities).

Indices

i city, $i \in \{1, \ldots, n\}$

j city, $j \in \{1, \ldots, n\}$

Coefficients

d_{ij} cost associated to the edge between city i and city j.

Decision variables

$$x_{ij} = \begin{cases} 1 & \text{if the edge connecting } i \text{ to } j \\ & \quad \text{belongs to the solution} \\ 0 & \text{it not} \end{cases}$$

The Traveling Salesperson Problem
Dantzig-Fulkerson-Johnson formulation (cont.)

Objective function

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

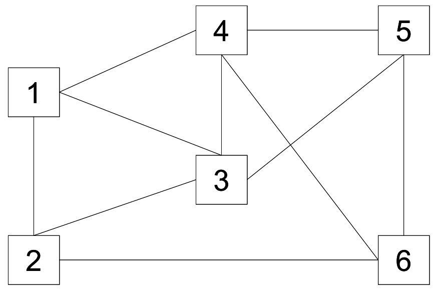
Constraints

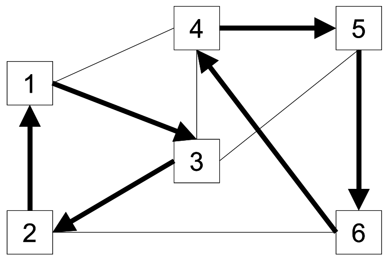
$$\begin{array}{rrcll}
\sum_{i=1}^{n} & x_{ij} & = & 1 & \forall_{j \in V} \\
\sum_{j=1}^{n} & x_{ij} & = & 1 & \forall_{i \in V} \\
\sum_{i,j \in S} & x_{ij} & \leq & |S| - 1 & \forall_{S \subset V} \\
 & x_{ij} & \in & \{0,1\} & \forall_{i,j \in V, i \neq j}
\end{array}$$

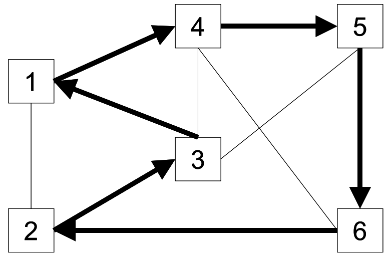
$|S|$ stands for the number of vertices of subgraph S.

Note that $S \equiv V$ is not included in $S \subset V$.

The Traveling Salesperson Problem
Dantzig-Fulkerson-Johnson formulation
Sub-cycles elimination







$$S = \{1,3,4\}$$

$$x_{13} = 1 \leq |S| - 1 = 2$$

$$S = \{4,5,6\}$$

$$x_{45} + x_{56} + x_{64} = 3 \not\leq |S| - 1 = 2$$

$$S = \{1,2,3\}$$

$$x_{23} + x_{31} = 2 \leq |S| - 1 = 2$$

$$S = \{4,5,6\}$$

$$x_{45} + x_{56} = 2 \leq |S| - 1 = 2$$

$$etc\ldots$$

Constructive heuristics

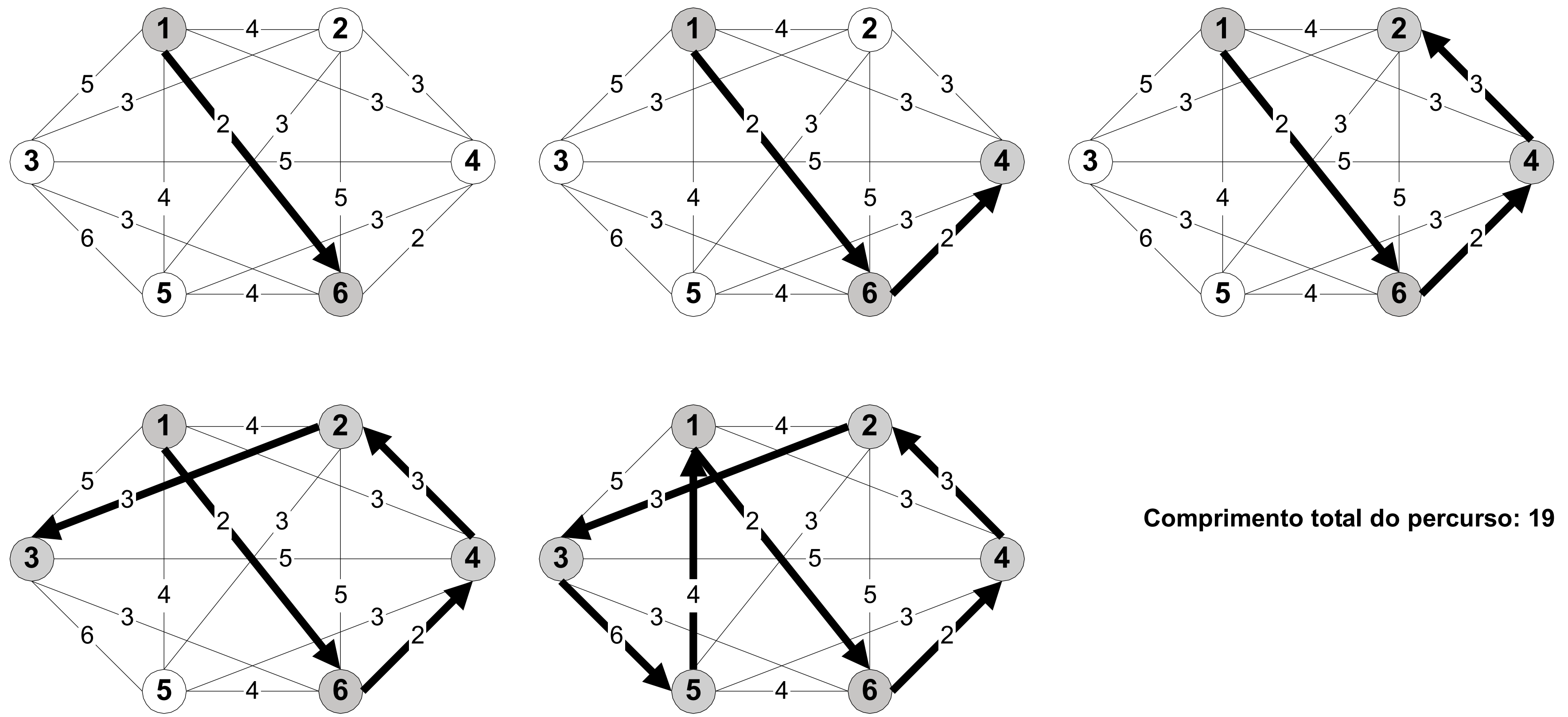
Build a solution, step by step, according to a set of rules defined before-hand. These rules concern:

- the choice of the initial sub-cycle (or starting point) – *initialization*;
- a criterion to choose the next element to add to the solution – *selection*;
- the selection of the position where the new element will be inserted – *insertion*.

TSP – Nearest neighbor

1. *Initialization* – Start with a partial tour with just one city i, randomly chosen;
2. *Selection* – Let $(1, \ldots, k)$ be the current partial tour $(k < n)$. Find city $k + 1$ that is not yet in the tour and that is closer to k.
3. *Insertion* – Insert $k + 1$ at the end of the partial tour.
4. If all cities are inserted then STOP, else go back to 2.

Nearest neighbor – exemple



Total length of the tour: 19

TSP – Nearest insertion of arbitrary city

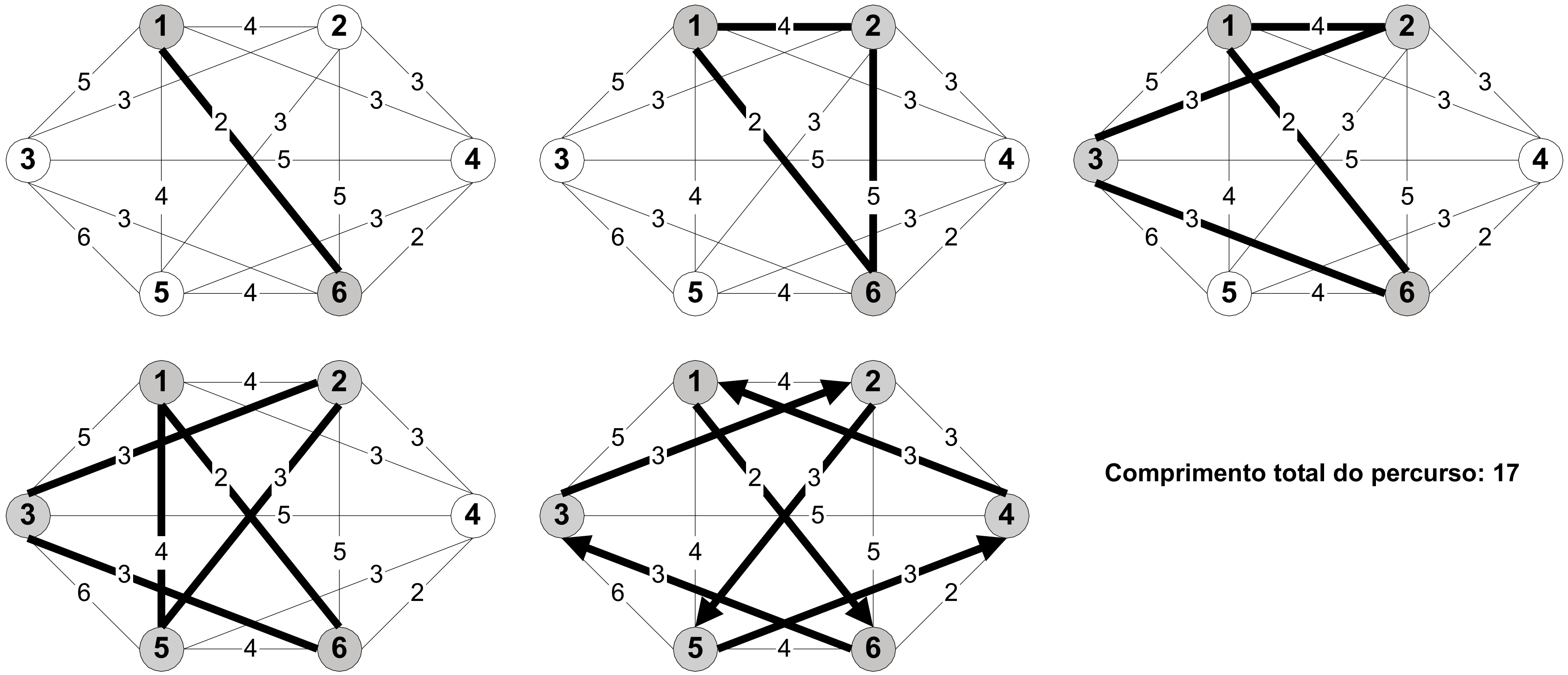
1. *Initialization* – Start with a partial tour with just one city i, randomly chosen;
   find the city j for which c_{ij} (distance or cost from i to j) is minimum and build the partial tour (i, j).

2. *Selection* – Given a partial tour, arbitrary select a city k that is not yet in the partial tour.

3. *Insertion* – Find the edge $\{i, j\}$, belonging to the partial tour, that minimizes $c_{ik} + c_{kj} - cij$. Insert k between i and j.

4. If all cities are inserted then STOP, else go back to 2.

Nearest insertion of arbitrary city – example



Total length of the tour: 17

TSP – Nearest insertion

1. *Initialization* – Start with a partial tour with just one city i, randomly chosen;
   find the city j for which c_{ij} (distance or cost from i to j) is minimum and build the partial tour (i, j).

2. *Selection* – Find cities k and j (j belonging to the partial tour and k not belonging) for which c_{kj} is minimized.

3. *Insertion* – Find the edge $\{i, j\}$, belonging to the partial tour, that minimizes $c_{ik} + c_{kj} - cij$. Insert k between i and j.

4. If all cities are inserted then STOP, else go back to 2.

This heuristic as a variant named “Farthest Insertion” that replaces the selection step by:

2. *Selection* – Find cities k and j (j belonging to the partial tour and k not belonging) for which $\min_{kj}\{c_{kj}\}$ is maximized.

TSP – Cheapest insertion

1. *Initialization* – Start with a partial tour with just one city i, randomly chosen.

2. *Selection* – Find cities k, i and j (i and j being the extremes of an edge belonging to the partial tour and k not belonging to that tour) for which $c_{ik} + c_{kj} - cij$ is minimized.

3. *Insertion* – Insert k between i and j.

4. If all cities are inserted then STOP, else go back to 2.

TSP – Convex hull[a]

1. *Initialization* – Start with a partial tour formed by the convex hull of all cities.
2. *Selection* – For each city not yet inserted in the partial tour, find the edge $\{i, j\}$, belonging to the partial tour, that minimizes $c_{ik} + c_{kj} - cij$. From all triplets $\{i, j, k\}$ evaluated in step 2, find the triplet $\{i^\star, j^\star, k^\star\}$ for which $\frac{c_{i^\star k^\star} + c_{k^\star j^\star}}{c_{i^\star j^\star}}$ is minimum.
3. *Insertion* – Insert $k^\star$ between $i^\star$ and $j^\star$.
4. If all cities are inserted then STOP, else go back to 2.

[a]Convex hull of a set A – convex shape that includes in its interior or frontier all the elements of set A

TSP – Nearest merger

1. *Initialization* – Start with n partial tours formed, each one, by just one city i.
2. *Selection* – Find two cities i and k (i belonging to a partial tour C and k belonging to another partial tour C') for which c_{ik} is minimized.
3. *Insertion* – Let i, j, k and l be cities so that $\{i, j\} \in C$, $\{k, l\} \in C'$ and $c_{ik} + c_{jl} - c_{ij} - c_{kl}$ is minimized.
   Insert $\{i, k\}\}$ e $\{j, l\}\}$ and delete $\{i, j\}\}$ e $\{k, l\}\}$.
4. If all cities are inserted then STOP, else go back to 2.

The minimum spanning tree problem

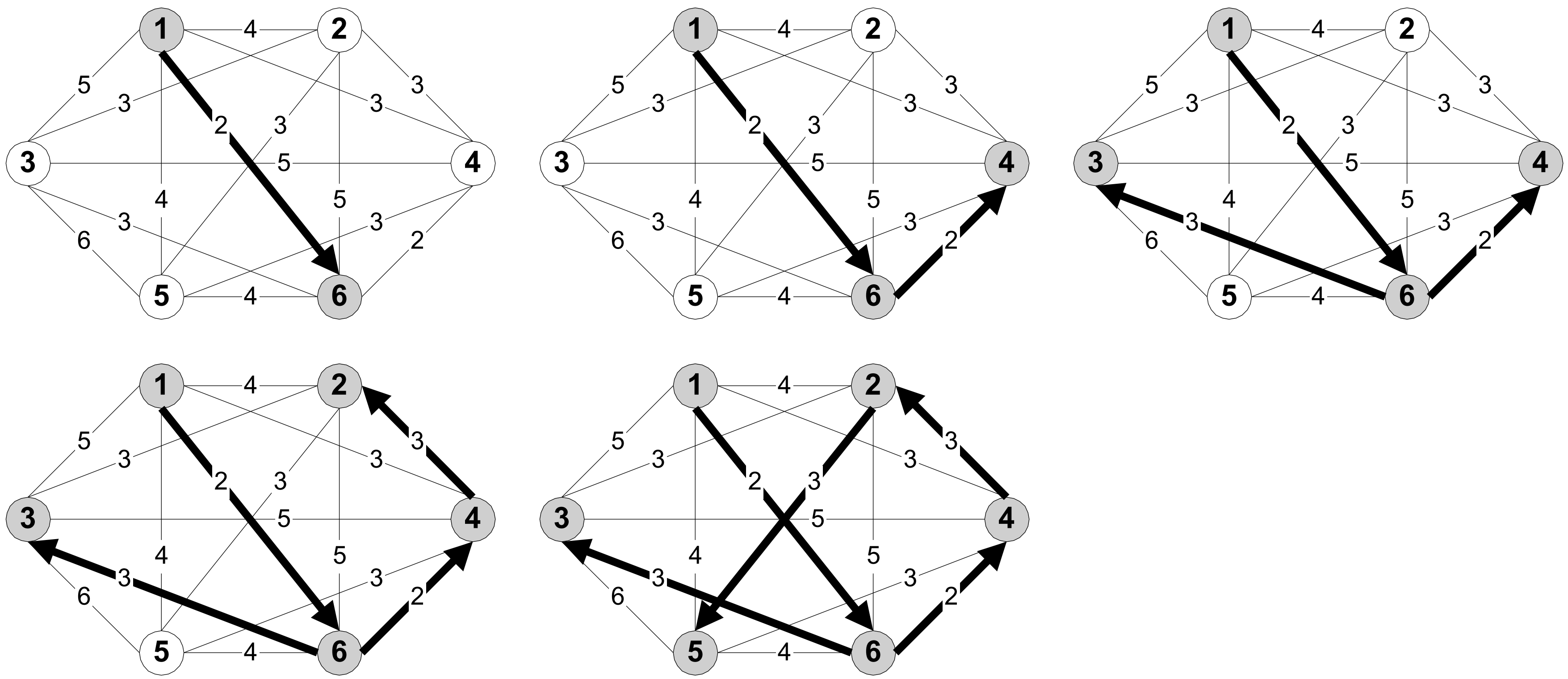
- Definitions (for non-oriented graphs):
  - a tree is an acyclic connected graph;
  - a graph is connected if there is a path (sequence of edges) connecting any pair of vertices.
- Problem:

  Find the tree of minimum total length (or cost) that supports all nodes of the graph (i.e. that connects all nodes).
- Applications:
  - communication networks;
  - power system networks.
  - ......

Prim's algorithm (greedy procedure)

1. Select a node randomly and connect it to the nearest node;
2. Find the node that is nearest to a node already inserted in the tree, among those not yet inserted, and connect those two nodes;
3. If all nodes are already inserted then STOP, else go back to 2.

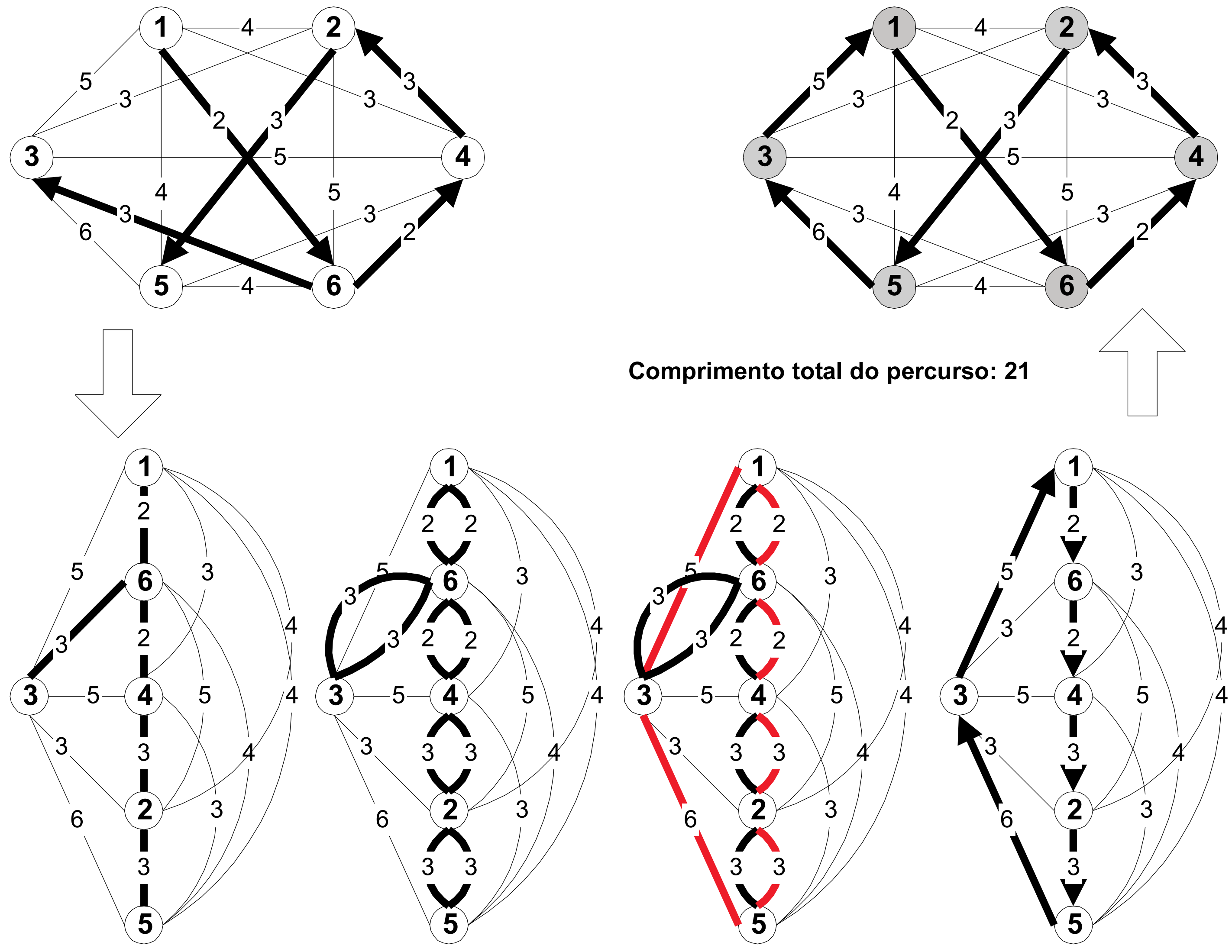
greedy procedure – Example



TSP – Minimum spanning tree

1. Build the minimum spanning tree that connects all cities.
2. Make a depth-first visit to the tree.
3. Insert shortcuts (replacing sequences of 2 ou more edges by just one edge) in the path generated by the depth-first visit, so that a tour is generated.

Minimum spanning tree – example



Total length of the tour: 21

Improvement heuristics

$\boxed{\text{Start from } \textbf{any feasible solution} \text{ and improve it by successive small changes.}}$ $\longrightarrow$ how to generate one?

$\downarrow$

(1) randomly;

(2) constructive heuristic.

In the second case we are designing a compound heuristic, in which the improvement algorithm is the second phase of the overall compound heuristic.

TSP – Heuristic *r-opt*[a]

1. Generate an initial *complete* feasible tour $\rightarrow C^0$.
   Make the current tour $C^k = C^0$.

2. Remove r edges from current tour C^k, making it uncomplete $\rightarrow C_i^k$.

3. Build all feasible solutions (complete tours) that include C_i^k (the uncomplete tour).

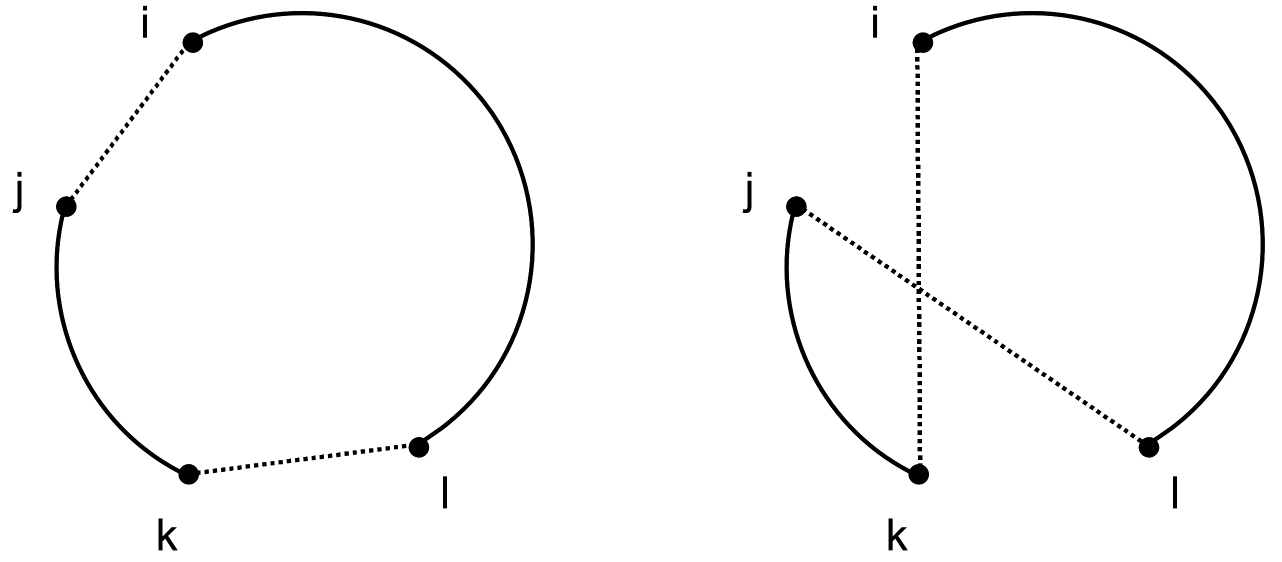
4. Select the best tour among these tours $\rightarrow C^\star$.

5. if $\textbf{length}(C^\star) < \textbf{length}(C^k)$ then $C^k = C^\star$ and go back to 2. Else STOP.

[a]or r-exchange

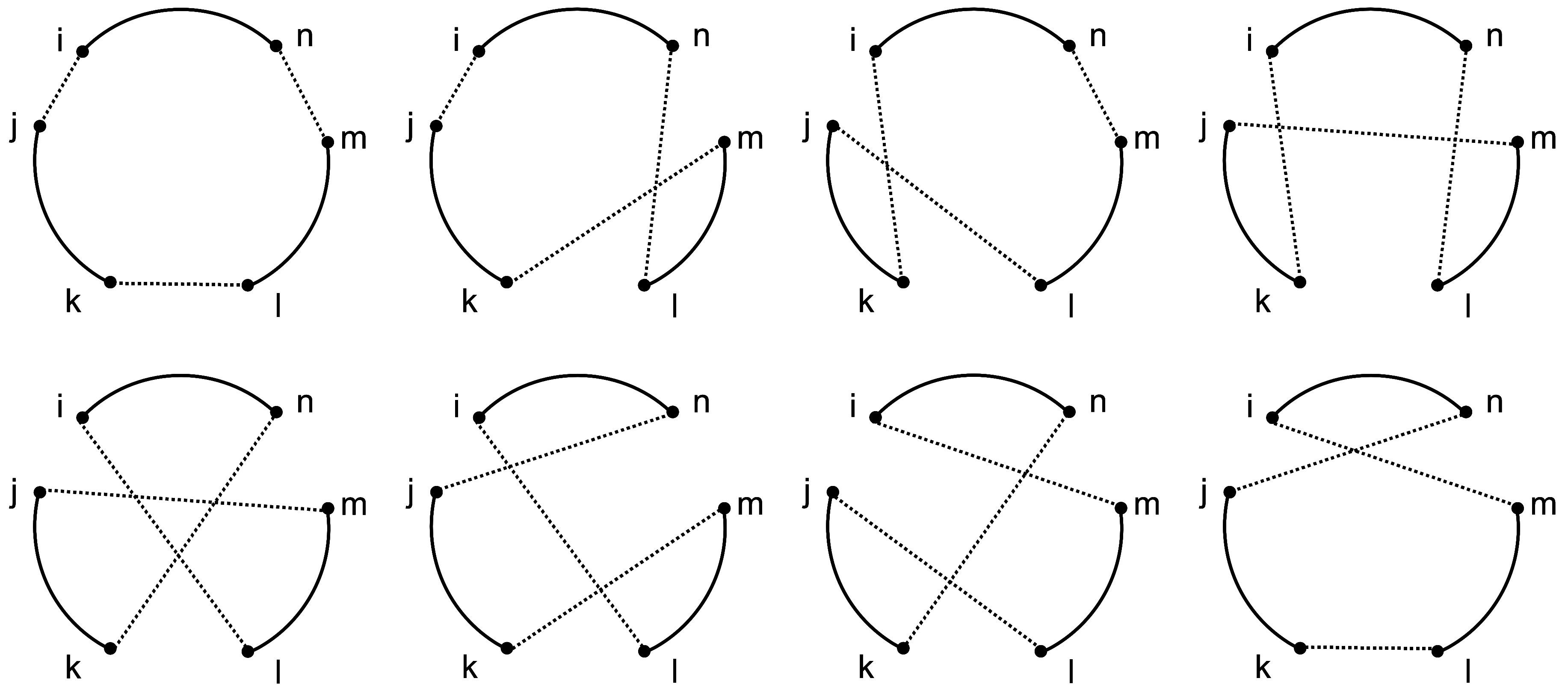
Algorithm 2-opt

In a 2-opt algorithm, when removing 2 edges there is only one alternative feasible solution:



Algorithm 3-opt

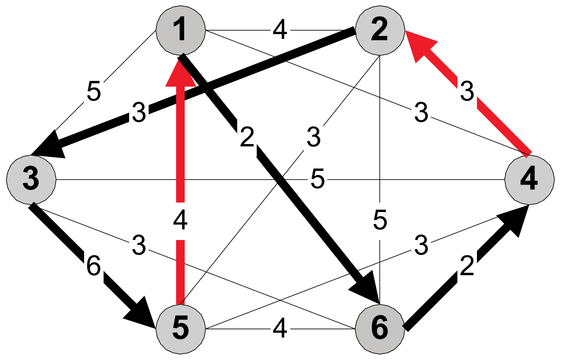
In a 3-opt algorithm, when removing 3 edges there are $2^3 - 1$ alternative feasible solutions:



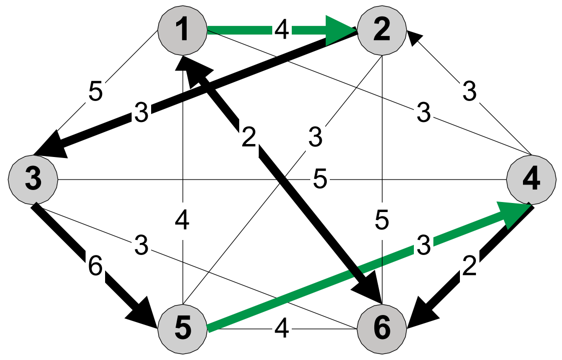
Algorithm 2-opt – example

Two examples of 2 edges exchange, one leading to a solution of equal value and other leading to a solution with a smaller value.

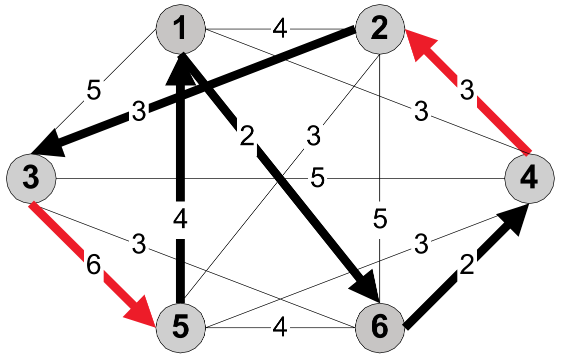
Comprimento total do percurso: 20



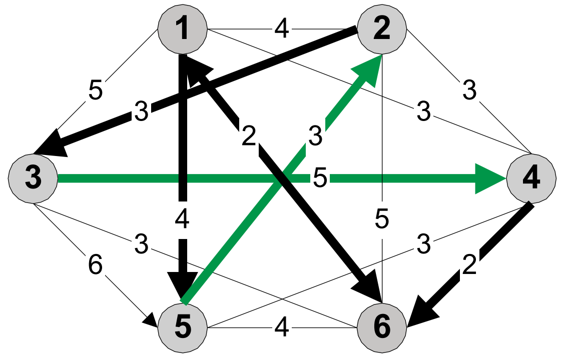
Comprimento total do percurso: 20



Comprimento total do percurso: 20



Comprimento total do percurso: 19



Solução melhor

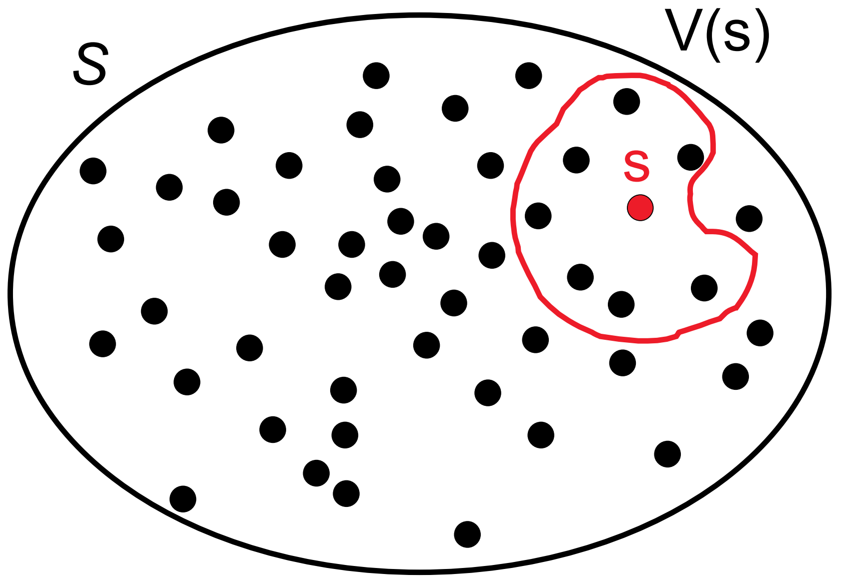
The algorithm would follow form this better solution until some stop criterion was reached (e.g. maximum number of exchanges, number of exchanges without improvement, etc.)

Local search and neighborhoods

Local search is based on the oldest optimization method: trial and error. But in a systematic way...

To systematize the search a **neighborhood structure** is defined and build.

The neighborhood of a given solution is the set of feasible solutions that, somehow, are alike the given solution, i.e. com similar elements and objective function values not very different.



Example: In the TSP it is possible to define as a neighborhood of a given tour all the tours that can be generated form that one by applying a 2-opt iteration.

Local search and neighborhoods

Generic local search algorithm:

1. Generate an initial solution $\rightarrow s_0$.
2. Current solution $s_i = s_0$.
3. Pick $s_j \in V(s_i)$.
4. If $f(s_j) < f(s_i)$, then $s_i = s_j$.
5. Else, $V(s_i) = V(s_i) - s_j$.
6. If $V(s_i) \neq \emptyset$, then go to 3.
7. Else, END.
   Local optimal solution $= s_i$.

We say that a "movement" has happened each time a new solution is accepted as current solution (also called "neighborhood center") – step 4.

Local search algorithm for the TSP, based on 2-opt movements

1. Build an initial tour.
2. Select randomly an edge from that tour.
3. Make a 2-opt movement with all the other edges of the tour and select the best tour therefore generated.
4. If it is better than the current tour then make it the current tour and go to 2.
5. Else, STOP. The local optimum was reached.

Different neighborhood structures originate different local search algorithms.

Local search and neighborhoods – conclusion

To devise a good neighborhood structure for a combinatorial optimization problem and build a search method is a science and an art $\rightarrow$ noble research work.

Bibliography

- Stephan Mertens. *TSP Algorithms in Action. Animated Examples of Heuristic Algorithms*, `http://www-e.uni-magdeburg.de/mertens/TSP/index.html`
- David S. Johnson, Lyle A. McGeoch (1995). *The Traveling Salesman Problem: A Case Study in Local Optimization*, `http://www.research.att.com/~dsj/papers/TSPchapter.pdf`.
- Goldbarg, Marco Cesar e Luna, Henrique Pacca (2000). *Otimização Combinatória e Programação Linear*, Editora CAMPUS.
- Golden, B.L. and Stewart, W.R. (1985). *Empirical analysis of heuristics* in THE TRAVELING SALESMAN PROBLEM, John Wiley & Sons, Inc..
- Sousa, Jorge Pinho (1991). *Apontamentos de Optimização Combinatória.*