Introduction to heuristics

Daniele Vigo

DEI – University of Bologna

daniele.vigo@unibo.it

Heuristic Algorithms

- Determine “good" feasible solutions for NP-Hard (=difficult) problems within “reasonable" computing time.
- If the “Feasibility" Problem is NP-Hard:
  - try to determine a feasible solution (possibly a “good" one).
- Exact algorithms: determine the “optimal” solution (but may require a very large computing time)

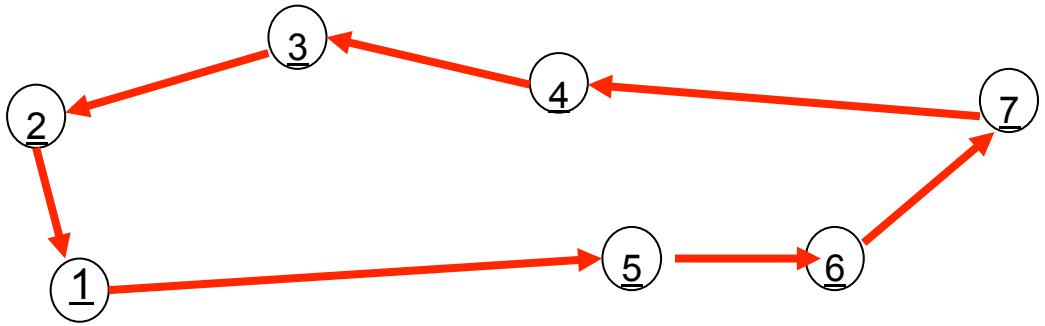
Performance of heuristics

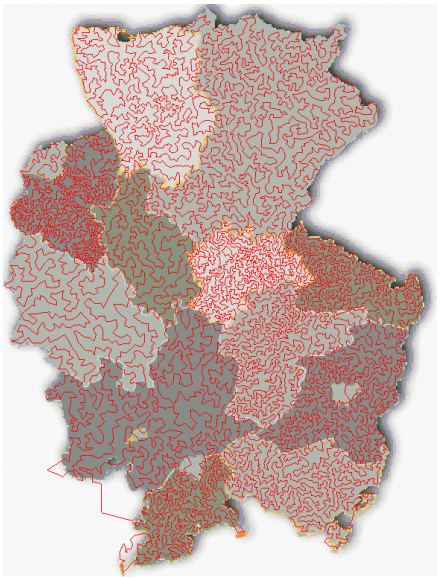
- Criteria for the evaluation of the “practical" performance of a heuristic algorithm (minimization problem):
  - “quality" of the heuristic solution value: \gap" with respect to the optimal solution value (or to a good Lower Bound): average value, maximum value, percentage of optimal solutions found,
  - “computing time" (average value, maximum value): much smaller than the computing time of the corresponding exact algorithms.

Example 1: TSP

- Traveling Salesman Problem (TSP)

  Given a weighted graph determine the Hamiltonian circuit with minimum cost (closed cycle that visits once all nodes of the graph)





Example 2: Knapsack

- Knapsack Problem 0-1 (KP-01):

  Given a set of n items, characterized by a profit p_j and a weight w_j (j = 1…n) and a container (knapsack) having capacity c (c >= w_j) choose a subset of items with maximum total profit and that fits into the knapsack

ILP model

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$\max \quad \sum_{j=1,n} p_j x_j$$

$$\sum_{j=1,n} w_j x_j \leq W$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, n)$$

or

$$0 \leq x_j \leq 1 \text{ integer} \quad (j = 1, \ldots, n)$$

Main types of heuristics

- There are two classical families of heuristics
  - Construction heuristics
  - Local Search algorithms
- Recent advanced methods: Meta-heuristics
  - Tabu search
  - Genetic algorithms / evolutionary methods
  - Simulated annealing
  - Ant systems …

2.1
Construction heuristics

Construction heuristics

- Iterative algorithms: Iteratively determine the new elements to be inserted or removed into the solution until a complete solution is found
- Primal Algorithms:
  - start from a *feasible* solution, at each iteration try to “improve" the value of the objective function of the current solution by keeping its feasibility.
- Dual Algorithms:
  - start from an *infeasible* solution (optimal), at each iteration try to “improve" the feasibility of the current solution by keeping its optimality.

“Greedy” algorithms

- iterative algorithms
- at each iteration, perform the “best choice" with respect to an appropriate score (easily computable)
- the performed choices are never changed

“Greedy” heuristics

- Ex. Greedy algorithm KP-01(Dantzig, 1957)
  1. Sort the items according to decreasing values of profit/weight ratio $p_j/w_j \geq p_{j+1}/w_{j+1}$
  2. Insert the items until the first item s that does not fit (critical item)

$$s = \min\left\{ i : \sum_{j=1}^{i} w_j > c \right\}$$

Dantzig Algorithm example

- $n = 5$, $c = 10$
- $p = (12, 12, 7, 6, 2)$ $w = (4, 5, 3, 3, 2)$
- $s = 3$, $(x_j) = (1, 1, 0, 0, 0)$, $z = 24$
- An upper bound can be computed by inserting a fraction of the critical item that fills the knapsack
  - U = [12 + 12 + 1/3*7] = 26
- Optimal solution $(x_j) = (1, 0, 1, 1, 0)$, $z^* = 25$

Example 2

Example 2: $n = 6$; $c = 85$;

$$
\begin{array}{lcl}
(p_j) & = & (\ 110,\ 150,\ 70,\ 80,\ 35,\ 5\ ) \\
(w_j) & = & (\ 40,\ 60,\ 30,\ 40,\ 20,\ 5\ ) \\
(p_j/w_j) & = & (\ 2.75,\ 2.5,\ 2.33,\ 2.0,\ 1.75,\ 1.0\ )
\end{array}
$$

Optimal solution: $S = \{1, 4, 6\}$, value$= z^* = 195$.

Greedy1: $S = \{1, 3, 6\}$, value$= 185$.

Dual Greedy

Greedy2 (Dual Algorithm, Time complexity $O(n^2)$):

1. Initialize:

$S := \{1, \ldots, n\}$ (selected items),

$W := \sum_{j=1}^{n} w_j$ (weight of the selected items);

2. if $W \leq c$ ($\Leftrightarrow$ S feasible) STOP;

3. among the items $j \in S$, determine the item j^* having the *minimum score* p_j/w_j and set $S := S \setminus \{j^*\}$, $W := W - w_{j^*}$; go to 2.

Example 2:

Greedy2: $S = \{1\}$, value= 110.

It can be shown that the solution found by *Greedy1* cannot be worse than that found by *Greedy2*.

Primal and Dual Algorithms

- Primal Algorithms:
  - start from a *feasible* solution, at each iteration try to “improve" the value of the objective function of the current solution by keeping its feasibility.
- Dual Algorithms:
  - start from an *infeasible* solution (optimal with respect to the value of the objective function), at each iteration try to “improve" the feasibility of the current solution by keeping its optimality.

Construction Heuristics

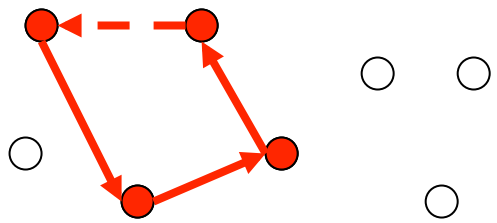
- The solution is iteratively defined according to a simple expansion rule
- In some cases they are based on a pre-order of the elements (Dispatching Rule):
  - The elements are inserted in the solution according to the specified order
- Greedy Algorithm:
  - Expansion rule: maximization (or min) of a simple local criterion
  - No backtracking

Construction Heuristics for TSP

- Tour Construction Methods:
  - Expand a cycle or a path until a feasible solution is obtained
- Main components:
- Choiche of the initial cycle (or vertex)
- At each iteration:
  - Selection criterion: choice of the vertex to be inserted
  - Insertion criterion: choice of the insertion position

Construction Heuristics for TSP (2)

- At the generic iteration *h* the current (partial) is a path (or a cycle) that spans a vertex subset $S \subseteq V$:
  - sequence $\{s_1, s_2, \ldots, s_{|S|}\}$



Stop when $S \equiv V$ $(|S| = n)$

Nearest Neighbor algorithm

- expands a path

  Choose an initial vertex *i* arbitrarily,

  it := 1, s(1) := *i*,

  while it < *n* do

  determine *k* : $c_{s(it),k} := \min \{c_{s(it),j} : j \notin S\}$,

  s(it+1) := k, it := it+1,

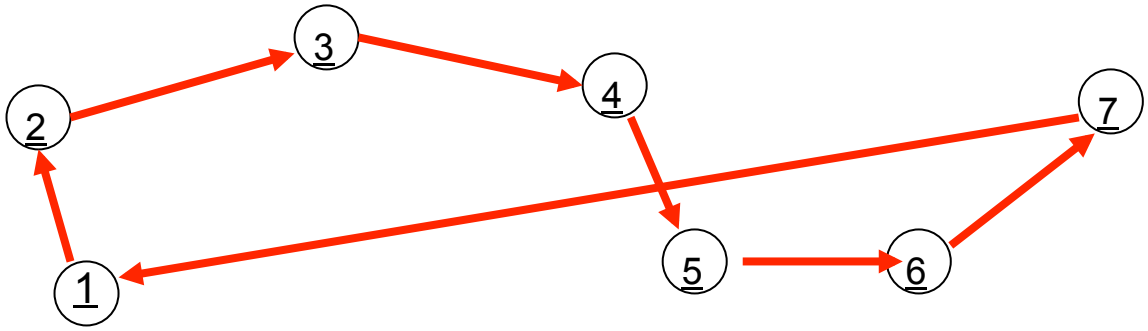
  end while

  $O(n^2)$

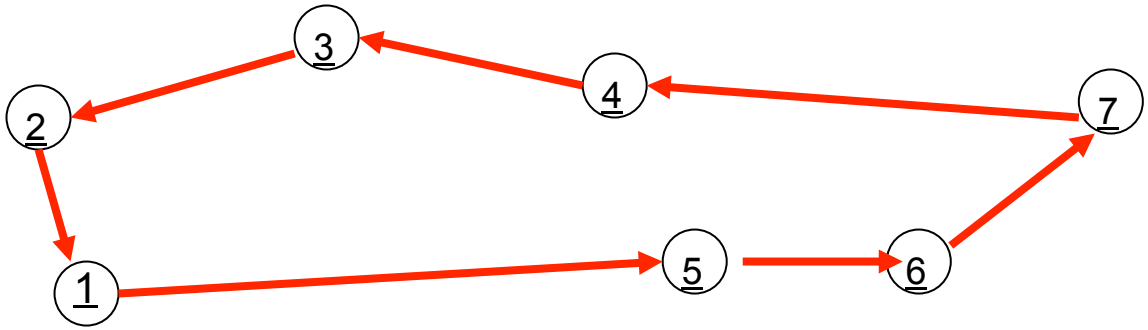
- Often the closing arc is very long

Nearest Neighbor algorithm (2)

- Ex. i = 1



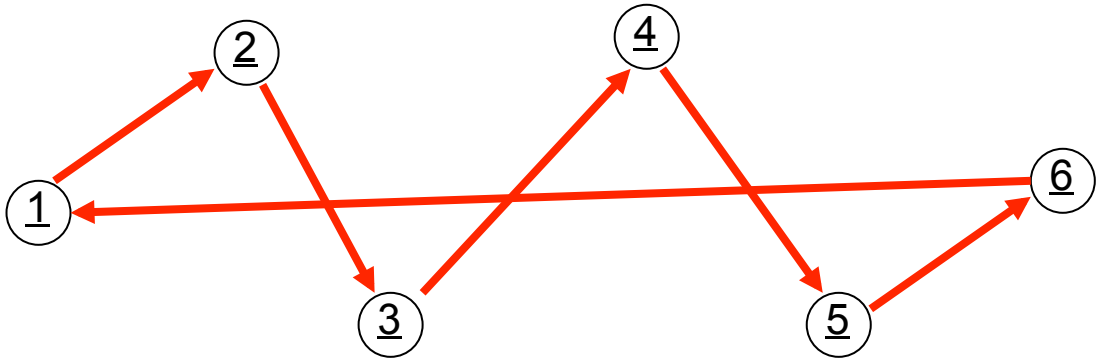
- Ex. i = 5



- Executed *n* times (starting from each vertex)

Nearest Neighbor algorithm (3)

- Sometimes the optimal tour cannot be found



optimal tour {1,2,4,6,5,3}

Randomized algorithms

- Variant of the greedy algorithms: at each iteration, “randomly perturb" the scores.
- Different executions of the algorithm generally lead to different solutions.
- Execute the algorithm several times, and store the best solution found.
- Ex. KP01: replace the score with $p_j/w_j + r_j$ where r_j uniformly random in [0,d], d ≥ 0

Randomized Algoritmhms

Example 2: $n = 6$; $c = 85$ (consider $d = 0.6$);

$$
\begin{array}{lcl}
(p_j) & = & (\ 110,\ 150,\ 70,\ 80,\ 35,\ 5\ ) \\
(w_j) & = & (\ 40,\ 60,\ 30,\ 40,\ 20,\ 5\ ) \\
(p_j/w_j) & = & (\ 2.75,\ 2.5,\ 2.33,\ 2.0,\ 1.75,\ 1.0\ )
\end{array}
$$

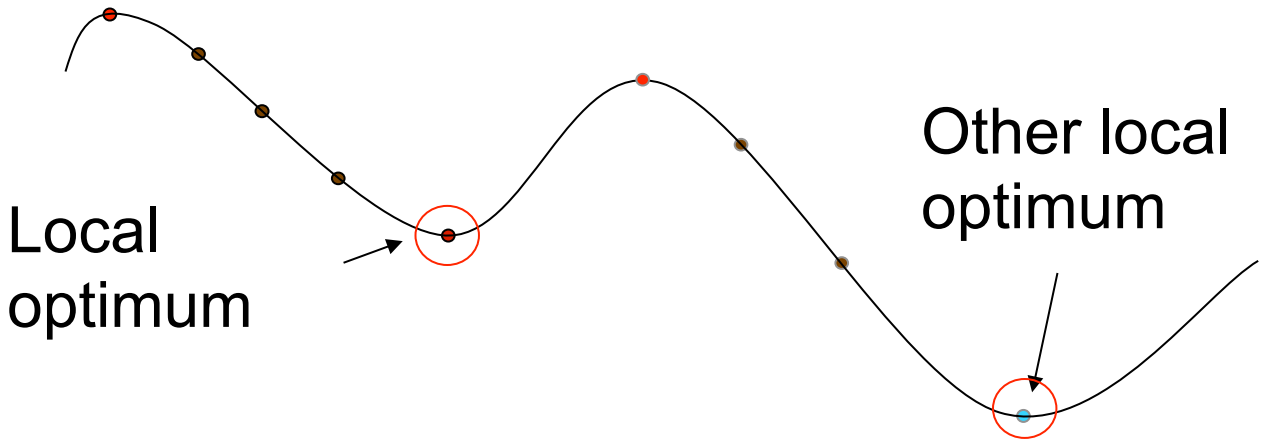
- Execution 1:
  Iter. 1: $r_1 := 0.2, r_2 := 0.4, \ldots; j^* := 1, S := \{1\}, \bar{c} := 45.$
  Iter. 2: $r_3 := 0, r_4 := 0.1, r_5 := 0.6, \ldots; j^* := 5, S := \{1, 5\}, \bar{c} := 25.$
  Iter. 3: $\ldots; j^* := 6, S := \{1, 5, 6\}, \bar{c} := 20$; value $= 150$.

- Execution 2:
  Iter. 1: $r_1 := 0.1, r_2 := 0.2, \ldots; j^* := 1, S := \{1\}, \bar{c} := 45.$
  Iter. 2: $r_3 := 0.0, r_4 := 0.4, \ldots; j^* := 4, S := \{1, 4\}, \bar{c} := 5.$
  Iter. 3: $\ldots; j^* := 6, S := \{1, 4, 6\}, \bar{c} := 0$; value $= 195$.
  (the optimal solution has been found!)

2.2
Local search algorithms

Local search algorithms

- Start from a (feasible) solution
- Iteratively attempt to improve it through simple modifications of the current solution (moves)
- Stop when the current solution is no longer improved by the movEx.



Local search algorithms

Ex. KP-01, $z = \sum p_i x_i$, e $W' = \sum w_i x_i$

Type of moves:

1. If an item j exists: $x_j = 0$ and $W' + w_j \leq c$, set $x_j = 1$ and update z and W'
2. If a pair (i, j) of items exists: $x_i = 1$ and $x_j = 0$ and with $p_i \leq p_j$, $w_i \geq w_j$ (not possible with Greedy) set $x_i = 0$ and $x_j = 1$ and update z and W'
3. If a triple (i, j, h) of items exists: $x_i = 1$ and $x_j = x_h = 0$ and with $p_i < p_j + p_h$, $W' - w_i + w_j + w_h \leq c$ set $x_i = 0$ and $x_j = x_h = 1$ and update z and W'

Example KP-01

Local_Search:

1. The current solution S is the initial solution (found through one of the previous algorithms);

2. Consider all the feasible solutions corresponding to the *moves*:
   $S \cup \{j\}$ (with $j \notin S$), *(insertion of 1 item)*
   $S \cup \{j\} \setminus \{i\}$ (with $j \notin S, i \in S$), *(exchange between 2 items)*
   let R be the best of these solutions;

3. if $\sum_{j \in R} p_j \leq \sum_{j \in S} p_j$, STOP ($\Leftrightarrow$ "local optimum");

4. set $S := R$ and go to 2.

Example KP-01

$$n = 6,\ c = 85 \quad (p_j) = (\ 110,\ 150,\ 70,\ 80,\ 35,\ 5\ )$$
$$(w_j) = (\ 40,\ 60,\ 30,\ 40,\ 20,\ 5\ )$$

Optimal solution: $S = \{1, 4, 6\}$, value $= z^* = 195$.

- Starting from the solution $S = \{2, 5, 6\}$ (value $= 190$):

  no feasible move exists (local optimum).

- Starting from the solution $S = \{1, 3, 6\}$ (value $= 185$),
  the new solution is:

  $S = \{1, 4, 6\}$ (value $= 195$, exchange between items 4 and 3):
  local optimum.

- Starting from the solution $S = \{3\}$ (value $= 70$),
  the new solutions are:

  $S = \{1, 3\}$ (value $= 180$, addition of item 1);

  $S = \{1, 4\}$ (value $= 190$, exchange between items 4 and 3);

  $S = \{1, 4, 6\}$ (value $= 195$, addition of item 6): local optimum.

Neighborood Function

- Given an optimization problem $P = (f, S)$
  - S set of feasible solutions of P
  - $f : S \rightarrow R$ objective function
  - S^* set of optimal solutions
- Neighborhood:

  $N : S \rightarrow 2^S$ that $\forall\, i \in S$ defines $N(i) \subseteq S$

  set of the solutions “close” to i
  - In general $i \in N(i)$, $\forall\, i \in S$
- Local Search algorithms attempt to improve a solution by exploring its neighborhood

Iterative Improvement Algorithm

- Basic Local Search (simple descent)
- "First Improvement" version

```
Procedure FI_Simple_Descent (s)
 /*s ∈ S initial solution*/
    found := TRUE ;
    while found = TRUE do
          found := FALSE ;
          for each s′ ∈ N (s) do
                if  f (s′) < f (s) then
                      s := s′ ; found = TRUE ; break;
    end while;
    return (s);
```

Iterative Improvement Algorithm

- Converges at a local optimum s with respect to $N(\bullet)$, i.e. $s : f(s) \leq f(i)\ \forall\ i \in N(s)$
- "Best Improvement" version

```
Procedure BI_Simple_Descent (s)
    found := TRUE ;
    while found = TRUE do
          found := FALSE ; s_best = s;
          for each s' ∈ N (s) do
                if  f (s') < f (s_best) then s_best = s'
          if s_best < > s then s := s_best ; found =
    TRUE ;
    end while;
    return (s);
```

Iterative Improvement Algorithm

- $|N(s)|$:= Neighborhood cardinality
- each iteration requires time:

  O($|N(s)|$ × time to evaluate f(s))
- The number of iterations required to reach a local optimum may be very high

Local Search Algorithms

- Wide applicability
- High flexibility with respect to problem modifications. They require :
  - Evaluator of a solution (objective function)
  - Feasibility check of a solution
  - Neighborhood function
  - Efficient Neighborhood search technique
- Can be used also when the current solution is not feasible

  $f_{LS} = f + penalty * constraint\ violation$

Local Search Algorithms (2)

- Central problem: Neighborhood choice
  - Highly problem dependant
  - No general rules available
- objective: build Neighborhood Functions that lead to high quality solutions
- Given a problem it is possible to define various Neighborhood Functions
- Ex. KP-01: insertion, exchange 1-1, exchange 1-2 …

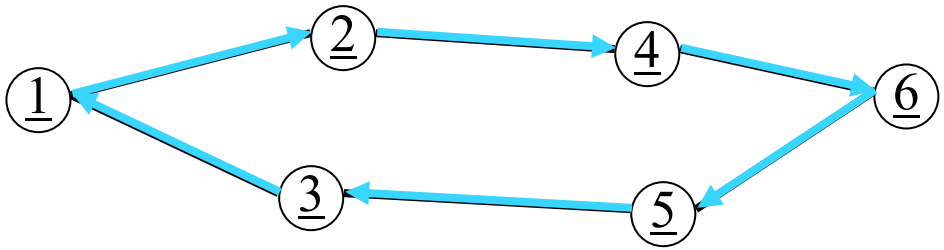
Exchange Neighborhoods

- Very general class : can be applied when the solution is a sequence (TSP) or a partition (KP)
- Ex. KP-01, $z = \sum p_i x_i$, e $W' = \sum w_i x_i$

1. Insertion: if an item j exists such that $x_j = 0$ e $W' + w_j \leq c$, Let $x_j = 1$ and update z and W'
   - Solution evaluation: constant time O(1)
   - $|N(s)| = O(n)$
   - time for an iteration: $O(n)$

Exchanges for the TSP

- Given a solution *s* t may be described through a successor vector
- $\sigma(i)$:= successor of vertex *i* in the solution



i	1	2	3	4	5	6
$\sigma(i)$	2	4	1	6	3	5
$\pi(i)$	3	1	5	2	6	4

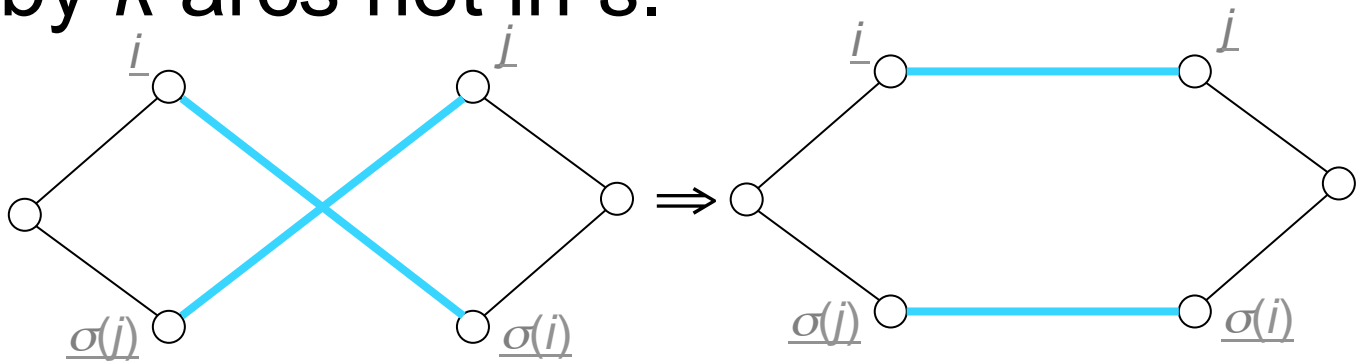
If the solution is oriented we can also define:
$\pi(i)$:= predecessor of vertex *i* in the solution

Note that $\pi(\sigma(i)) = \sigma(\pi(i)) = i$

Exchanges for the TSP (2)

- We remove *k* arcs from *s* and we replace them by *k* arcs not in *s*.

k = 2



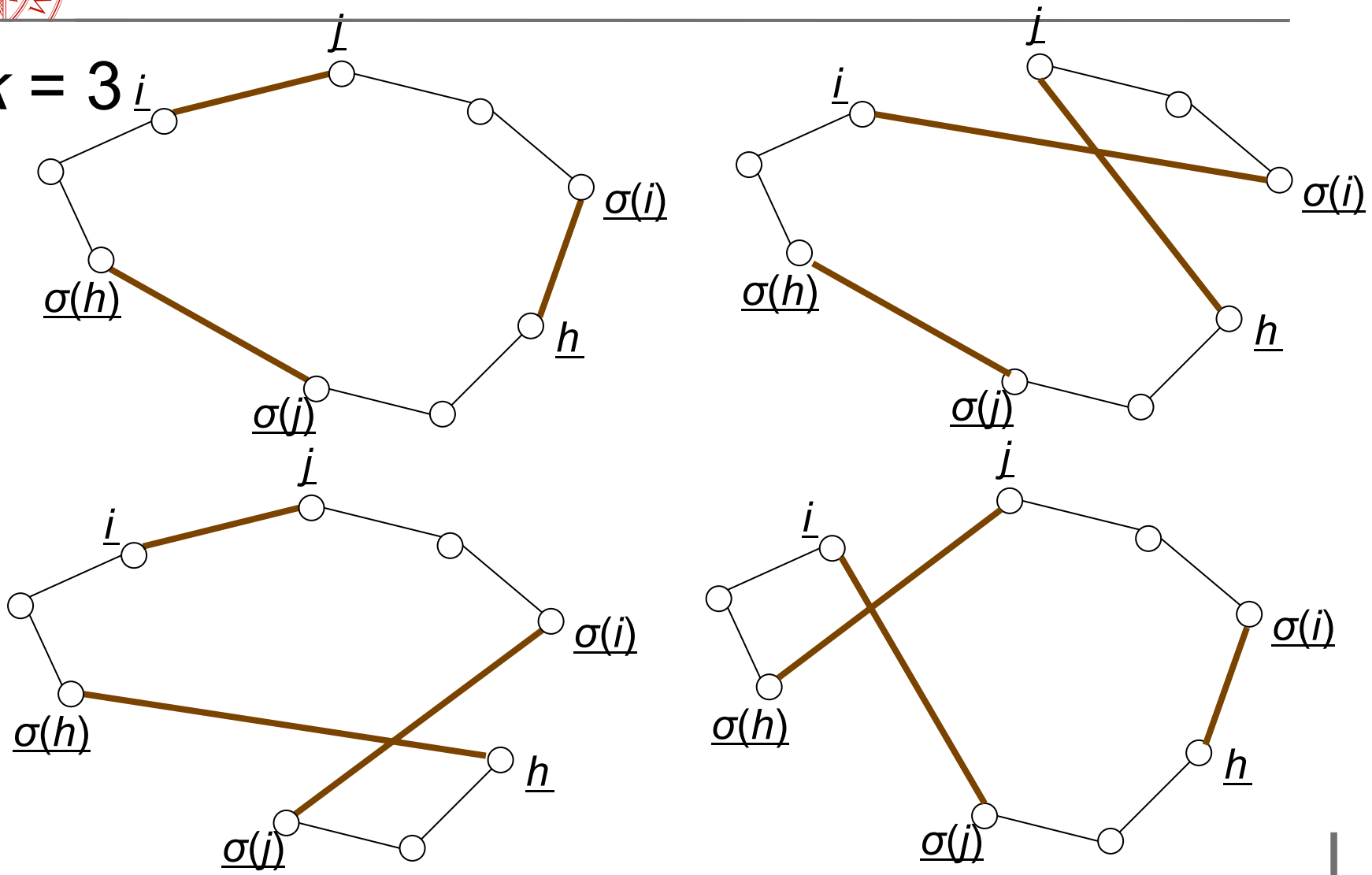
1. remove arcs $(i,\sigma(i))$ e $(j,\sigma(j))$
2. insert arcs (i, j) e $(\sigma(i),\sigma(j))$

cost new sol. $\gamma_{ij} = c_{i,j} + c_{\sigma(i),\sigma(j)} - c_{\iota,\sigma(i)} - c_{j,\sigma(j)}$

$|N(s)| = O(n^2)$, evaluation in $O(1)$

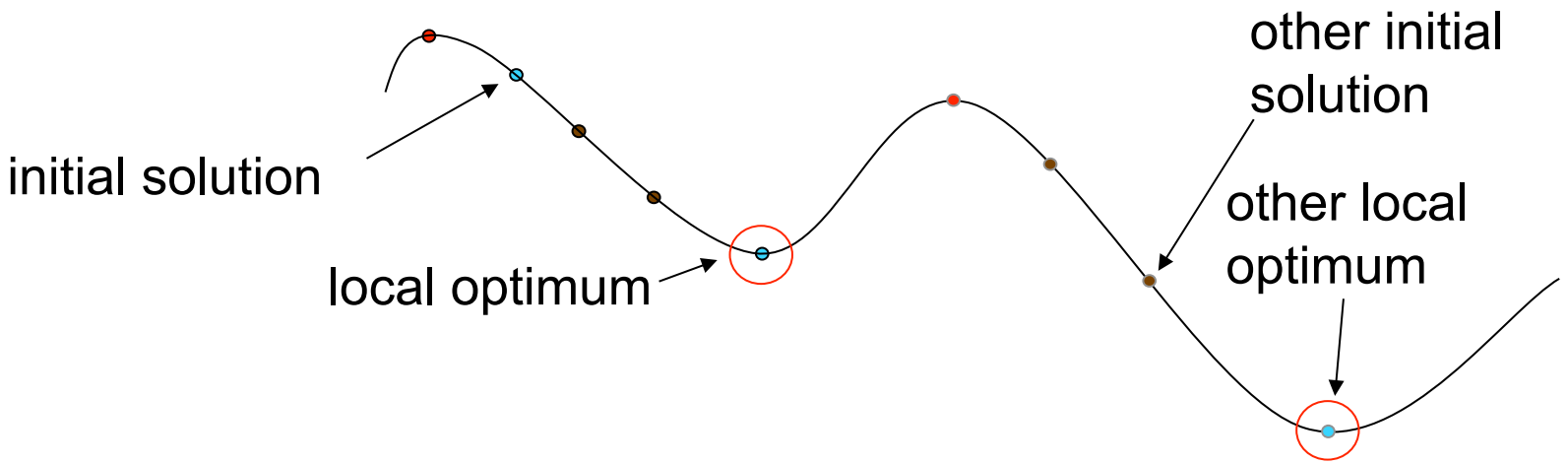
Exchanges for the TSP (3)

- $k = 3$



Improvements

- The quality of the reached local optimum depends from the initial solution and the neighborhood used
- Multi-Start Technique: generate different stating solutions + local search for each



Multi-start

```
s_best :=  + ∞;
repeat
    Generate a solution s (different at each iteration);
    s' = FI_Simple_Descent(s);  /* or BI_Simple_Descent(s) */
    if  f (s′) < f (s_best)
          then s_best := s′;
until  (stopping criterion) /*nº iterations,  maximum time */
```

- Multi-Level: a different $N(\bullet)$ can be used at each iteration