

# Heuristic Algorithms

Master's Degree in Computer Science/Mathematics

Roberto Cordone

DI - Università degli Studi di Milano



Schedule: Thursday 14.30 - 16.30 on MS-Teams

Friday 14.30 - 16.30 on MS-Teams

Office hours: on appointment

E-mail: [roberto.cordone@unimi.it](mailto:roberto.cordone@unimi.it)

Web page: <https://homes.di.unimi.it/cordone/courses/2020-ae/2020-ae.html>

Ariel site: <https://rcordoneha.ariel.ctu.unimi.it>

*Algorithm* SteepestDescent( $I, x^{(0)}$ )

$x := x^{(0)};$

Stop := *false*;

While Stop = *false* do {  $t_{\max}$  iterations }

$\tilde{x} := \arg \min_{x' \in N(x)} f(x);$

If  $f(\tilde{x}) \geq f(x)$  then Stop := *true*; else  $x := \tilde{x};$

EndWhile;

Return ( $x, f(x)$ );

The complexity of the *steepest descent* heuristic depends on

- ① the number of iterations  $t_{\max}$  from  $x^{(0)}$  to the local optimum found, which depends on the structure of the search graph (width of the attraction basins) and is hard to estimate *a priori*
- ② the search for the best solution in the neighbourhood ( $\tilde{x}$ ), which depends on how the search itself is performed, but usually only requires a standard complexity estimation

# The exploration of the neighbourhood

Two strategies to explore the neighbourhood are possible

- ① **exhaustive search**: evaluate all the neighbour solutions;  
the complexity of a single step is the product of
  - the number of neighbour solutions ( $|N(x)|$ )
  - the evaluation of the cost of each solution ( $\gamma_f(|B|, x)$ )

If it is not possible to generate only feasible solution:

- visit a superset of the neighbourhood ( $\tilde{N}(x) \supset N(x)$ )
  - for each element  $x$ , evaluate the feasibility ( $\gamma_x(|B|, x)$ )
  - for the feasible ones, evaluate the cost ( $\gamma_f(|B|, x)$ )
- ② **efficient exploration of the neighbourhood** without a complete visit:  
find the best neighbour solution solving an auxiliary problem

*Only some special neighbourhoods allow that*

# Exhaustive visit of the neighbourhood

*Algorithm* SteepestDescent( $I, x^{(0)}$ )

$x := x^{(0)}$ ;

Stop := false;

While Stop = false do

$\tilde{x} := x$ ;

$$\{ \tilde{x} := \arg \min_{x' \in N(x)} f(x') \}$$

    For each  $x' \in \tilde{N}(x)$  do

        If  $x' \in N(x)$  then

            If  $f(x') < f(\tilde{x})$  then  $\tilde{x} := x'$ ;

        EndIf;

    EndFor;

    If  $f(\tilde{x}) \geq f(x)$  then Stop := true; else  $x := \tilde{x}$ ;

EndWhile;

Return ( $x, f(x)$ );

The complexity of the neighbourhood exploration combines three terms

- 1  $|\tilde{N}(x)|$ : the number of subsets visited
- 2  $\gamma_X$ : the time to evaluate their feasibility
- 3  $\gamma_f$ : the time to evaluate the objective for a feasible solution

# Evaluating or updating the objective: the additive case

The first way to accelerate an exchange algorithm is to **minimize the time to evaluate the objective**: in particular, it is faster to **update  $f(x)$  rather than to recompute it**

The update of an additive objective  $f(x) = \sum_{j \in x} \phi_j$  requires to

- sum  $\phi_i$  for each element  $i \in A$ , added to  $x$
- subtract  $\phi_j$  for each element  $j \in D$ , deleted from  $x$

$$\delta f(x, A, D) = f(x \cup A \setminus D) - f(x) = \sum_{i \in A} \phi_i - \sum_{j \in D} \phi_j$$

Examples: swap of objects (*KP*), columns (*SCP*), edges (*CMSTP*), ...

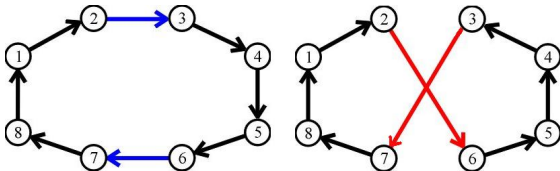
This update has two fundamental properties:

- it takes **constant time for a constant number of elements  $|A| + |D|$**
- **$\delta f(x, A, D)$  does not depend on  $x$**  (we will talk about it later)

# Example: the symmetric $TSP$

To generate neighbourhood  $N_{\mathcal{R}_2}$  for the  $TSP$  we

- delete two nonconsecutive arcs  $(s_i, s_{i+1})$  and  $(s_j, s_{j+1})$
- add the two arcs  $(s_i, s_j)$  and  $(s_{i+1}, s_{j+1})$
- revert the path  $(s_{i+1}, \dots, s_j)$  (modifying  $O(n)$  arcs!)



If the graph and the cost function are symmetric, the variation of  $f(x)$  is

$$\delta f(x, A, D) = c_{s_i, s_j} + c_{s_{i+1}, s_{j+1}} - c_{s_i, s_{i+1}} - c_{s_j, s_{j+1}}$$

but this is not true for the asymmetric  $TSP$

*What if the objective function is not additive?*

# Evaluating or updating the objective: the quadratic case

The *MDP* has a quadratic objective function: computing it costs  $\Theta(n^2)$   
Moving from  $x$  to  $x' = x \setminus \{i\} \cup \{j\}$  (neighbourhood  $N_{S_1}$ ), the update is

$$\delta f(x, i, j) = f(x \setminus \{i\} \cup \{j\}) - f(x) = \sum_{h, k \in x \setminus \{i\} \cup \{j\}} d_{hk} - \sum_{h, k \in x} d_{hk}$$

which depends on  $O(n)$  distance terms, related to points  $i$  and  $j$

There is a general trick for the symmetric quadratic functions with  $d_{ii} = 0$

$$\begin{aligned} \delta f(x, i, j) &= \sum_{h \in x \setminus \{i\} \cup \{j\}} \sum_{k \in x \setminus \{i\} \cup \{j\}} d_{hk} - \sum_{h \in x} \sum_{k \in x} d_{hk} \Rightarrow \\ \Rightarrow \delta f(x, i, j) &= 2 \sum_{k \in x} d_{jk} - 2 \sum_{k \in x} d_{ik} - 2d_{ij} = 2(D_j(x) - D_i(x) - d_{ij}) \end{aligned}$$

If  $D_\ell(x) = \sum_{k \in x} d_{\ell k}$  is known for each  $\ell \in B$ , the computation takes  $O(1)$

# Example: the *MDP*

Let us consider  $f(x)/2$

Evaluate the exchange

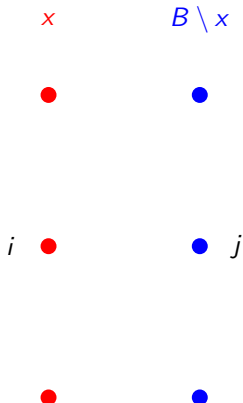
$$x \rightarrow x' = x \setminus \{i\} \cup \{j\}$$

with  $i \in x$  and  $j \in B \setminus x$

$$f(x') = f(x) - D_i + D_j - d_{ij}$$

- the pairs including  $i$  are lost
- the pairs including  $j$  are acquired
- but the pair  $(i, j)$  is in excess

The cost is computed in  $O(1)$  time for each solution





# Example: the *MDP*

Let us consider  $f(x)/2$

Evaluate the exchange

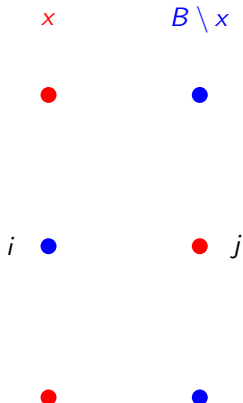
$$x \rightarrow x' = x \setminus \{i\} \cup \{j\}$$

with  $i \in x$  and  $j \in B \setminus x$

$$f(x') = f(x) - D_i + D_j - d_{ij}$$

- the pairs including  $i$  are lost
- the pairs including  $j$  are acquired
- but the pair  $(i, j)$  is in excess

The cost is computed in  $O(1)$  time for each solution



# Example: the *MDP*

Let us consider  $f(x)/2$

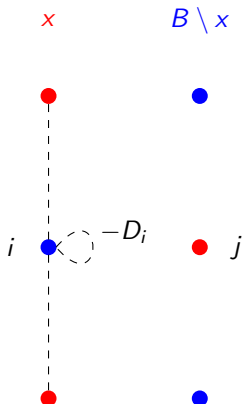
Evaluate the exchange

$$x \rightarrow x' = x \setminus \{i\} \cup \{j\}$$

with  $i \in x$  and  $j \in B \setminus x$

$$f(x') = f(x) - D_i + D_j - d_{ij}$$

- the pairs including  $i$  are lost
- the pairs including  $j$  are acquired
- but the pair  $(i, j)$  is in excess



The cost is computed in  $O(1)$  time for each solution

# Example: the MDP

Let us consider  $f(x)/2$

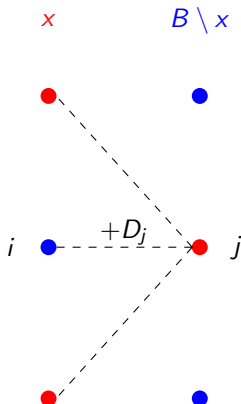
Evaluate the exchange

$$x \rightarrow x' = x \setminus \{i\} \cup \{j\}$$

with  $i \in x$  and  $j \in B \setminus x$

$$f(x') = f(x) - D_i + D_j - d_{ij}$$

- the pairs including  $i$  are lost
- the pairs including  $j$  are acquired
- but the pair  $(i, j)$  is in excess



The cost is computed in  $O(1)$  time for each solution

# Example: the *MDP*

Let us consider  $f(x)/2$

Evaluate the exchange

$$x \rightarrow x' = x \setminus \{i\} \cup \{j\}$$

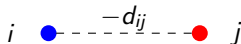
with  $i \in x$  and  $j \in B \setminus x$

$$f(x') = f(x) - D_i + D_j - d_{ij}$$

- the pairs including  $i$  are lost
- the pairs including  $j$  are acquired
- but the pair  $(i, j)$  is in excess

The cost is computed in  $O(1)$  time for each solution

$x$                        $B \setminus x$



# Example: the *MDP*

$x$        $B \setminus x$

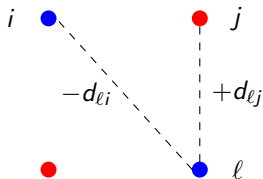


Update of the data structures:

- $D_\ell = D_\ell - d_{\ell i} + d_{\ell j}$ ,  $\ell \in B$

For each element  $\ell \in B$

- $d_{\ell i}$  disappears
- $d_{\ell j}$  appears



The auxiliary data structure is updated in  $O(n)$  time for each iteration

# Example: the *MDP*

$x$        $B \setminus x$

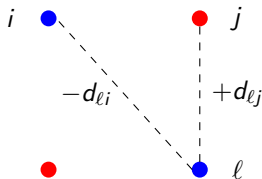


Update of the data structures:

- $D_\ell = D_\ell - d_{\ell i} + d_{\ell j}$ ,  $\ell \in B$

For each element  $\ell \in B$

- $d_{\ell i}$  disappears
- $d_{\ell j}$  appears



The auxiliary data structure is updated in  $O(n)$  time for each iteration

# Example: the *MDP*

$x$        $B \setminus x$

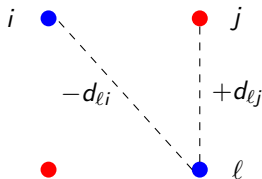


Update of the data structures:

- $D_\ell = D_\ell - d_{\ell i} + d_{\ell j}$ ,  $\ell \in B$

For each element  $\ell \in B$

- $d_{\ell i}$  disappears
- $d_{\ell j}$  appears



The auxiliary data structure is updated in  $O(n)$  time for each iteration

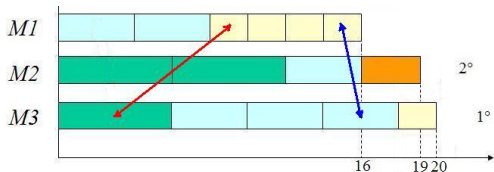
# Updating the objective function: nonlinear examples

Many nonlinear functions can be updated with similar tricks

- save aggregated information on the current solution  $x^{(t)}$
- use it to compute  $f(x')$  efficiently for each  $x' \in N(x^{(t)})$
- update it when moving to the following solution  $x^{(t+1)}$

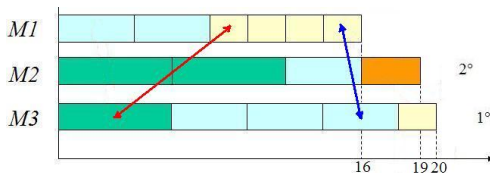
Using the transfer ( $N_{T_1}$ ) and swap ( $N_{S_1}$ ) neighbourhoods for the *PMSP*, the objective can be updated in constant time by managing

- 1 the completion time for each machine
- 2 the indices of the machines with the first and second maximum time





# Example: the *PMSP*



Consider the exchange  $o = (i, j)$  of tasks  $i$  and  $j$   
( $i$  on machine  $M_i$ ,  $j$  on machine  $M_j$ )

- compute in constant time the new completion times:  
one increases, the other decreases (or both remain constant)
- test in constant time whether either exceeds the maximum
- if the maximum time decreases, test in constant time whether the other time or the second maximum time becomes the maximum

Once the neighbourhood is visited and the exchange selected, update

- the two modified completion times (each one in constant time)
- their positions in a max-heap (each one in time  $O(\log |M|)$ )

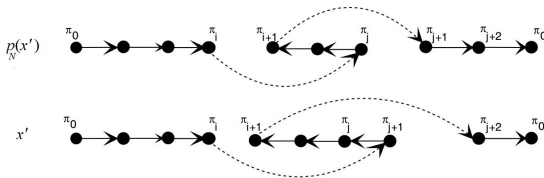
# Use of local auxiliary information

The auxiliary information used to compute  $f(x')$  can be

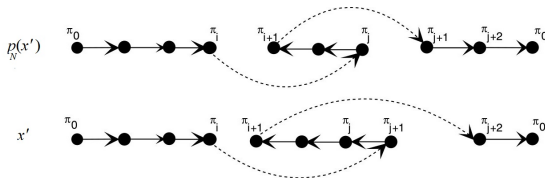
- global, that is referring to the current solution  $x$
- local, that is referring to the solution  $p_N(x')$  visited before  $x'$  in neighbourhood  $N(x)$  according to a suitable order

Consider the neighbourhood  $N_{\mathcal{R}_2}$  for the asymmetric *TSP*:

- the neighbour solutions differ from  $x$  for  $O(n)$  arcs
- general neighbour solutions differ from each other for  $O(n)$  arcs
- if the pairs of arcs  $(s_i, s_{i+1})$  and  $(s_j, s_{j+1})$  follow the lexicographic order, the reverted path changes only by one arc



# Example: the asymmetric TSP



The variation of  $f(x)$  between two generic neighbour solutions is

$$\delta f(x, i, j) = c_{s_i, s_j} + c_{s_{i+1}, s_{j+1}} - c_{s_i, s_{i+1}} - c_{s_j, s_{j+1}} + c_{s_j \dots s_{i+1}} - c_{s_{i+1} \dots s_j}$$

but moving from exchange  $(s_i, s_j)$  to exchange  $(s_i, s_{j+1})$

- the first four terms change, but they can be checked in constant time
- the last two terms can be updated in constant time

$$\begin{cases} c_{s_{j'} \dots s_{i+1}} = c_{s_j \dots s_{i+1}} + c_{s_{j+1}, s_j} \\ c_{s_{i+1} \dots s_{j'}} = c_{s_{i+1} \dots s_j} + c_{s_j, s_{j+1}} \end{cases}$$

*Is it acceptable to explore the neighbourhood in a predefined order ?*

# What about feasibility?

Defining neighbourhoods with the Hamming distance or with operations can generate also unfeasible subsets, that must be removed

$$\tilde{N}_H(x) = \{x' \subseteq B : d(x', x) \leq k\} \supseteq N_O(x) = \tilde{N}_H(x) \cap X$$

$$\tilde{N}_O(x) = \{x' \subseteq B : \exists o \in \mathcal{O} : o(x) = x'\} \supseteq N_O(x) = \tilde{N}_O(x) \cap X$$

(Examples: KP, BPP, SCP, CMSTP...)

If it is not possible to avoid *a priori* the unfeasible subsets, one must

- test the **feasibility** of each element of  $\tilde{N}(x)$  to obtain  $N(x)$
- for the feasible elements, evaluate the **cost**

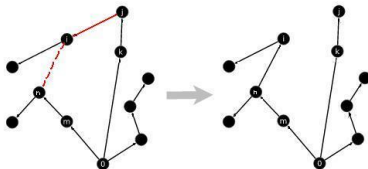
The feasibility test can be made efficient with techniques similar to the ones used for the objective evaluation

*Example: update in constant time the total volume of a subset in the KP*

# Example: the *CMSTP*

Consider the swap neighbourhood  $N_{S_1}$  (add one edge, delete another)

- if the two edges are in the same branch, the solution remains feasible
- if they are in different branches, one loses weight, the other acquires it: **the variation is equal to the weight of the subtree transferred**



If each vertex saves the weight of its appended subtree, to test feasibility compare this weight with the residual capacity of the receiving branch  
(the weight appended to  $i$  with the residual capacity of the left branch)

Once the best exchange is performed, the information must be updated in time  $O(n)$  visiting the ancestors of the receiving vertex  $h$

# A general scheme of sophisticated exploration

The use of auxiliary information requires

- 1 the **inicialization** of suitable data structures
  - partly **local**, i. e., related to neighbour solutions
  - partly **global**, i. e., related to the current solution
- 2 their **update** between subsequent solutions or iterations

*Algorithm SteepestDescent( $I, x^{(0)}$ )*

$x := x^{(0)}$ ; **GI** := **InitializeGI()**; Stop := *false*;

*While* Stop = *false* *do*

$\tilde{x} := 0$ ;  $\tilde{\delta} := 0$ ; **LI** := **InitializeLI()**

*For each*  $x' \in N(x)$  *do*

*If*  $f(x') < f(\tilde{x})$  *then*  $\tilde{x} := x'$ ;

**LI** := **UpdateLI(LI,  $x'$ )**

*EndFor*;

*If*  $f(\tilde{x}) \geq f(x)$

*then* Stop := *true*;

*else*  $x := \tilde{x}$ ; **GI** := **UpdateGD(GI,  $\tilde{x}$ )**

*EndIf*

*EndWhile*;

*Return* ( $x, f(x)$ );

# Partial saving of the neighbourhood (1)

When performing an operation  $o \in \mathcal{O}$  on a solution  $x \in X$  the variation

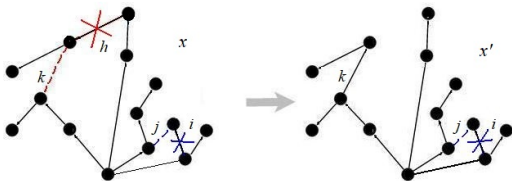
$$\delta f_o(x) = f(o(x)) - f(x)$$

sometimes depends only on a part of  $x$  (possibly, very small)

For example, consider the swap neighbourhood  $N_{S_1}$  for the *CMST*:

- add an edge  $k \in B \setminus x$
- delete an edge  $h \in x$

Two branches are involved: one acquires a subtree, the other loses it



The effect of swap  $(i, j)$  depends only on the branches including  $i$  and  $j$ : it is the same in  $x$  and  $x'$  and is not affected by swap  $(h, k)$

$$\delta f_{i,j}(x) = \delta f_{i,j}(x')$$

## Partial saving of the neighbourhood (2)

There exists a subset of operations  $\tilde{\mathcal{O}} \subset \mathcal{O}$  such that

$$\delta f_o(x') = \delta f_o(x) \text{ for each } o \in \tilde{\mathcal{O}} \text{ and for each } x' = o(x)$$

It is then advantageous to

- 1 compute and save  $\delta f_o(x)$  for every  $o \in \mathcal{O}$ , that is keep the set of feasible exchanges and their associated values  $\delta f$
- 2 perform the best operation  $o^*$ , and generate a new solution  $x'$
- 3 recompute and save  $\delta f_o(x')$  only for  $o \in \mathcal{O} \setminus \tilde{\mathcal{O}}$ , that is remove the exchanges on modified branches, recompute their values, and retrieve  $\delta f_o(x')$  for all  $o \in \tilde{\mathcal{O}}$  (their values are still correct)
- 4 go back to point 2

If the branches are numerous,  $|\mathcal{O} \setminus \tilde{\mathcal{O}}| \subset |\mathcal{O}|$  and the saving is very strong

*It is typical of problems whose solution is a partition*



# Trade-off between efficiency and effectiveness

The complexity of an exchange heuristic depends on three factors

- ① number of iterations
- ② cardinality of the visited neighbourhood
- ③ computation of the feasibility and cost for the single neighbour

The first two factors are clearly conflicting:

- a small neighbourhood is fast to explore, but requires several steps to reach a local optimum
- a large neighbourhood requires few steps, but is slow to explore

The optimal trade-off is somewhere in the middle: a neighbourhood

- large enough to include good solutions
- small enough to be explored quickly

but it is hard to identify, because

- efficiency quickly worsens as size increases
- the resulting solution also changes with the neighbourhood  
(large neighbourhoods have better local optima)

# Fine tuning of the neighbourhoods

It is also possible to define a neighbourhood  $N$  and tune its size

- explore only a promising subneighbourhood  $N' \subset N$

For example, if the objective function is additive, one can

- add only elements  $j \in B \setminus x$  of low cost  $\phi_j$
- delete only elements  $i \in x$  of high cost  $\phi_i$
- terminate the visit after finding a promising solution

For example, the **first-best strategy** stops the exploration at the first solution better than the current one

If  $f(\tilde{x}) < f(x)$  then  $x := \tilde{x}$ ; Stop := true;

The effectiveness depends on the objective

- if the cost of some elements influences very much the objective, it is worth taking it into account, fixing or forbidding them

and on the structure of the neighbourhood

- if the landscape is smooth, the first improving solution approximates well the best solution of the neighbourhood: it is better to stop
- if the landscape is rugged, the best solution of the neighbourhood could be much better: it is better to go on