

Heuristic Algorithms

Master's Degree in Computer Science/Mathematics

Roberto Cordone

DI - Università degli Studi di Milano



Schedule: **Thursday 14.30 - 16.30 in classroom 503**

Friday 14.30 - 16.30 in classroom 503

Office hours: **on appointment**

E-mail: **roberto.cordone@unimi.it**

Web page: **<https://homes.di.unimi.it/cordone/courses/2024-ae/2024-ae.html>**

Ariel site: **<https://myariel.unimi.it/course/view.php?id=4466>**

What to do when the axioms are violated

If a search space violates the desired axioms, one can try and change it

For the *TSP*, an alternative \mathcal{F}_A includes all paths starting from node 1

Let N_x be the set of nodes visited from x : the acceptable extensions are all arcs going out of the last node of path x and not closing a subtour

$$\Delta_A^+(x) = \{(h, k) \in A : h = \text{Last}(x), k \notin N_x \text{ or } k = 1 \text{ and } N_x = N\}$$

Unfortunately, the axioms are still not all satisfied

- the trivial axiom always holds
- the accessibility axiom holds: removing the last arc yields a path starting from node 1
- the hereditary axiom does not hold: not all subsets are paths
- the exchange axiom does not hold

Therefore, it is not even a greedoid

But the algorithm can still be a reasonable heuristic

The *Nearest Neighbour* heuristic for the TSP

The *Nearest Neighbour* (NN) heuristic adopts the alternative search space keeping the objective function as the selection criterium

- Start with an empty set of arcs: $x^{(0)} = \emptyset$
that represents a degenerate path going out of node 1
(*the optimal solution certainly visits node 1*)
- Find the **arc of minimum cost going out of the last node of x**

$$(i, j) = \arg \min_{(h, k) \in \Delta_A^+(x)} c_{hk}$$

(*the objective function is additive*)

- If $j \neq 1$, go back to point 2; otherwise, terminate
($\Delta_A^+(x)$ allows the return to node 1 only at the last step)

The algorithm is very intuitive and **its complexity is $\Theta(n^2)$**

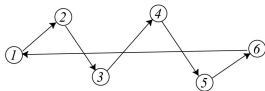
It is not exact, but **log n -approximated** (*under the triangle inequality*)

The *Nearest Neighbour* heuristic: example

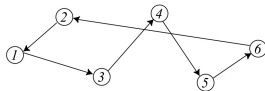
Consider a complete graph (the arcs are not reported for clarity)



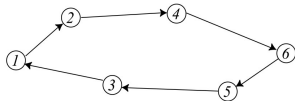
Starting from node 1



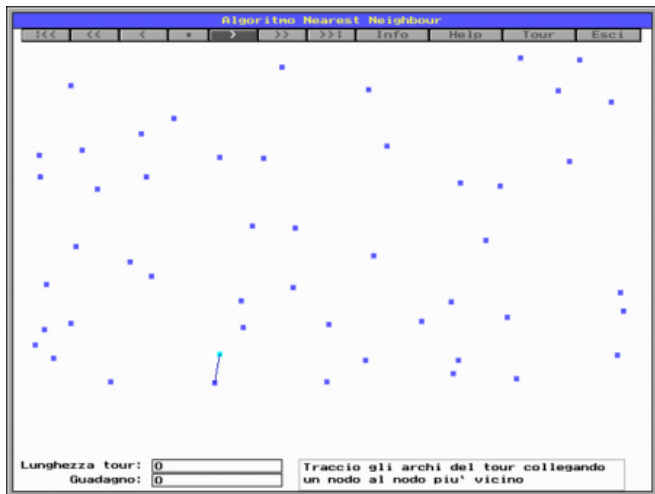
Starting from node 2



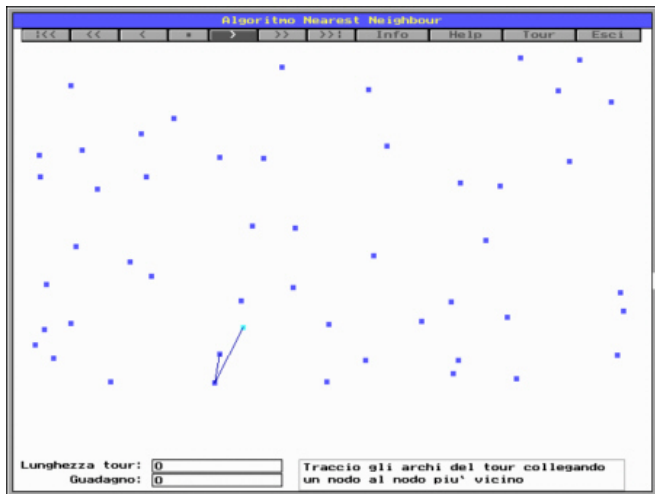
The optimal solution cannot be found starting from any node



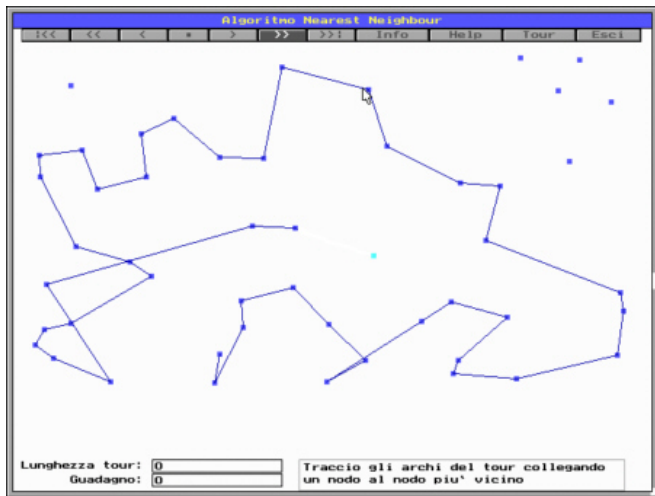
A larger example



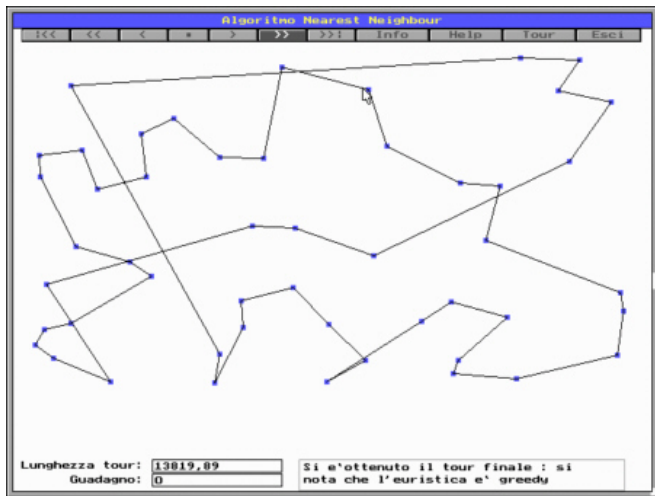
A larger example



A larger example



A larger example



Heuristic constructive algorithms: the KP

If the problem does not admit a search space with suitable properties, one must **keep into account the constraints of the problem** adopting

- 1 not only a good definition of \mathcal{F}_A
- 2 but also a **sophisticated definition of the selection criterium** $\varphi_A(i, x)$

This allows effective results, even if not provably optimal

In the KP , the drawback derives from the volume of the objects: **promising objects have a large value, but also a small volume**

- define the selection criterium as the **unitary value** $\varphi_A(i, x) = \frac{\phi_i}{v_i}$

The resulting algorithm

- can perform very badly
- **with a small modification is 2-approximated**

Example: the KP

B	a	b	c	d	e	f
ϕ	7	2	4	5	4	1
v	5	3	2	3	1	1
ϕ/v	1.40	0.67	2.00	1.67	4	1

$$V = 8$$

The algorithm performs the following steps:

- 1 $x := \emptyset$;
- 2 select $i := e$ and update $x := \{e\}$;
- 3 select $i := c$ and update $x := \{c, e\}$;
- 4 select $i := d$ and update $x := \{c, d, e\}$;
- 5 select $i := f$ and update $x := \{c, d, e, f\}$; (*object a does not fit*)
- 6 since $\Delta_A^+(x) = \emptyset$, terminate

The value of the solution found is 14,
the optimal solution is $x^* = \{a, c, e\}$ and its value is 15

Example: the KP

There are critical cases

B	a	b
ϕ	10	90
v	1	10
ϕ/v	10	9

$$V = 10$$

The algorithm performs the following steps:

- 1 $x := \emptyset$;
- 2 select $i := a$ and update $x := \{a\}$;
- 3 since $\Delta_A^+(x) = \emptyset$, terminate

The value of the solution found is 10, the optimum is 90:

there are instances with unlimitedly worse gaps

The reason of the mistake is that

- the first discarded object
- has a large volume, but also a large value

Example: 2-approximated algorithm for the KP

- 1 Start with an empty subset: $x^{(0)} = \emptyset$
- 2 Find the object $i^{(t)}$ of maximum unitary value in $B \setminus x^{(t-1)}$:

$$i^{(t)} := \arg \max_{i \in B \setminus x^{(t-1)}} \frac{\phi_{i^{(t)}}}{v_{i^{(t)}}}$$

- 3 If it respects the capacity, add $i^{(t)}$ to $x^{(t-1)}$: $x^{(t)} := x^{(t-1)} \cup \{i^{(t)}\}$ and go back to point 2
- 4 Build a solution with the first rejected object: $x' = \{i^{(t)}\}$
- 5 Return the better solution between x and x' : $f_A = \max[f(x), f(x')]$

It is easy to prove that

- the sum of the two solution values overestimates the optimum

$$f(x) + f(x') = \sum_{\tau=1}^t \phi_{i^{(\tau)}} \geq f^*$$

- the best of the two solution values is at least half their sum

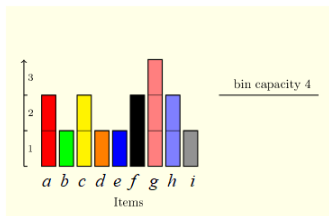
$$f_A = \max[f(x), f(x')] \geq \frac{f(x) + f(x')}{2} \geq \frac{1}{2} f^*$$

Bin Packing Problem

The *BPP* requires to divide a set O of voluminous objects into the minimum number of containers of given capacity drawn from a set C

$B = O \times C$ includes the object-container assignments (i, j)

- with exactly one container for each object
- with the total volume in each container not exceeding the capacity



Let us define the search space \mathcal{F}_A as the set of all partial solutions

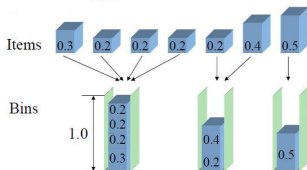
The objective function is a bad selection criterium, because it is flat

All the augmented subsets have the same value or increase it by 1

First-Fit heuristic

Consider the object-container pairs lexicographically

- Start with an empty subset: $x^{(0)} = \emptyset$
- Select pair (i, j) according to the following criterium:
 - i is the **first** (minimum index) **unassigned object**
 - j is the **first container with enough residual capacity** for i
(a used container, if possible; an unused one otherwise)
- Add the new assignment to the solution: $x^{(t)} := x^{(t-1)} \cup \{(i, j)\}$

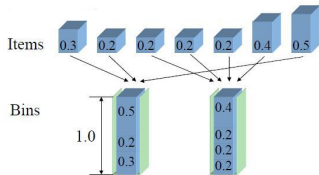
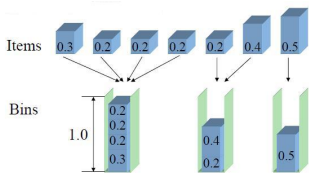


Notice that the choice of (i, j)

- does not minimise $f(x \cup \{(i, j)\})$ (another i could be better)
- is split into two phases (first i , then j)

Properties of the First-Fit heuristic

The solution is not optimal



but it is approximated:

- at least $f^* \geq \sum_{i \in O} v_i / V$ containers are necessary
- the occupied volume is $> V/2$ for all used containers, possibly except for the last one (*if a second half-empty container existed, its objects would have been assigned to the first*)
- the total volume exceeds that of the $f_A - 1$ “saturated” containers

$$\sum_{i \in O} v_i > (f_A - 1) \frac{V}{2}$$

which implies $(f_A - 1) < \frac{2}{V} \sum_{i \in O} v_i \leq 2f^* \Rightarrow f_A \leq 2f^*$

(the analysis can be improved to 1.7)

Decreasing First-Fit heuristic

The approximation ratio $\alpha = 2$ holds for any permutation of the objects

Intuition would suggest to select first the smallest objects, in order to keep the objective $f(x \cup \{i\})$ as small as possible, but this neglects that all objects must be assigned

By contrast, **it is better to select the largest object first** because

- each object in a container has a volume strictly larger than the residual capacity of all the previous containers
(*otherwise, it would have been assigned to one of them*)
- keeping the smallest objects in the end guarantees that many containers have a small residual capacity

This algorithm has a better approximation ratio: $f_A \leq \frac{11}{9} f^* + 1$

Pure and adaptive constructive algorithms

A constructive algorithm A is

- **pure** if the selection criterium φ_A depends only on the new element i
- **adaptive** if φ_A depends both on i and on the current solution x

Many criteria $\varphi_A(i, x)$ admit equivalent forms depending only on i

- in the *TSP*, $\varphi_A((i, j), x) = f(x \cup \{(i, j)\})$ is equivalent to c_{ij}
- in the *KP*, $\varphi_A(i, x) = f(x \cup \{i\})$ is equivalent to ϕ_i

So far, we have seen only pure constructive algorithms

An additive selection criterium yields a pure constructive algorithm

Set Covering

Given a binary matrix and a cost vector associated to the columns, find a minimum cost subset of columns covering all the rows

The objective is additive, but the solutions are not maximal subsets
(*actually, the smaller feasible subsets are better*)

An adaptive selection criterium $\varphi_A(i, x)$ is necessary: a pure one ($\varphi_A(i)$) could repeatedly choose columns covering the same rows

The more promising ideas are to consider

- the **objective function**: select columns of low cost
- the **constraints**: select columns covering many rows
- the **current subset x** : select columns covering new rows

In summary

- include in $\Delta_A^+(x)$ only **columns covering additional rows** not in x
- apply the adaptive selection criterium $\varphi_A(i, x) = \frac{c_i}{a_i(x)}$
where $a_i(x)$ is the **number of rows covered by i , but not by x**

Set Covering: a positive example

$$c \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 3 & 5 & 6 & 2 & 1 & 7 & 1 & 8 \\ \hline \end{array}$$

$$A \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

The algorithm performs the following steps:

- 1 $x := \emptyset$;
- 2 select $i := 1$ ($\varphi_A(i, x) = 3/4$) and update $x := \{1\}$ and $\Delta_A^+(x) = \{2, 5, 6, 7, 8\}$;
- 3 select $i := 5$ ($\varphi_A(i, x) = 1$) and update $x := \{1, 5\}$ and $\Delta_A^+(x) = \{2, 6\}$;
- 4 select $i := 2$ ($\varphi_A(i, x) = 5$) and update $x := \{1, 2, 5\}$;
- 5 now all the rows are covered and $\Delta_A^+(x) = \emptyset$, therefore terminate

The value of the solution found is $3 + 5 + 1 = 9$ and is the optimum

Set Covering: a negative example

But the algorithm can also fail

$$c \quad \begin{bmatrix} 25 & 6 & 8 & 24 & 12 \end{bmatrix}$$

$$A \quad \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The algorithm performs the following steps:

- 1 $x := \emptyset$;
- 2 since $c/a_i(x) = [4.1\bar{6} \quad 2 \quad 4 \quad 12 \quad 12]$, select $i := 2$;
- 3 since $c/a_i(x) = [8.\bar{3} \quad - \quad 8 \quad 12 \quad 12]$, select $i := 3$;
- 4 since $c/a_i(x) = [12.5 \quad - \quad - \quad 24 \quad 12]$, select $i := 5$;
- 5 since $c/a_i(x) = [25 \quad - \quad - \quad 24 \quad -]$, select $i := 4$;
- 6 all the rows are covered, therefore $\Delta_A^+(x) = \emptyset$ and terminate

The solution returned is $x = \{2, 3, 4, 5\}$ and its value is 50,
whereas the optimal solution $x^* = \{1\}$ has value $f^* = 25$

Approximability of the SCP

This algorithm has a nonconstant (logarithmic) approximation ratio

- at each step t , each column i is evaluated with criterium

$$\varphi_A(i, x^{(t-1)}) = \frac{c_i}{a_i(x^{(t-1)})}$$

- each row j is covered by a certain column (i_j) at a certain step (t_j)
- start assigning weight $\theta_j = 0$ to each row j
- when each row j is covered (step t_j), set its weight to

$$\theta_j = \frac{c_{i_j}}{a_{i_j}(x^{(t_j-1)})}$$

so that the total weight of the rows increases by c_{i_j} at step t_j ;
correspondingly, x includes column i_j and its cost increases by c_{i_j}

- the total cost of x is always equal to the total weight of the rows

$$f_A(x) = \sum_{i \in x} c_i = \sum_{j \in R} \theta_j$$

Approximability of the SCP

- at step t , there are $|R^{(t)}| \leq |R| - t$ uncovered rows
- the columns of the optimal solution could cover them all with cost f^*
 \Rightarrow at least one of such columns has unitary cost $\leq f^*/|R^{(t)}|$
- the column i selected has minimum unitary cost $\varphi_A(i, x^{(t-1)})$,
therefore $\leq f^*/|R^{(t)}|$ and the covered rows increase their weight by

$$\theta_j = \varphi_A(i, x^{(t-1)}) \leq \frac{f^*}{|R^{(t_j)}|} \Rightarrow \sum_{j \in R} \theta_j \leq \sum_{j \in R} \frac{f^*}{|R^{(t_j)}|}$$

The cost to cover each row j is not larger than the optimum divided by the number of rows uncovered at the step in which j gets covered

- the integer number $|R^{(t)}|$ strictly decreases at each step
- the sum can be overestimated reducing $|R^{(t)}|$ by 1 at each step
- The approximation ratio is limited by a logarithmic guarantee

$$f_A = \sum_{j \in R} \theta_j \leq \sum_{j \in R} \frac{f^*}{|R^{(t_j)}|} \leq \sum_{r=|R|}^1 \frac{f^*}{r} \leq (\ln |R| + 1) f^*$$

Application to the negative example

c	25	6	8	24	12
A	1	1	0	0	0
	1	1	0	0	0
	1	1	1	0	0
	1	0	1	1	0
	1	0	0	1	0
	1	0	0	0	1

- 1 since $\varphi_A(i, x) = [4.1\bar{6} \quad 2 \quad 4 \quad 12 \quad 12]$, select $i := 2$
and set $\theta_1 = \theta_2 = \theta_3 = 2$, that is $\leq f^*/|R^{(0)}| = 25/6 = 4.1\bar{6}$;
now weight $\theta_1 \leq 25/6$, and even more so $\theta_2 \leq 25/5$ and $\theta_3 \leq 25/4$
- 2 since $\varphi_A(i, x) = [8.\bar{3} \quad - \quad 8 \quad 12 \quad 12]$, select $i := 3$
and set $\theta_4 = 8$, that is $\leq f^*/|R^{(1)}| = 8.\bar{3}$
- 3 since $\varphi_A(i, x) = [12.5 \quad - \quad - \quad 24 \quad 12]$, select $i := 5$
and set $\theta_6 = 12$, that is $\leq f^*/|R^{(2)}| = 12.5$
- 4 since $\varphi_A(i, x) = [25 \quad - \quad - \quad 24 \quad -]$, select $i := 4$
and set $\theta_5 = 24$, that is $\leq f^*/|R^{(3)}| = 25$
- 5 all the rows are covered, therefore $\Delta_A^+(x) = \emptyset$ and the algorithm terminates

Now $f_A = \sum_{j \in R} \theta_j = 50$ and the approximation holds: $f_A \leq (\ln |R| + 1) f^* \approx 2.79 f^*$