Heuristic Algorithms for Combinatorial Optimization problems

Ph.D. course in Computer Science

Roberto Cordone

DI - Università degli Studi di Milano

E-mail: roberto.cordone@unimi.it

Web page: https://homes.di.unimi.it/cordone/courses/2023-haco/2023-haco.html

Constructive heuristics

In Combinatorial Optimization every solution x is a subset of B

A constructive heuristic updates a subset $x^{(t)}$ step by step

1. start from an empty subset: $x^{(0)} = \emptyset$
   (*obviously a subset of any optimal solution*)
2. stop when a termination condition holds
   (*the following subsets cannot be optimal solutions*)
3. select the "best" element $i^{(t)} \in B$ among the "acceptable" ones at the current step t
   (*try and keep $x^{(t)}$ within a feasible and optimal solution*)
4. add $i^{(t)}$ to the current subset $x^{(t)}$: $x^{(t+1)} := x^{(t)} \cup \left\{ i^{(t)} \right\}$
   (*the selection can never be undone!*)
5. go back to point 2

Such processes admit a nice modelling tool

The construction graph

Every construction heuristic A defines a construction graph

- the node set $\mathcal{F}_A \subseteq 2^B$ (search space) is the collection of all subsets $x \subseteq B$ acceptable for A
- the arc set is the collection of all pairs $(x, x \cup \{i\})$ such that $x \in \mathcal{F}_A$, $i \in B \setminus x$ and $x \cup \{i\} \in \mathcal{F}_A$

The arcs represent the elementary extensions of the acceptable subsets

The construction graph is by definition acyclic

Each possible execution of A is a maximal path of the construction graph

- from the empty subset $\emptyset$
- to a subset x that cannot be acceptably extended

$$\Delta_A^+(x) = \{i \in B \setminus x : x \cup \{i\} \in \mathcal{F}_A\} = \emptyset$$

Termination condition

A constructive heuristic A terminates when

- the current subset $x^{(t)}$ has no outgoing arc

$$\Delta_A^+\left(x^{(t)}\right) = \left\{ i \in B \setminus x^{(t)} : x^{(t)} \cup \{i\} \in \mathcal{F}_A \right\} = \emptyset$$

- that is, extending $x^{(t)}$ implies to leave the search space $\mathcal{F}_A$

$$x^{(t)} \cup \{i\} \notin \mathcal{F}_A \text{ for each } i \in B \setminus x^{(t)}$$

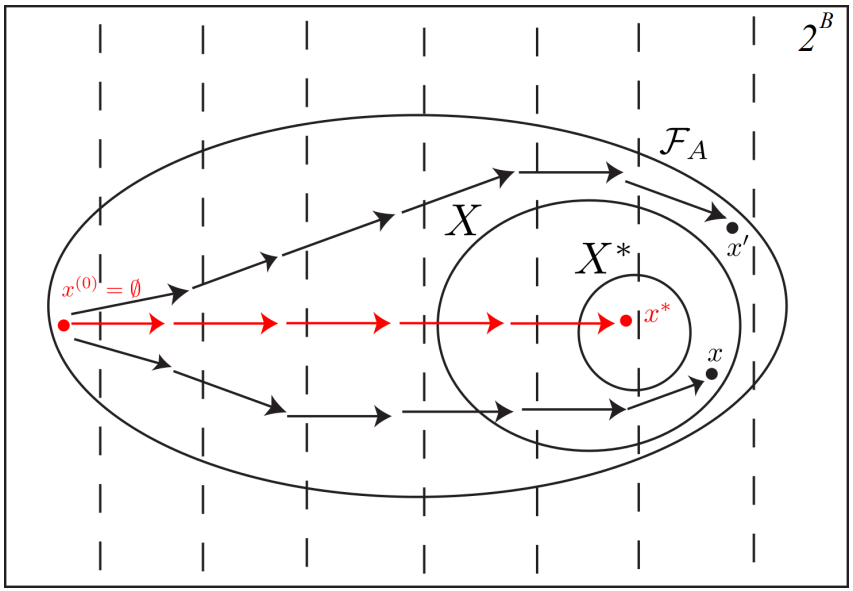
Different behaviours are possible

- sometimes all visited subsets are feasible (e.g., KP)
- often the last subset is the only feasible solution
- $x^{(t)}$ could even move in and out of X (or X^*)

(*but this is uncommon*)

- the path can visit or not X and X^*

Exact constructive algorithms

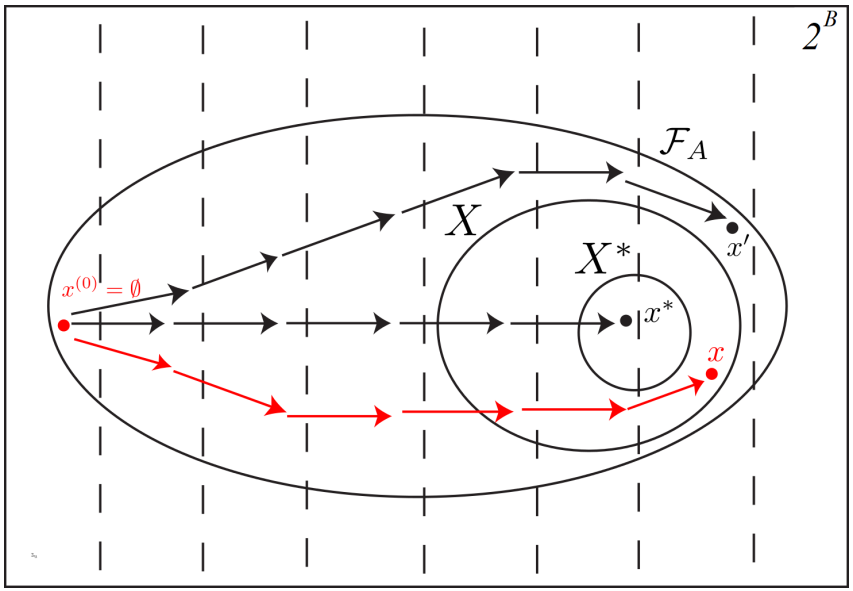


The algorithm visits a sequence of subsets $\emptyset = x^{(0)} \subset \ldots \subset x^{(t_f)}$ terminating

- in an optimal solution $x^* \in X^*$
- in a nonoptimal feasible solution $x \in X$
- in an unfeasible subset x'

Example: *MST* problem, both with $\mathcal{F}_{\text{Kruskal}}$ and $\mathcal{F}_{\text{Prim}}$

Heuristic constructive algorithms (1)

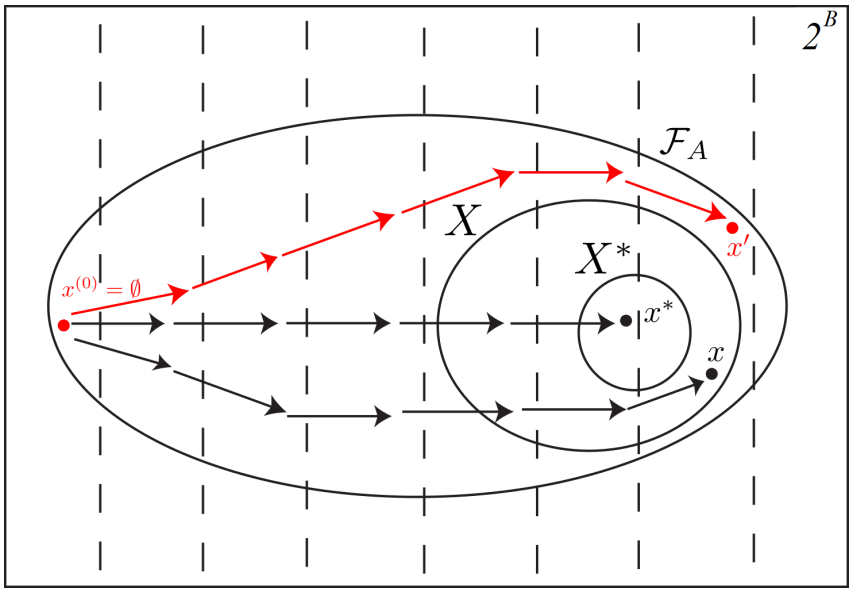


The algorithm visits a sequence of subsets $\emptyset = x^{(0)} \subset \ldots \subset x^{(t_f)}$ terminating

- in an optimal solution $x^* \in X^*$
- in a nonoptimal feasible solution $x \in X$
- in a unfeasible subset x'

Example: *KP*, *MDP*, etc. . .

Heuristic constructive algorithms (2)



The algorithm visits a sequence of subsets $\emptyset = x^{(0)} \subset \ldots \subset x^{(t_f)}$ terminating

- in an optimal solution $x^* \in X^*$
- in a nonoptimal feasible solution $x \in X$
- in an unfeasible subset x'

Example: *TSP* on a noncomplete graph

Pseudocode

A constructive heuristic (for minimization problems) can be described as

Algorithm Greedy(I)

$x := \emptyset$; $x^* := \emptyset$;

If $x \in X$ *then* $f^* := f(x)$ *else* $f^* := +\infty$;

While $\Delta_A^+(x) \neq \emptyset$ *do*

$i := \arg \min_{i \in \Delta_A^+(x)} \varphi_A(i, x)$;

$x := x \cup \{i\}$;

If $x \in X$ and $f(x) < f^*$ *then* $x^* := x$; $f^* := f(x)$;

Return (x^*, f^*);

The path (sequence of subsets) visited by the algorithm is determined by

- the set $\Delta_A^+(x) \subseteq B \setminus x$, that is derived from the construction graph
- the selection criterium $\varphi_A : B \times \mathcal{F}_A \to \mathbb{R}$ used to select the element i to add to the current subset $x^{(t)}$ to generate $x^{(t+1)}$, that can be seen as a weight function on the arcs $(x, x \cup \{i\})$

The solution returned is the best visited during the execution (*usually, the last one*)

Definition of the construction graph

Ideally, the search space $\mathcal{F}_A$ should include

- the empty subset: $\emptyset \in \mathcal{F}_A$ (A *starts from* $\emptyset$)
- all feasible solutions: $X \subseteq \mathcal{F}_A$ (*perhaps excluding provably nonoptimal solutions*)
- only subsets reachable from $\emptyset$ (*unreachable subsets are useless*)

Using $\mathcal{F}_A$ requires a fast inclusion test to answer the decision problem

- "is subset $x^{(t)}$ acceptable?" ($x^{(t)} \in \mathcal{F}_A$?)

or at least a fast update test: if $x^{(t)} \in \mathcal{F}_A$, is $x^{(t)} \cup \{i\} \in \mathcal{F}_A$?

Natural candidates are the partial solutions (subsets of feasible solutions), for which, however, the inclusion test

- "is subset $x^{(t)}$ a partial solution?" ($\exists x \in X : x^{(t)} \subseteq x$?)

generalises $\exists x \in X : \emptyset \subseteq x$? that is the feasibility problem

- "is there any feasible solution?" ($\exists x \in X$?)

and could be $\mathcal{NP}$-complete

In that case, one needs to relax the search space

A natural selection criterium

If the objective function can be extended from X to $\mathcal{F}_A$,
it looks natural to use the objective function as the selection criterium

$$\varphi_A(i, x) = f(x \cup \{i\})$$

Algorithm Greedy(I)
$x := \emptyset$; $x^* := \emptyset$;
If $x \in X$ *then* $f^* := f(x)$ *else* $f^* := +\infty$;
While $\Delta_A^+(x) \neq \emptyset$ *do*
$i := \arg \min_{i \in \Delta_A^+(x)} f(x \cup \{i\})$;
$x := x \cup \{i\}$;
If $x \in X$ and $f(x) < f^*$ *then* $x^* := x$; $f^* := f(x)$;
Return (x^*, f^*);

The fractional knapsack problem (*FKP*)

Select from a set of objects of identical volume a maximum value subset which could be contained in a knapsack of limited capacity

In the *FKP* the capacity simply imposes a cardinality constraint: the feasible solutions are those with $|x| \leq \lfloor V/v \rfloor$

Algorithm GreedyFKP(I)
$x := \emptyset$; $x^* := \emptyset$;
If $x \in X$ *then* $f^* := f(x)$ *else* $f^* := -\infty$;
While $|x| < \lfloor V/v \rfloor$ *do*
$i := \arg\max_{i \in B \setminus x} \phi_i$;
$x := x \cup \{i\}$;
If $x \in X$ and $f(x) > f^*$ *then* $x^* := x$; $f^* := f(x)$;
Return (x^*, f^*);

- Define $\mathcal{F}_A = X$: subset x can be extended as long as $|x| < \lfloor V/v \rfloor$
- Any or no element of $B \setminus x$ extends x feasibly ($\Delta_A(x) = B \setminus x$ or $\emptyset$)
- The objective function is additive, and therefore

$$f(x \cup \{i\}) = f(x) + \phi_i \Rightarrow \arg\max_{i \in B \setminus x} f(x \cup \{i\}) = \arg\max_{i \in B \setminus x} \phi_i$$

- The last subset visited is the best solution found

Example: the fractional knapsack problem

B	a	b	c	d	e	f
ϕ	7	2	4	5	4	1

$$v_i = 1 \text{ for each } i \in B$$

$$V = 4$$

The algorithm performs the following steps:

1. $x := \emptyset$;
2. since $|x| = 0 < 4$, evaluate $i := a$ and update $x := \{a\}$;
3. since $|x| = 1 < 4$, evaluate $i := d$ and update $x := \{a, d\}$;
4. since $|x| = 2 < 4$, evaluate $i := c$ and update $x := \{a, c, d\}$;
5. since $|x| = 3 < 4$, evaluate $i := e$ and update $x := \{a, c, d, e\}$;
6. since $|x| = 4 \not< 4$, terminate

This algorithm always finds the optimal solution

But why?

The knapsack problem

Select from a set of objects of different volume a maximum value subset which could be contained in a knapsack of limited capacity

Algorithm GreedyKP(I)
$x := \emptyset$; $x^* := \emptyset$; $f^* := 0$;
While $\exists i \in B \setminus x : v_i \leq V - \sum_{j \in x} v_j$ *do*
$i := \arg \max_{i \in B \setminus x : v_i \leq V - \sum_{j \in x} v_j} \phi_i$;
$x := x \cup \{i\}$;
Return $(x, f(x))$;

- Define $\mathcal{F}_A = X$: only some elements of $B \setminus x$ extend x feasibly

$$\Delta_A^+(x) = \{i \in B \setminus x : \sum_{j \in x} v_j + v_i \leq V\}$$

- The objective function is additive, and therefore

$$f(x \cup \{i\}) = f(x) + \phi_i \Rightarrow \arg \max_{i \in \Delta_A^+(x)} f(x \cup \{i\}) = \arg \max_{i \in \Delta_A^+(x)} \phi_i$$

- The last subset visited is the best solution found

Example: the knapsack problem

B	a	b	c	d	e	f
ϕ	7	2	4	5	4	1
v	5	3	2	3	1	1

$$V = 8$$

The algorithm performs the following steps:

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

This algorithm does not find the optimal solution $x^* = \{a, c, e\}$

But why?

Example: the *MDP*

Select from a set of points a subset of k points with the maximum sum of the pairwise distances

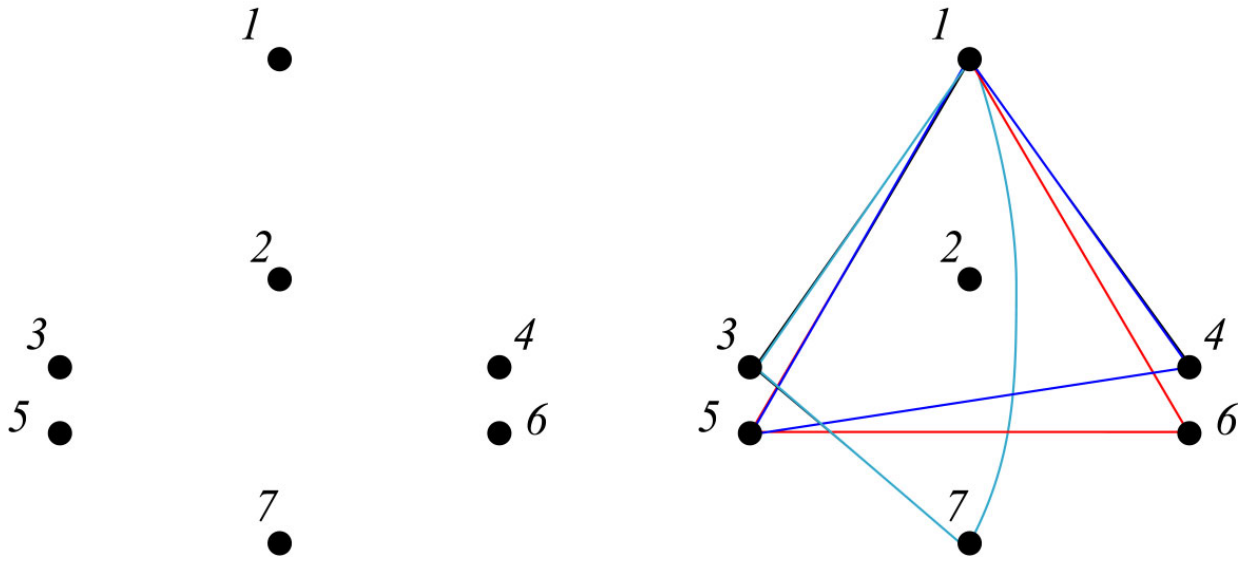
Algorithm GreedyMDP(I)
$x := \emptyset$;
While $|x| < k$ *do*
    $i := \arg\max_{i \in B \setminus x} \sum_{j \in x} d_{ij}$;
    $x := x \cup \{i\}$;
Return $(x, f(x))$;

- Define $\mathcal{F}_A$ as the set of all partial solutions
- The subset x can be extended as long as $|x| < k$
- Any element of $B \setminus x$ extends x in a feasible way
- The objective function is quadratic, and therefore

$$f(x \cup \{i\}) = f(x) + 2\sum_{j \in x} d_{ij} + d_{ii} \Rightarrow \arg\max_{i \in B \setminus x} f(x \cup \{i\}) = \arg\max_{i \in B \setminus x} \sum_{j \in x} d_{ij}$$

- The last subset visited is the best (and only feasible) solution found

Example: the Maximum Diversity Problem



The algorithm has two strong drawbacks

1. at the first step, all points are equivalent ($f(\{i\}) = 0$ for all $i \in B$)
2. the final result is nonoptimal even if
   - the first step selects the pair of farthest points (that is, $(1, 7)$)
   - the algorithm is repeated selecting each point as the first (e.g., 5)

But why?

The Travelling Salesman Problem

Given a directed graph and a cost function defined on the arcs,
find a minimum cost circuit visiting all the nodes of the graph

Define $\mathcal{F}_A$ as the collection of all subsets of arcs that form no subtour and keep a degree ≤ 1 in all nodes, (it is a superset of the partial solutions)

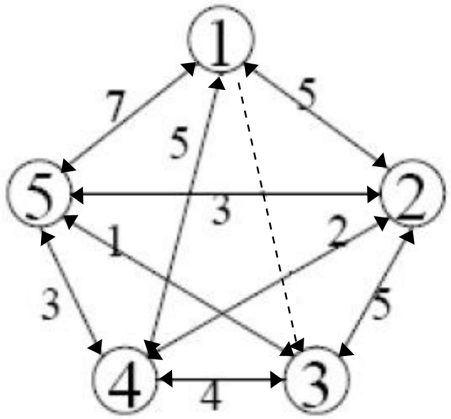
The selection criterium is the objective function
(it is additive, therefore equivalent to the cost of the new arc)

Algorithm GreedyTSP(I)
$x := \emptyset$; $x^* := \emptyset$;
$f^* := +\infty$;
While $\Delta_A^+(x) \neq \emptyset$ *do*
    $i := \arg \min_{i \in \Delta_A^+(x)} c_i$;
    $x := x \cup \{i\}$;
If $x \in X$ *then* $x^* := x$; $f^* := f(x)$;
Return (x^*, f^*);

Only the last subset visited can be feasible (if any!)

Example: the Travelling Salesman Problem

For the sake of simplicity consider a symmetric graph



The algorithm performs the following steps:

1. $x := \emptyset$;
2. since $\Delta_A^+(x) \neq \emptyset$, select $i := (3,5)$ and update x;
3. since $\Delta_A^+(x) \neq \emptyset$, select $i := (2,4)$ and update x ($(5,3) \notin \Delta_A^+(x)$);
4. since $\Delta_A^+(x) \neq \emptyset$, select $i := (5,2)$ and update x ($(4,2) \notin \Delta_A^+(x)$);
5. since $\Delta_A^+(x) \neq \emptyset$, select $i := (4,1)$ and update x:
   notice that $(2,5)$, $(4,5)$, $(5,4)$, $(3,4)$ and $(4,3) \notin \Delta_A^+(x)$!
6. since $\Delta_A^+(x) = \emptyset$, terminate

The algorithm does not find a feasible solution

Adding arc $(1,3)$ with cost 100, it finds a feasible, but nonoptimal, solution

But why?

In summary

A constructive heuristic A finds

- the optimum when $\Delta_A^+\left(x^{(t)}\right)$ and $\varphi_A\left(i, x\right)$ guarantee that the current subset $x^{(t)}$ is always included in at least one optimal solution
- a feasible solution when $\Delta_A^+\left(x^{(t)}\right)$ guarantees that the current subset $x^{(t)}$ is always included in at least one feasible solution
- a general subset when these properties are lost at some step t

An ideal constructive algorithm always keeps one open way to the optimal solution

In practice, some of these properties is usually lost at some step of the algorithm

A characterization in the additive case

Assume that

1. the objective function be additive

$$\exists \phi : B \to \mathbb{N} : f(x) = \sum_{i \in x} \phi_i$$

2. the solutions be the bases (maximal subsets) of the search space

$$X = \mathcal{B}_{\mathcal{F}} = \{Y \in \mathcal{F} : \nexists Y' \in \mathcal{F} : Y \subset Y'\}$$

It is a very frequent case (*KP*, *MAX-SAT*, *TSP*, but not *MDP*, *SCP*)

In this case, the constructive algorithm always finds the optimal solution if and only if $(B, \mathcal{F})$ is a *matroid embedding*

Since the definition of *matroid embedding* is rather complex,
let us focus on some important structures

1. greedoids (necessary condition)
2. matroids or greedoids with the strong exchange property (sufficient)

Greedoids

A greedoid $(B, \mathcal{F})$ with $\mathcal{F} \subseteq 2^B$ is a pair such that

- trivial axiom: $\emptyset \in \mathcal{F}$ *The empty set is acceptable*
- accessibility axiom:
  if $x \in \mathcal{F}$ and $x \neq \emptyset$ then $\exists i \in x : x \setminus \{i\} \in \mathcal{F}$
  Any acceptable subset can be built adding elements in suitable order
- exchange axiom: if $x, y \in \mathcal{F}$ with $|x| = |y| + 1$,
  then $\exists i \in x \setminus y$ such that $y \cup \{i\} \in \mathcal{F}$
  Any acceptable subset can be extended with a suitable element of any other acceptable subset of larger cardinality

The exchange axiom implies that all bases have the same cardinality

All of these conditions

- hold in the fractional *KP*, *MST* problem (both Kruskal and Prim),...
- do not hold in the general *KP*, *TSP*...
- hold in the *MDP*, but the objective function is not additive

Greedoids make greedy algorithms possible, but not necessarily exact

Matroids

A matroid is a set system $(B, \mathcal{F})$ with $\mathcal{F} \subseteq 2^B$ such that

- trivial axiom: $\emptyset \in \mathcal{F}$
- hereditarity axiom: if $x \in \mathcal{F}$ and $y \subset x$ then $y \in \mathcal{F}$
  Any acceptable subset can be built adding its elements in any order
- exchange axiom: if $x, y \in \mathcal{F}$ with $|x| = |y| + 1$,
  then $\exists i \in x \setminus y$ such that $y \cup \{i\} \in \mathcal{F}$
  Any acceptable subset can be extended with a suitable element of any other subset of larger cardinality

The hereditarity axiom is a stronger version of accessibility

- it holds in Kruskal's search space for the *MST* problem
- it does not hold in Prim's search space for the *MST* problem

We already know some examples of matroids

Uniform matroid: fractional and general knapsack

$$\mathcal{F} = \{x \subseteq B : |x| \leq \lfloor V/v \rfloor\}$$

- Trivial axiom: the empty set respects the cardinality constraint
- Hereditarity axiom: if x respects the cardinality constraint, all of its subsets also respect it
- Exchange axiom: if x and y respect the cardinality constraint and $|x| = |y| + 1$, one can always add a suitable element of x to y without violating the cardinality (*in fact, any element of* x)

For the general *KP* the first two axioms hold, but the third one does not

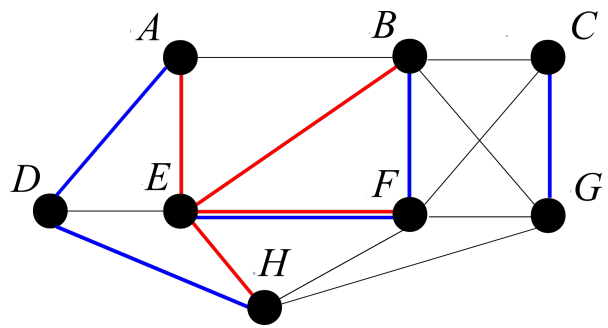
Example:

If $V = 6$ and $v = [\,3\ 3\ 2\ 2\ 1\,]$, the subsets $x = \{3, 4, 5\}$ and $y = \{1, 2\}$ are in $\mathcal{F}$, but no element of x can be added to y

Graphic matroid: minimum spanning tree

$$\mathcal{F} = \{x \subseteq B : x \text{ forms no cycles } \}$$

- Trivial axiom: the empty set of edges forms no cycles
- Hereditarity axiom: if x is acyclic, all of its subsets are acyclic
- Exchange axiom: if x and y are acyclic and $|x| = |y| + 1$, one can always add a suitable edge of x to y without forming any cycle (*not all edges of x work*)



$x = \{(A, D), (D, H), (E, F), (B, F), (C, G)\}$

$y = \{(A, E), (B, E), (E, F), (E, H)\}$

(A, D), (D, H) *and* (C, G) *can be added to* y

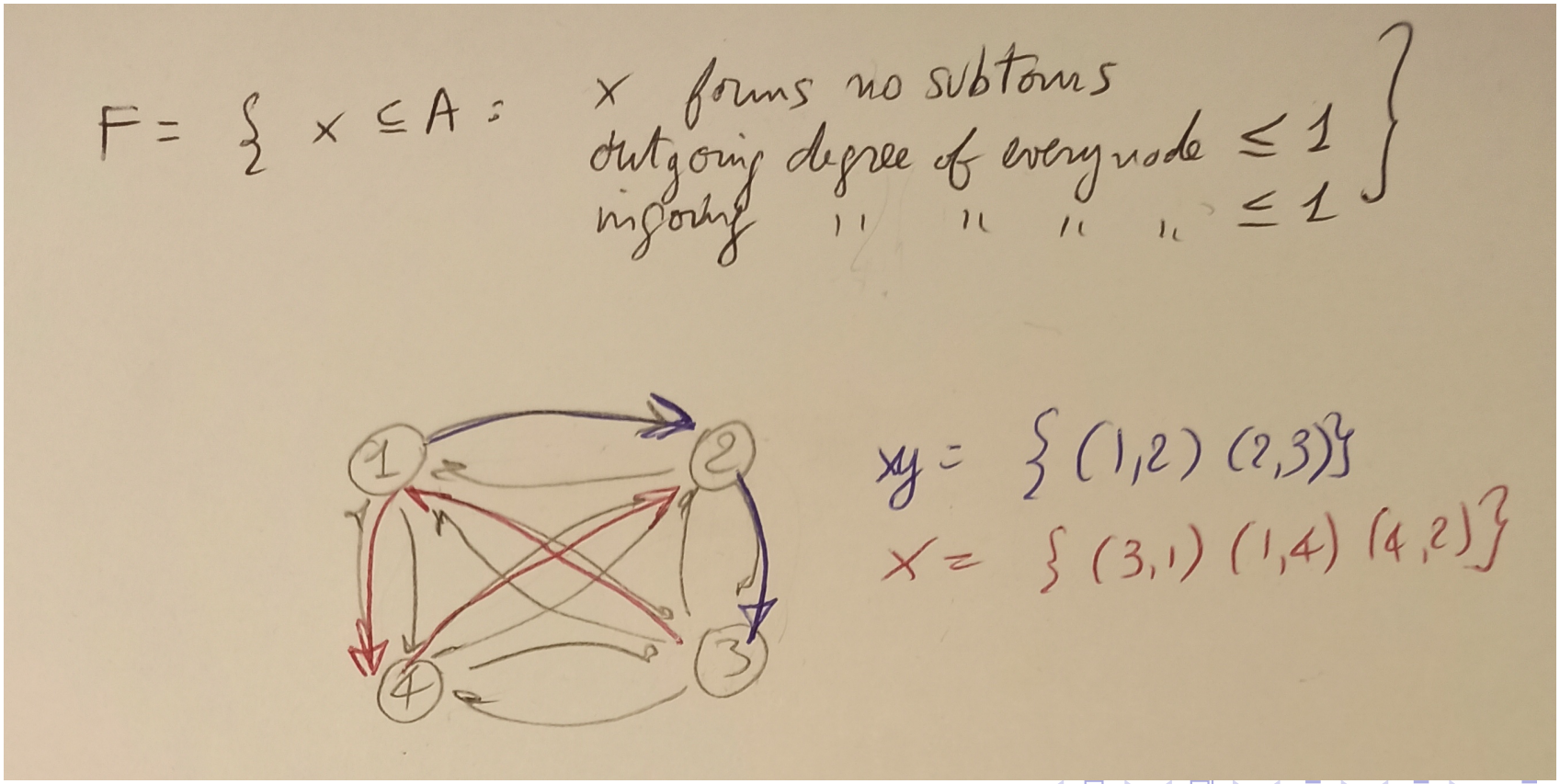
Travelling Salesman Problem

For the *TSP* the first two axioms hold

- the empty set has no subtours and degrees ≤ 1
- any proper subset of a set $\in \mathcal{F}$ (no subtours and degrees ≤ 1) also belongs to $\mathcal{F}$

but the third axiom is violated

Example: $y = \{(1,2),(2,3)\}$ and $x = \{(3,1),(1,4),(4,2)\}$



No arc of x can be added to y remaining in $\mathcal{F}$

Greedoids with the strong exchange axiom

The optimality of the greedy algorithm can be proved for greedoids (weaker second axiom) if the exchange axiom is strengthened

- strong exchange axiom:

$$\begin{cases} x \in \mathcal{F}, y \in \mathcal{B}_{\mathcal{F}} \text{ such that } x \subseteq y \\ i \in B \setminus y \text{ such that } x \cup \{i\} \in \mathcal{F} \end{cases} \Rightarrow \exists j \in y \setminus x : \begin{cases} x \cup \{j\} \in \mathcal{F} \\ y \cup \{i\} \setminus \{j\} \in \mathcal{F} \end{cases}$$

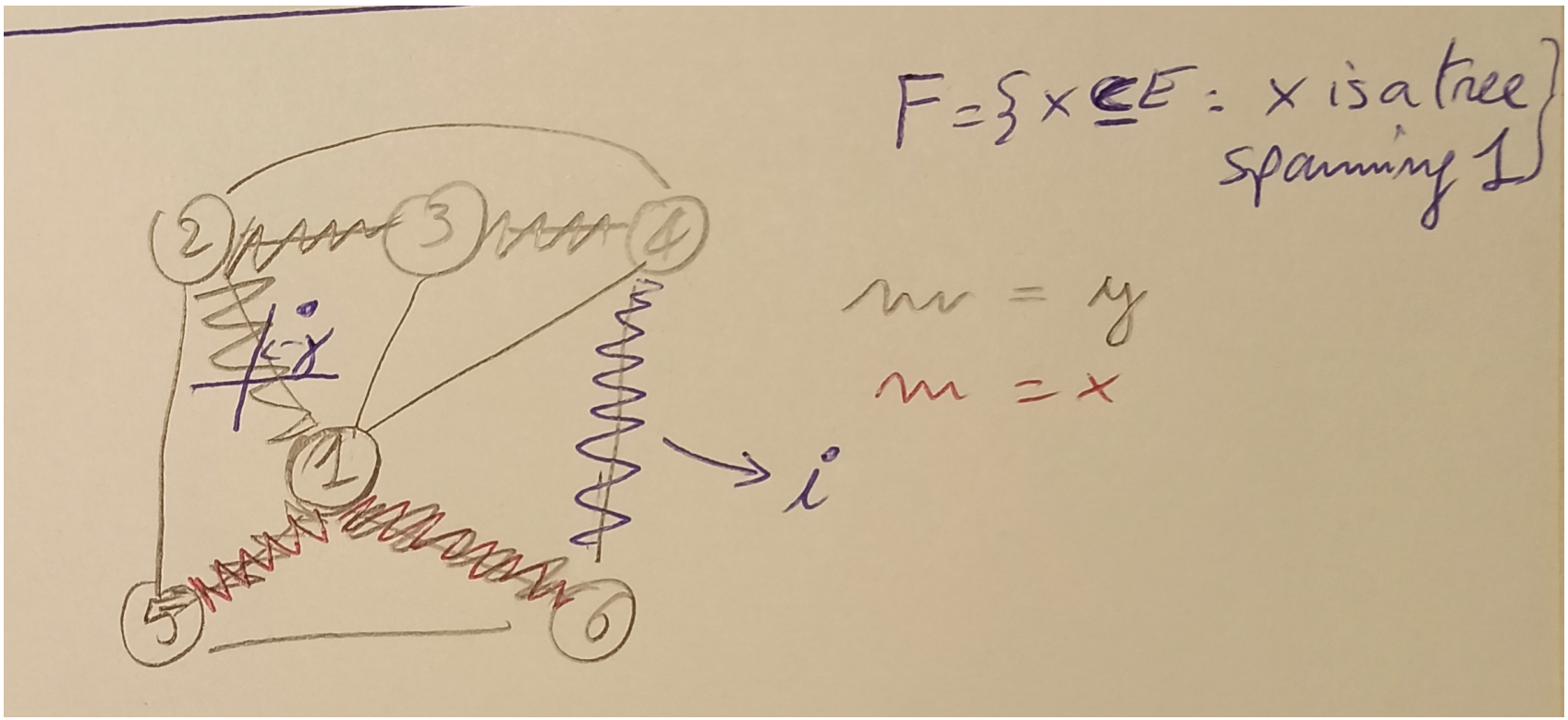
Given a basis and one of its subsets (from which the basis is accessible), if there is an element that "leads astray" the subset from the base, there must be another one which keeps it on the right way and it must be feasible to exchange the two elements in the basis

Greedoids with the strong exchange axiom: *MST*

A classical example of greedoid with strong exchange axiom is given by

- B = edge set of a graph
- $\mathcal{F}$ = collection of the trees including a given vertex v_1

that yields Prim's algorithm for the *MST* problem



The trivial and the accessibility axiom hold (the hereditary one does not)
The exchange axiom holds in the strong form

Optimal constructive algorithms

Notice that the optimality of a constructive algorithm A depends on

- the properties of the problem (e. g., additive objective function, bases as feasible solutions)
- the properties of the search space $\mathcal{F}_A$ (that is, of the algorithm)

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

It is easy to prove that

- the sum of the two solution values overestimates the optimum

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

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

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

The *Nearest Neighbour* heuristic for the *TSP*

Consider the *TSP* on a complete graph $G = (N, A)$ with the usual search space (arc subsets with no subtour and induced degree ≤ 1 on all nodes)

- the constructive algorithm always finds a feasible solution
- the solution can be unlimitedly bad (*Why?*)

Change the search space: $\mathcal{F}_A$ 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

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

As for the axioms

- the trivial axiom always holds
- the accessibility axiom holds: removing the last arc yields a path starting from node 1
- the hereditarity axiom does not hold: not all subsets are paths
- the exchange axiom does not hold (not a greedoid, therefore)

The *Nearest Neighbour* heuristic for the *TSP*

The *Nearest Neighbour* (*NN*) heuristic adopts 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

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

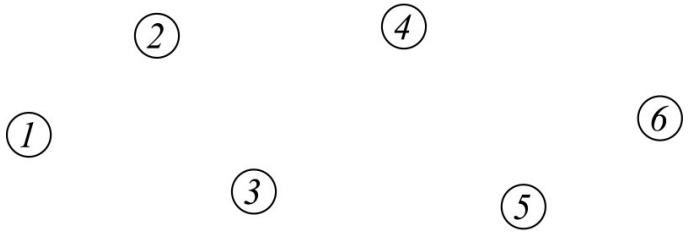
  (*it is a pure constructive algorithm*)
- If $j \neq 1$, go back to point 2; otherwise, terminate
  ($\Delta_A^+(x)$ *allows the return to node* 1 *only at the last step*)

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

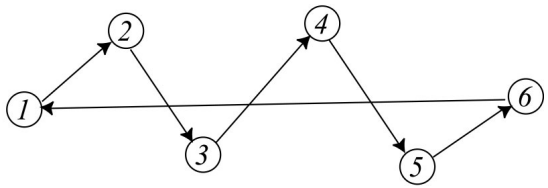
It is not exact, but $\log n$-approximated (*under the triangle inequality*)

The *Nearest Neighbour* heuristic: example

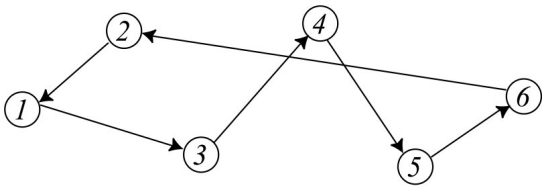
Consider a complete graph (the arcs are not reported for clarity)



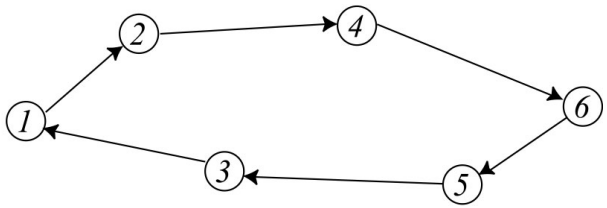
Starting from node 1



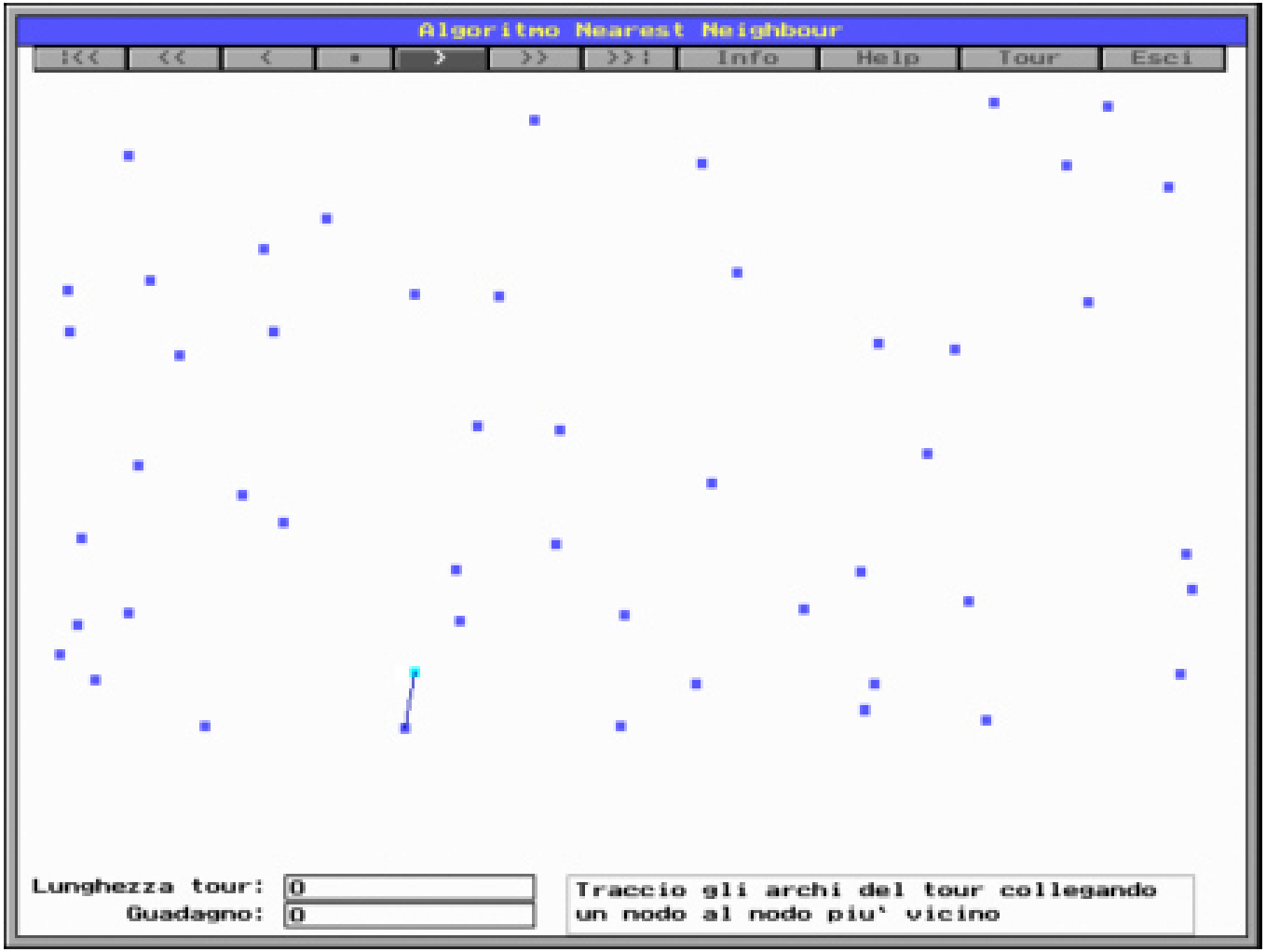
Starting from node 2



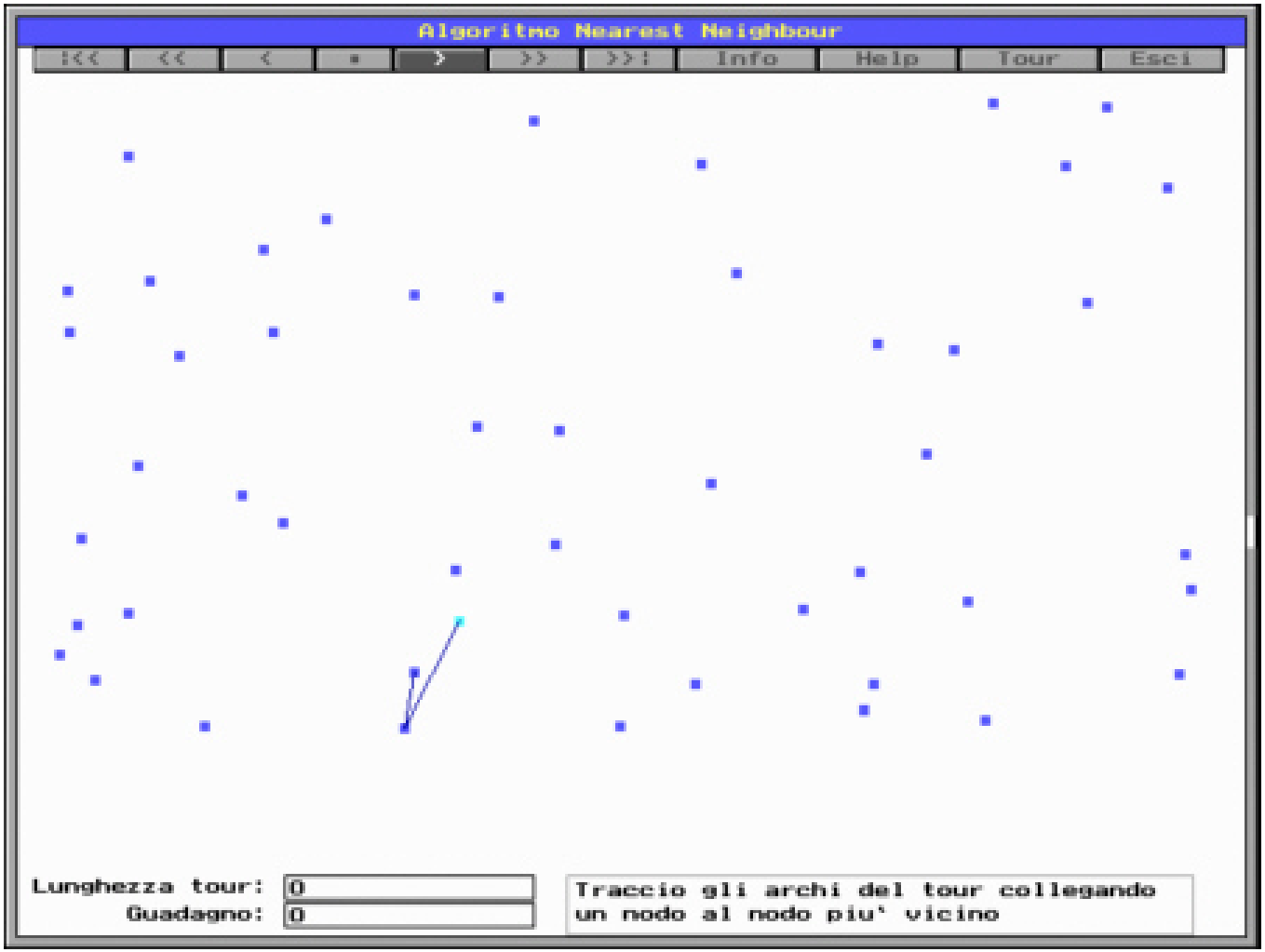
The optimal solution cannot be found starting from any node



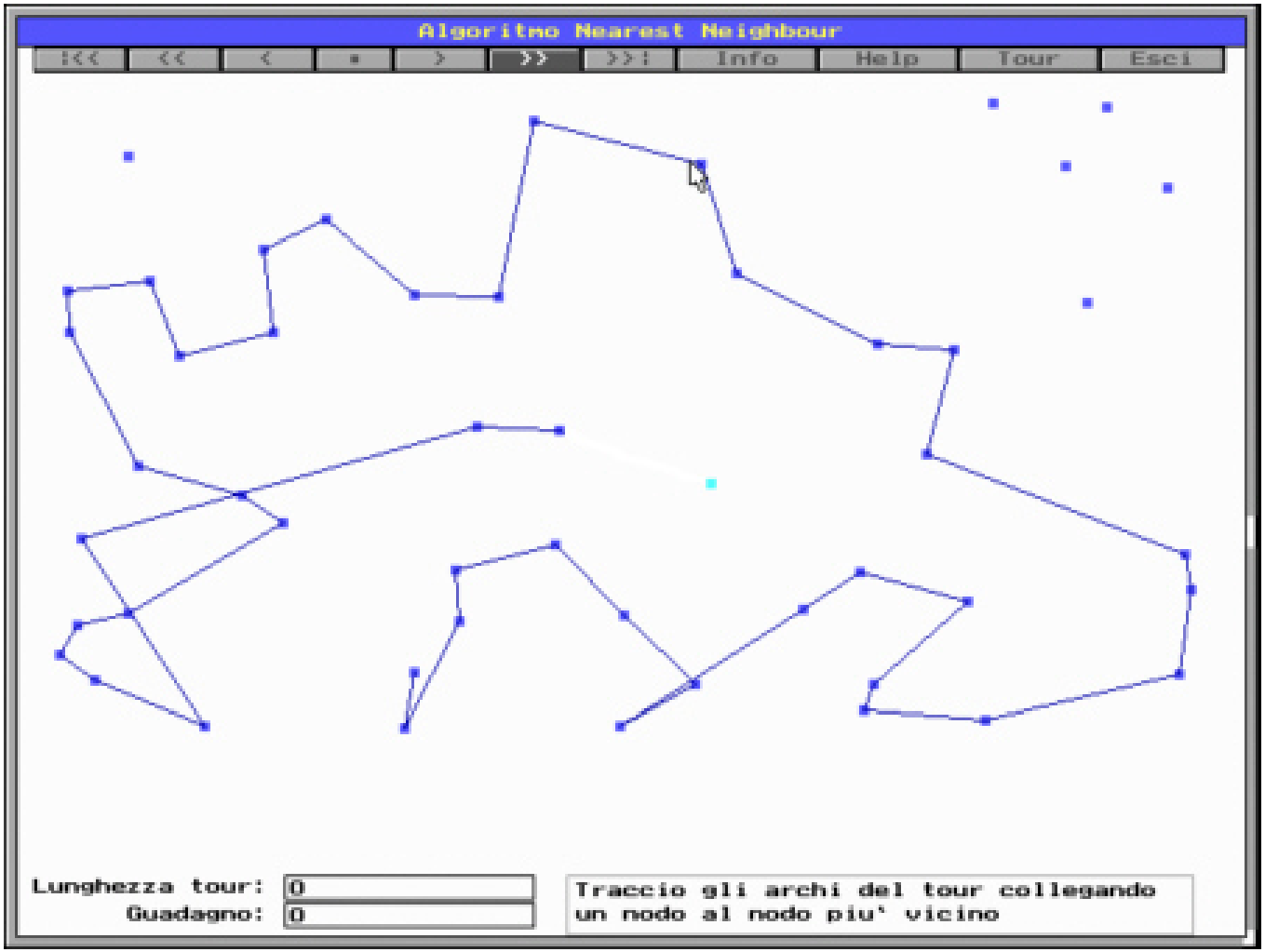
A larger example



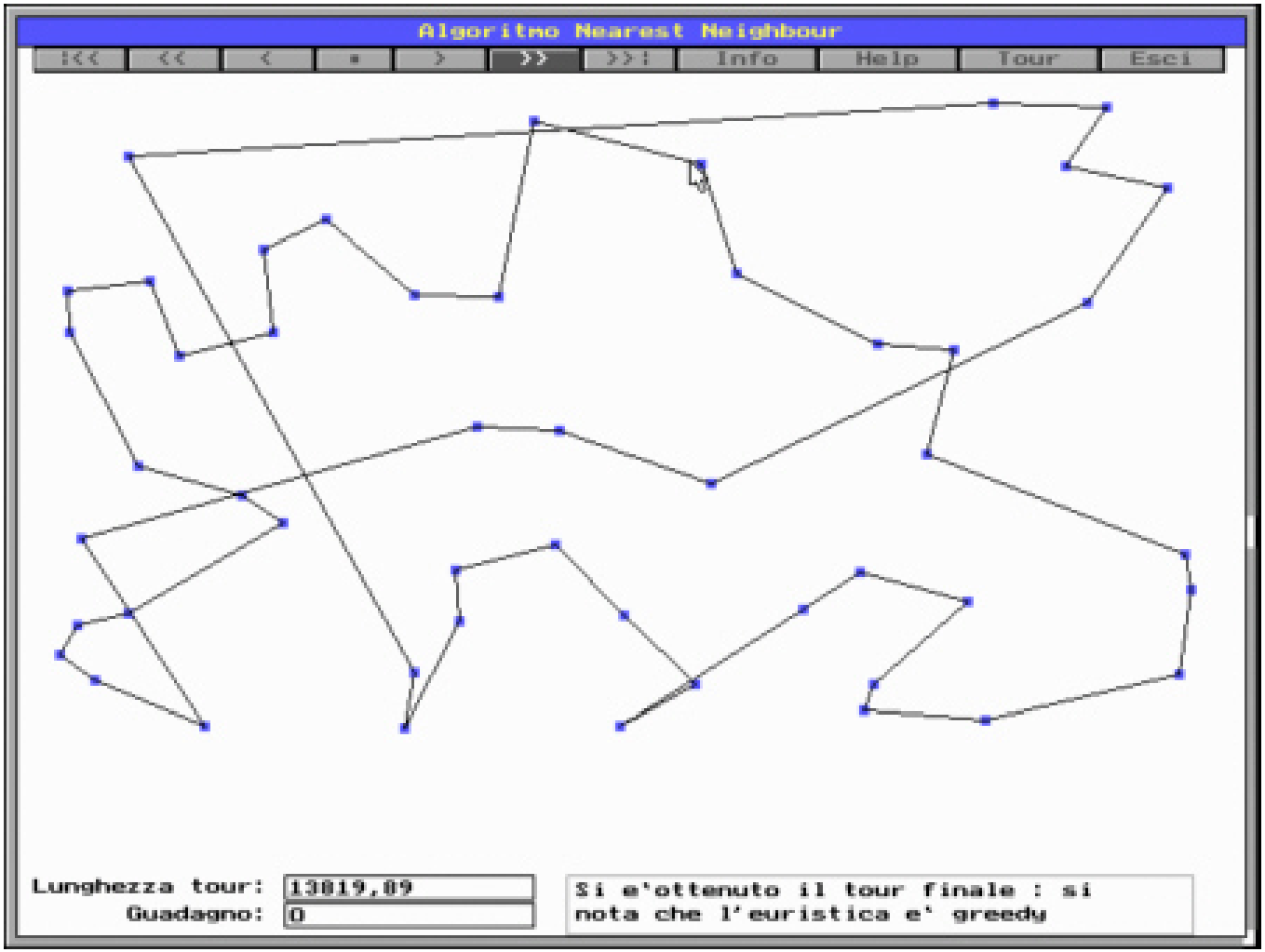
A larger example



A larger example



A larger example



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

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 $\varphi_A(i, x) = \dfrac{c_i}{a_i(x)}$

  where $a_i(x)$ is the number of rows covered by i, but not by x

Set Covering: a positive example

c	3	5	6	2	1	7	1	8

A								
	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

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

$$c \quad \begin{array}{|ccccc|} \hline 25 & 6 & 8 & 24 & 12 \\ \hline \end{array}$$

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

The algorithm performs the following steps:

1. $x := \emptyset$;
2. since $c/a_i(x) = \left[\, 4.1\bar{6} \quad 2 \quad 4 \quad 12 \quad 12 \,\right]$, select $i := 2$;
3. since $c/a_i(x) = \left[\, 8.\bar{3} \quad - \quad 8 \quad 12 \quad 12 \,\right]$, 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

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

- 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

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

  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*

- since the column chosen is always that with minimum $\varphi_A\left(i, x^{(t-1)}\right)$ and $a_i\left(x^{(t-1)}\right)$ decreases, the row weights increase step by step
- at step t, there are $|R^{(t)}| \leq |R| - t$ uncovered rows and the columns of the optimal solution could cover them all with cost f^*
  $\Rightarrow$ at least one of such columns has unitary cost $\leq f^*/|R^{(t)}|$
- each column i selected has minimum unitary cost, therefore not larger than that, and the covered rows assume weight

$$\theta_j \leq \frac{f^*}{|R^{(t_j)}|} \Rightarrow \sum_{j \in R} \theta_j \leq \sum_{j \in R} \frac{f^*}{|R^{(t_j)}|}$$

  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 $\varphi_A(i,x) = \left[\, 4.1\bar{6} \quad 2 \quad 4 \quad 12 \quad 12 \,\right]$, 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 $\varphi_A(i,x) = \left[\, 8.\bar{3} \quad - \quad 8 \quad 12 \quad 12 \,\right]$, 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

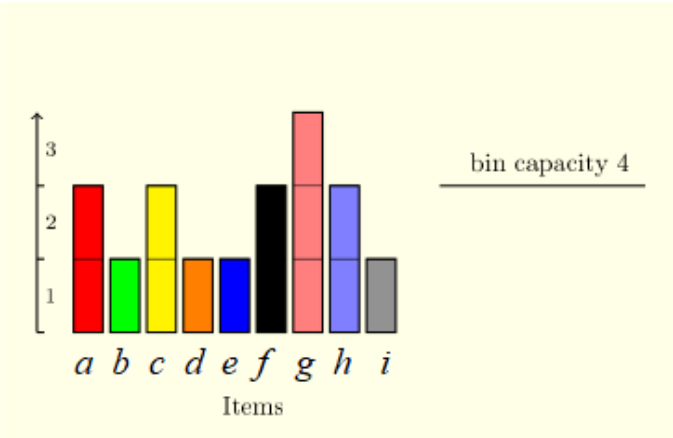
Now $f_A = \sum_{j \in R} \theta_j = 50$ and the approximation holds: $f_A \leq (\ln|R| + 1)\, f^* \approx 2.79\, f^*$

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

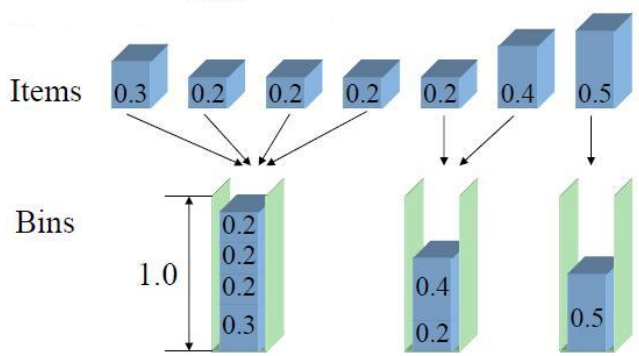
The objective function is a bad selection criterium, because it is flat

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: $x^{(t)} := x^{(t-1)} \cup \{(i, j)\}$

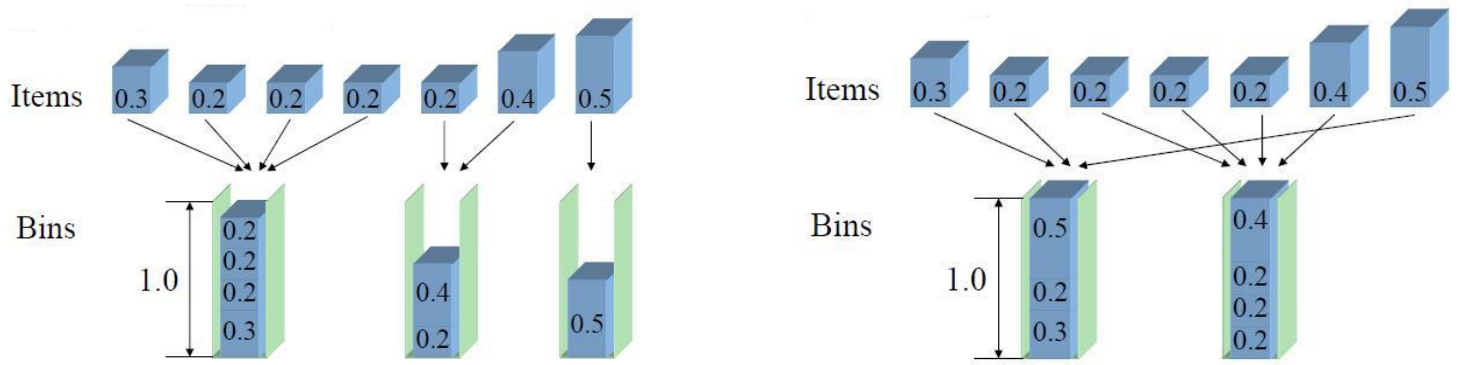


Notice that the choice of (i, j)

- is not determined by $f(x \cup \{(i, j)\})$ (*another i could be better*)
- depends on both i and x (x *determines if* (i, j) *increases* f *or not*)

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

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

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

(*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: $f_A \leq \frac{11}{9} f^* + 1$