MATHEURISTICS
FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Module 1- Lesson 1

Prof. Maurizio Bruglieri
Politecnico di Milano

Agenda

- **Lesson 1 (**Wednesday 17th, 14.30-17):
  Introduction to Combinatorial Optimization and Mathematical Programming.
  Matheuristics: general features and classification.
  Rounding and search around heuristics.

- **Lesson 2** (Friday 19th, 10.30-13):
  Approximated heuristics. Dual heuristics.

- **Lesson 3** (Tuesday 23th, 10.30-13):
  Relaxation techniques. Lagrangean heuristics. Surrogated heuristics.

- **Lesson 4** (Friday 26th, 10.30-13):
  Decomposition based heuristics.

What is a decision problem

Two conditions:

1. there must exist different alternatives or *feasible solutions* for the problem
2. at least one criterion or *objective* is specified allowing to compare the feasible solutions (making some of them better than others)

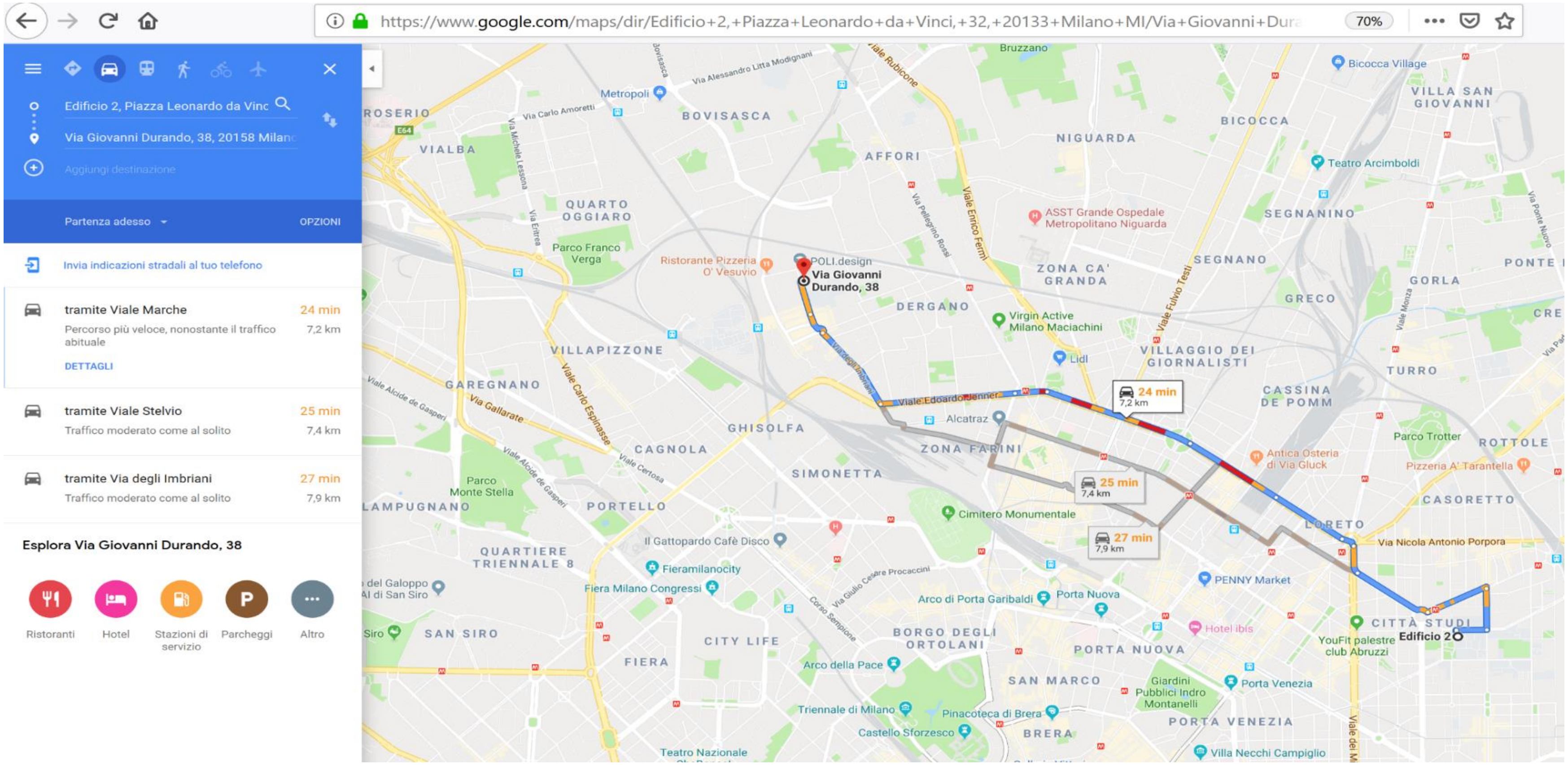
Combinatorial optimization problems

- A Combinatorial Optimization Problem (COP) is a decision problem with a finite (but exponential) number of feasible solutions

- A COP is characterized by:
  1. a description of all its *input parameters* (e.g. costs, demand, capacity)
  2. a statement of which properties a *feasible solution* must satisfy
  3. an *objective* to be either minimized or maximized

- An *instance* of a COP is obtained each time the input parameters are specified (i.e. their numerical values are fixed)

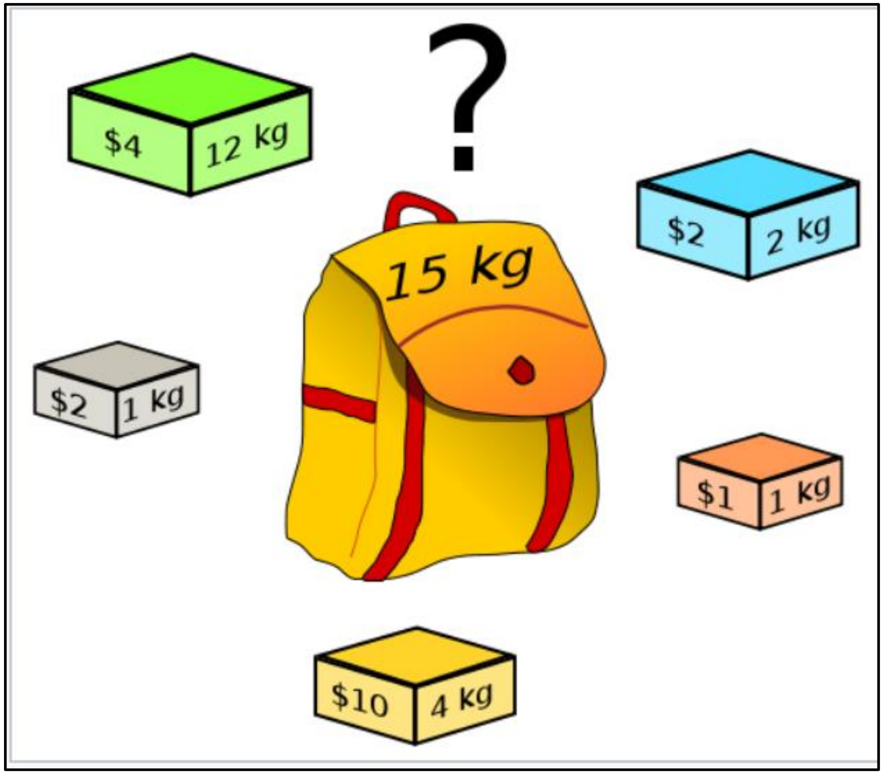
Example 1: shortest path problem



Given a directed graph $G=(N,A)$, with costs c_{ij} for each $(i,j) \in A$, and two nodes $o, d \in N$, determine a path from o to d such that the sum of the costs of its arcs is minimal.

Example 2: knapsack problem

Given a knapsack with capacity C and n items with profit p_i and size w_i, for $i = 1, \ldots, n$, determine a subset of items such that their total profit is as large as possible and their total size does not exceed C



Example 3: Traveling Salesman Problem (TSP)

Given a directed graph *G*=(*N*,*A*), with costs c_{ij} for each $(i,j) \in A$, determine a cycle visiting all nodes such that the sum of the costs of its arcs is minimal.

Minimum cost tour of the 50 USA landmarks
(from http://www.math.uwaterloo.ca/tsp/usa50/index.html)

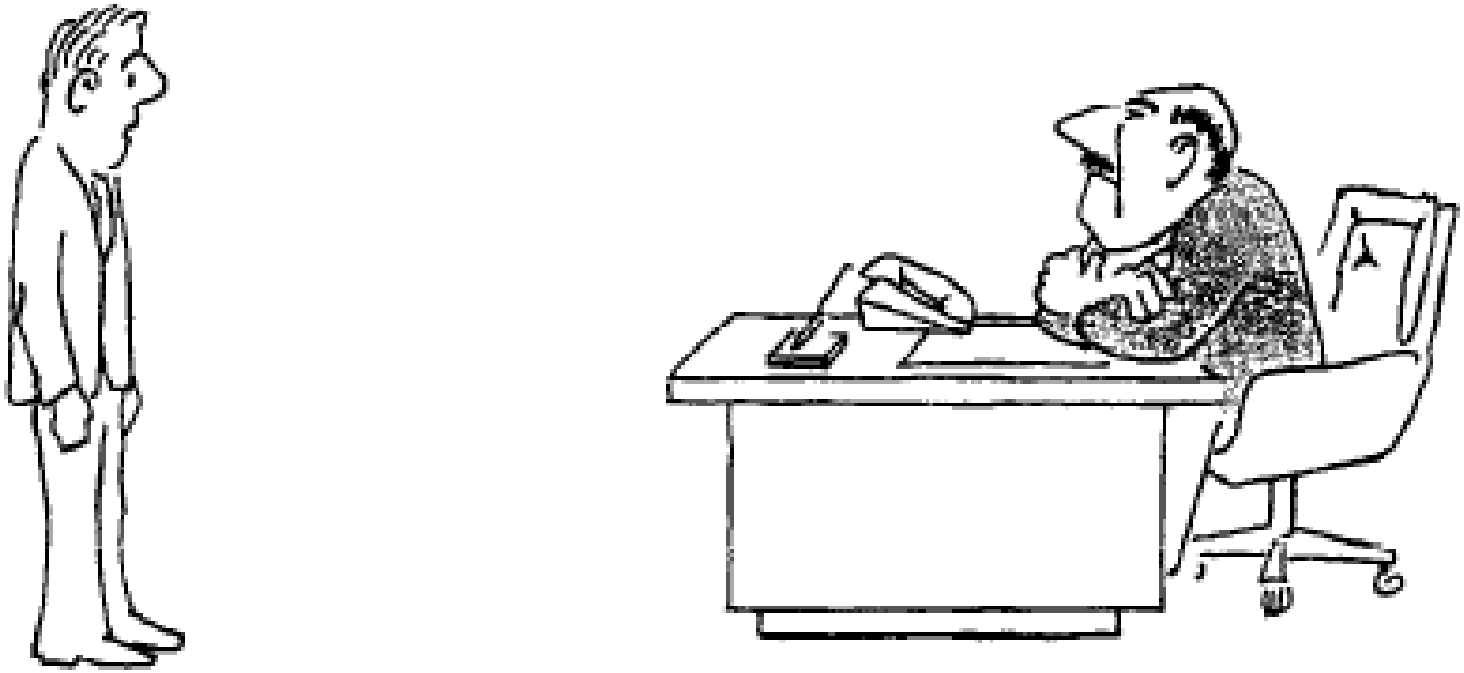
Computational complexity of COPs

- The shortest path problem can be solved in polynomial time (e.g. in $O(n^2)$ by Dijkstra's algorithm, if $c_{ij} \geq 0$, otherwise in $O(n^3)$ by Floyd-Warshall)

- Knapsack can be solved in pseudo-polynomial time ($O(nC)$ by dynamic progr.)

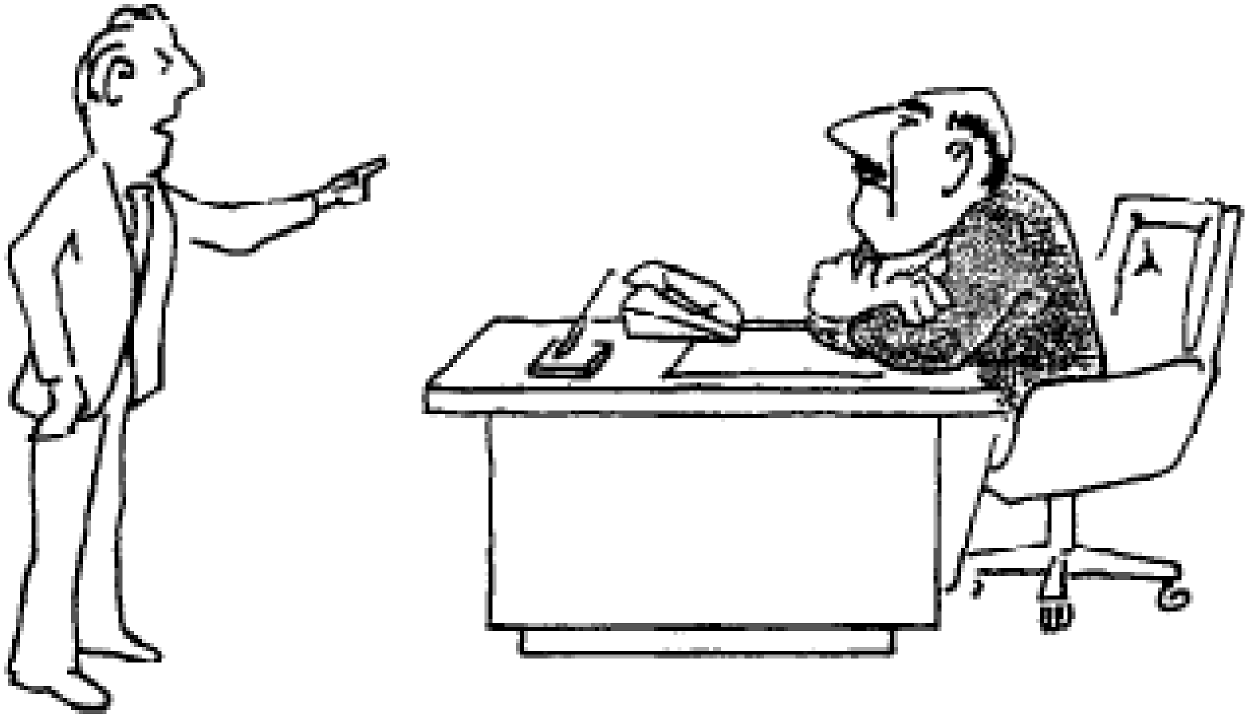
- For the TSP no pseudo-polynomial time algorithm: it is **strongly NP-hard**

From Garey-Johnson's book



«I can't find an efficient algorithm, I guess I'm just too dumb»

From Garey-Johnson's book



«I can't find an efficient algorithm, because no such algorithm is possible!»

From Garey-Johnson's book

«I can't find an efficient algorithm, but neither can all these famous people»

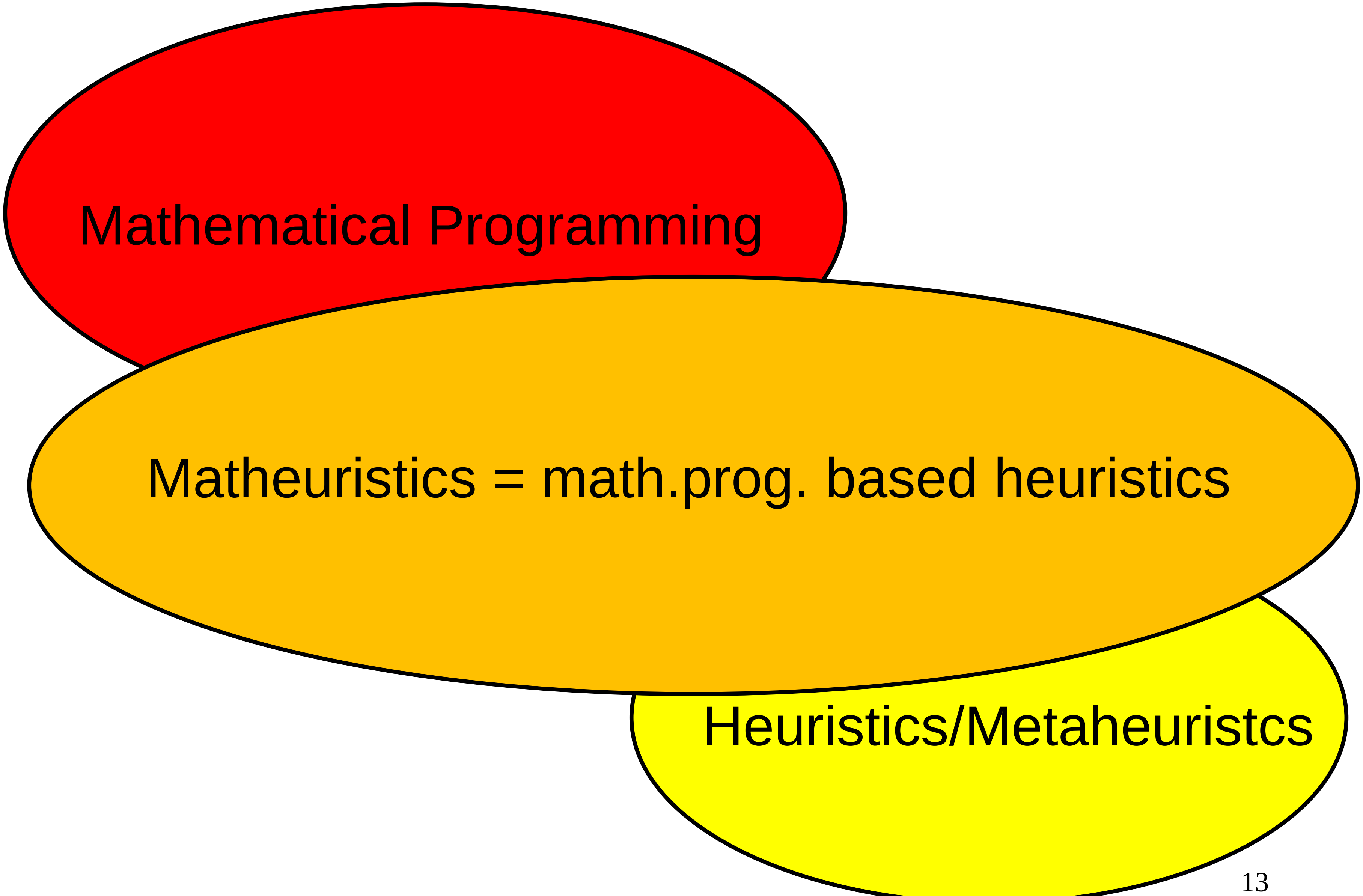
How to prove a COP is NP-hard ?

- Given a COP P_1 we can prove its NP-hardness building a polynomial time ***reduction*** from another well-known NP-hard COP P_2 to P_1

- Whenever a COP includes as particular case another well-known NP-hard problem it is NP-hard too (e.g. the VRP is NP-hard since the TSP is so)

- Online compendium of NP-hard problems:
  https://www.nada.kth.se/~viggo/problemlist/compendium.html

What are matheuristics?



Mathematical Programming formulation

- A *Mathematical Program* is characterized by:
  1. Decision variables
  2. One objective function
  3. Constraints

- General form of a Mathematical Program:

$$\min f(x)$$
$$x \in X$$

$X \subseteq \Re^n$ feasible solution set

$f: X \rightarrow \Re$ objective function

- A Mathematical Program is a Linear Program if the objective function and the constraints are linear function of the decision variables

- An Integer Linear Program (ILP) is a Linear Program with integer variables

- Most NP-hard COPs can be modeled as ILP (never as a compact LP)

Formulation of a knapsack problem

We want to realize a song compilation collecting in a CD with 800 Mb of capacity some music files. The level of appreciation of each song (in a scale from 1 to 10) and the size of each file are reported in the following table:

Song	**Appreciation**	**Size (MB)**
Light my fire	8	210
Fame	7	190
I will survive	8.5	235
Imagine	9	250
Let it be	7.5	200
I feel good	8	220

Parameters: n= # songs, g_i = appreciation of song *i*, w_i = size of *i* , *C* = CD capacity
Decision variables: x_i= 1 if file *i* is selected for the CD, 0 otherwise

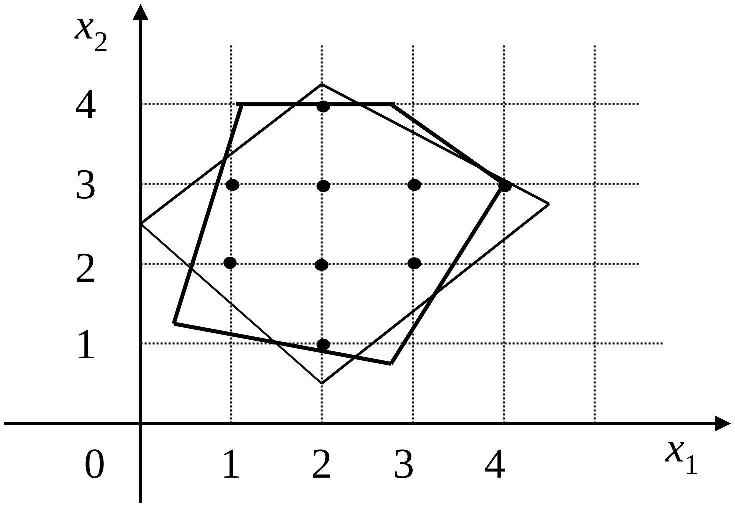
$$\max z = \sum_{i=1}^{n} g_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leq C$$

$$x_i \in \{0,1\} \text{ for } i = 1, .., n$$

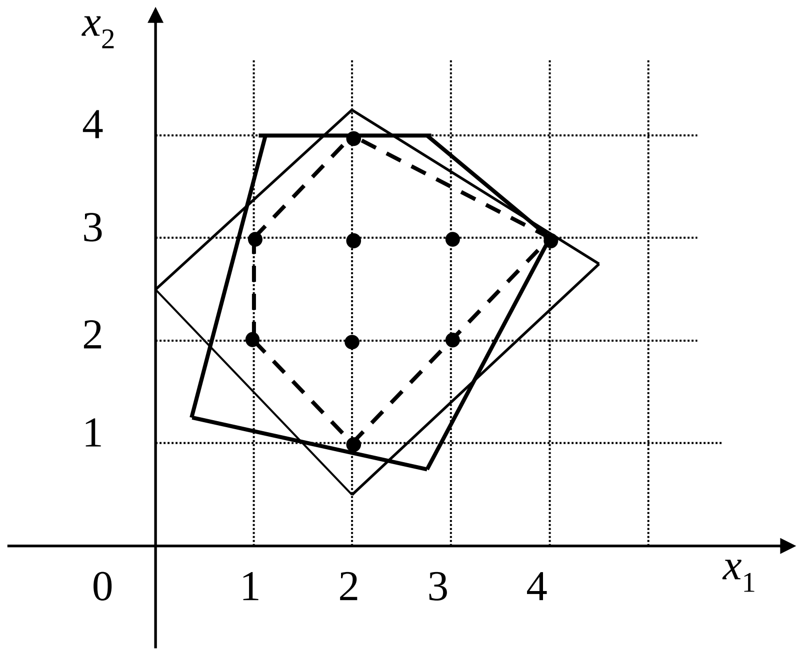
Mathematical Programming formulation

- The feasible region of an ILP is a subset $S \subset \mathbb{Z}^n$
- A ***formulation*** of $S \subset \mathbb{Z}^n$ is a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ such that $P \cap \mathbb{Z}^n = S$



Mathematical Programming formulation

- Given two formulations P_1 and P_2 of $S \subset \mathbb{Z}^n$, P_1 is better than P_2 if $P_1 \subset P_2$



- conv(S)is the *ideal formulation*

since $\begin{array}{c} min\, f(x) \\ s.t.\ x \in S \end{array} \Leftrightarrow \begin{array}{c} min\, f(x) \\ s.t.\ x \in conv(S) \end{array}$

Heuristics algorithms

• **Motivation**: finding the optimal solution of a NP-hard problem is computationally too heavy in practical cases (e.g. for big size instances)

Heuristic algorithm (*from greek word eureka=discover****)***: method providing a feasible solution, non necessarily optimal, for a problem

• Heuristics *with approximation guarantee*

evaluation in the **worst case** (and sometimes in the *average case*, less frequent)

• Heuristics *without approximation guarantee*

Heuristics algorithms classification

- Constructive heuristics:
  - Greedy algorithm
  - Regret algorithm
  - Savings algorithm
- Local search

Greedy algorithms

<u>Main idea</u>: the solution is built step by step and at each step the more advantageous choice, compatible with the constraints, is made

greedy (input: E; output: S);
begin $S := \emptyset$;
  while $E \neq \emptyset$ **do**
    e:=element of E providing the best value of $S \cup \{e\}$;
    $E := E - \{e\}$;
    if $S \cup \{e\}$ is feasible **then** $S := S \cup \{e\}$;
  endwhile
  return S;
end;

these steps have to be efficient

Example: The Traveling Salesman Problem (TSP)

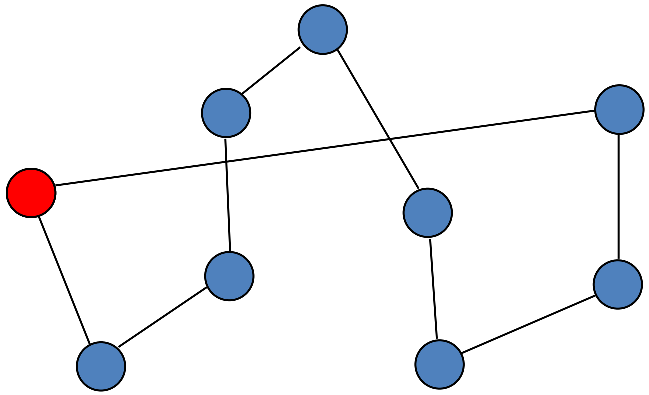
Nearest Neighbourhood heuristic

1. Choose a starting node *p* and mark it

2. Repeat (*n*–1) times:

    link last marked node with the nearest not marked node

3. Link the last marked node with node *p*



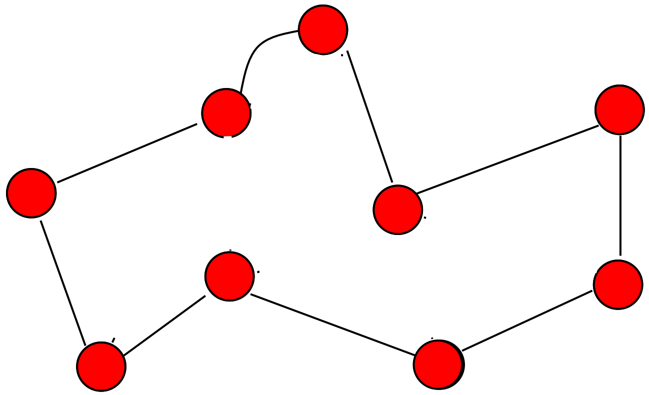
Example: The Traveling Salesman Problem (TSP)

Nearest Insertion heuristic

1. Choose two nodes and build a partial cycle between them

2. repeat (n–2) times:

   insert in the partial cycle the nearest node to one of the nodes in the cycle



Regret algorithms

Example: We want to decide how to assign *n* tasks to *n* employees, on the basis of the following estimation of the times necessary to each employee to perform each task

	Task 1	Task 2	Task 3	Task 4	Task 5
Empl. 1	15	22	18	35	27
Empl. 2	10	15	15	20	22
Empl. 3	12	30	13	25	23
Empl. 4	18	24	19	27	29
Empl. 5	13	23	14	32	25

Regret of each task: difference in absolute value between the minimum time and the second one

Saving algorithms

Example: Clarke and Wright' algorithm for capacitated Vehicle Routing Problem (VRP)

1. compute the savings for each arc (i,j): $s_{ij} = c_{i0} + c_{0j} - c_{ij}$
2. Order the edges for non-increasing values of their savings
3. Add edges to routes until the capacity constraint cannot be satisfied

Local search algorithm

Objective: find (quickly) good feasible solutions for NP-hard problems

Idea: improve iteratively the solutions obtained through heuristic algorithms (e.g., with *greedy* algorithms)

Technique:

(a) Detect small changes (perturbations) in the structure of the feasible solutions preserving feasibility

(b) Apply the changes until the objective function value can improve

Local search algorithm

Consider the generic problem

$$\min_{s \in S} f(s)$$

where S rapresents the feasible solutions set

and $f(s)$ is the objective function to be optimized

Definition

We call *move m* from a solution to another one an operator

$$m: S \longrightarrow S$$

Given a feasible solution $s \in S$, operator move m applied to s returns a feasible solution $m(s) \in S$

Neighborhood

In general we use moves that do not modify too much the structure of a solution

i.e., we prefer $m(s)$ *near* to solution s

Definition

We call *neighborhood* of a solution s the set

$$N(s) = \{\overline{s} \in S \,|\, \exists m : \overline{s} = m(s)\}$$

$N(s)$ is the set of all feasible solutions that it is possible to obtain applying to s the *move* m in all possible ways.

Example:

consider the sequence s=(c,a,d,e,b) corresponding to the solution of an ordering problem (e.g. of objects).

Consider the move that changes the position of a pair of objects.

The neighborhood of s is

$N(s)$={(a,c,d,e,b); (d,a,c,e,b); (e,a,d,c,b); (b,a,d,e,c);

(c,d,a,e,b); (c,e,d,a,b); (c,b,d,e,a);

(c,a,e,d,b); (c,a,b,e,d);

(c,a,d,b,e) }

Alternatively the neighborhood can be defined through a ***distance*** between solutions in *S*

Defintion

We call *l-neighborhood* of a solution *s*

$$N_l(s) = \{\bar{s} \in S \,|\, d(\bar{s}, s) \le l\}$$

where $d : S \times S \to \Re^+$ defines a measure of *distance* in *S*

$N_l(s)$ is the set of all the feasible solutions that are at distance at most *l* from *s*

When the feasible solutions of a problem can be represented through a boolean vector the Hamming distance can be used

Definition

The Hamming distance $d_H(s_1,s_2)$ between two boolean vectors s_1 and s_2 is the number of components where the vectors are different

Example: $s_1=(0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1)$ and

$s_2=(1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1)$

have Hamming distance $d_H(s_1,s_2)$ equal to 4

Consider a feasible solution s=(1,0,1,1) of a knapsack problem with 4 objects

the neighborhood $N_1(s) = \{\bar{s} \in S \mid d_H(\bar{s}, s) \le 1\}$
consists of all the feasible solutions that differ from s in at most one component

(0111)	~~(1111)~~	(0001)	(0000)
(0011)	**(1011)**	(1010)	(0110)
(1110)	(1001)	(1000)	(0101)
(1100)	(1101)	(0010)	(0100)

Consider a feasible solution s=(1,0,1,1) of a knapsack problem with 4 objects

the neighborhood $N_2(s) = \{\bar{s} \in S \mid d_H(\bar{s}, s) \leq 2\}$
consists of all the feasible solutions that differ from s in at most two components

(0111)	~~(1111)~~	(0001)	(0000)
(0011)	**(1011)**	(1010)	(0110)
~~(1110)~~	(1001)	(1000)	(0101)
(1100)	~~(1101)~~	(0010)	(0100)

Local search algorithm

```
Local_Search(s);
begin    s*:=s; END=false;
         repeat
               s:=best solution in N(s)
               if   f(s) < f(s*) then
                    s*:= s;
               else END=true;
         until  not END;
         return s*;
end;
```

Minimum problem

Local Optima

Definition

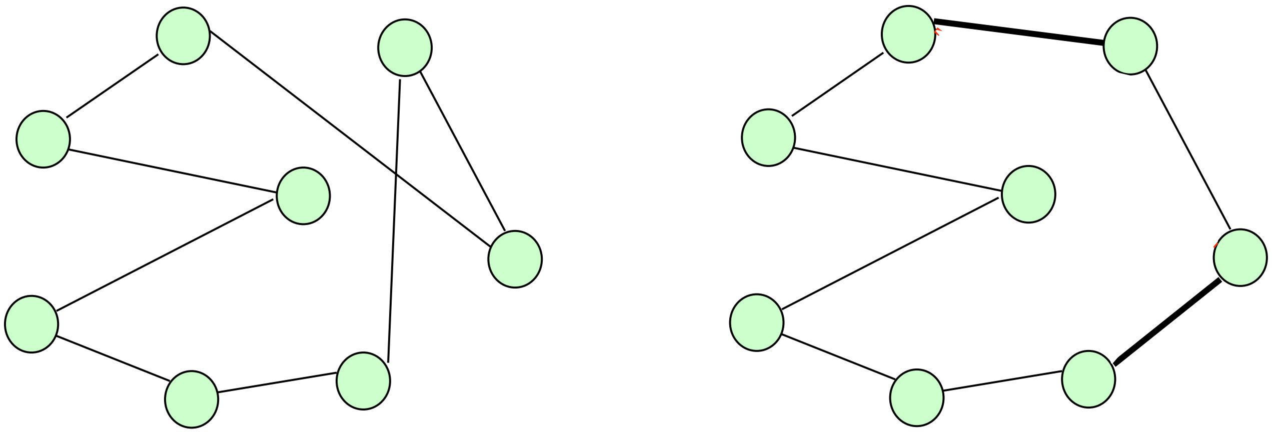
Solution $s \in S$ is called ***local optimum***, respect neighborhood $N(s)$, if the following relation holds $f(s) \leq f(\overline{s}), \forall \overline{s} \in N(s)$

The solutions detected by a local search algorithm are local optima respect the neighborhood used

Neighborhood example

Symmetric Traveling Salesman Problem (TSP)

2-opt neighborhood

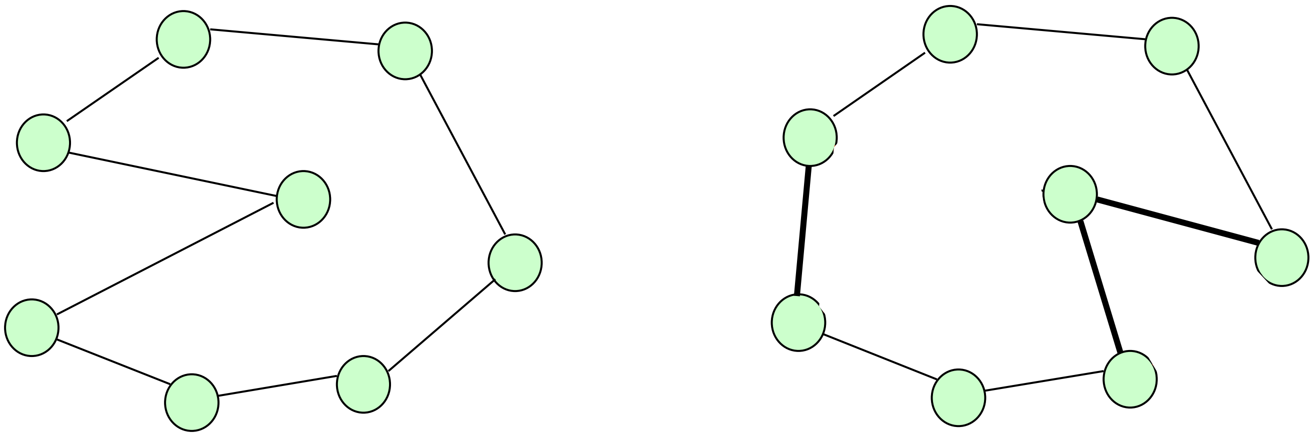


The *move* is the swap of a pair of edges with onother one

Note: after the removal of a pair of edges the choice of the edges to insert is unique

Symmetric Traveling Salesman Problem (TSP)

3-opt *neighborhood*



The *move* is the swap of three edeges with other three ones

Note: after the removal of the three edges, the choice of the new three edges is not unique: how many alternatives are there ?

Symmetric Traveling Salesman Problem (TSP)

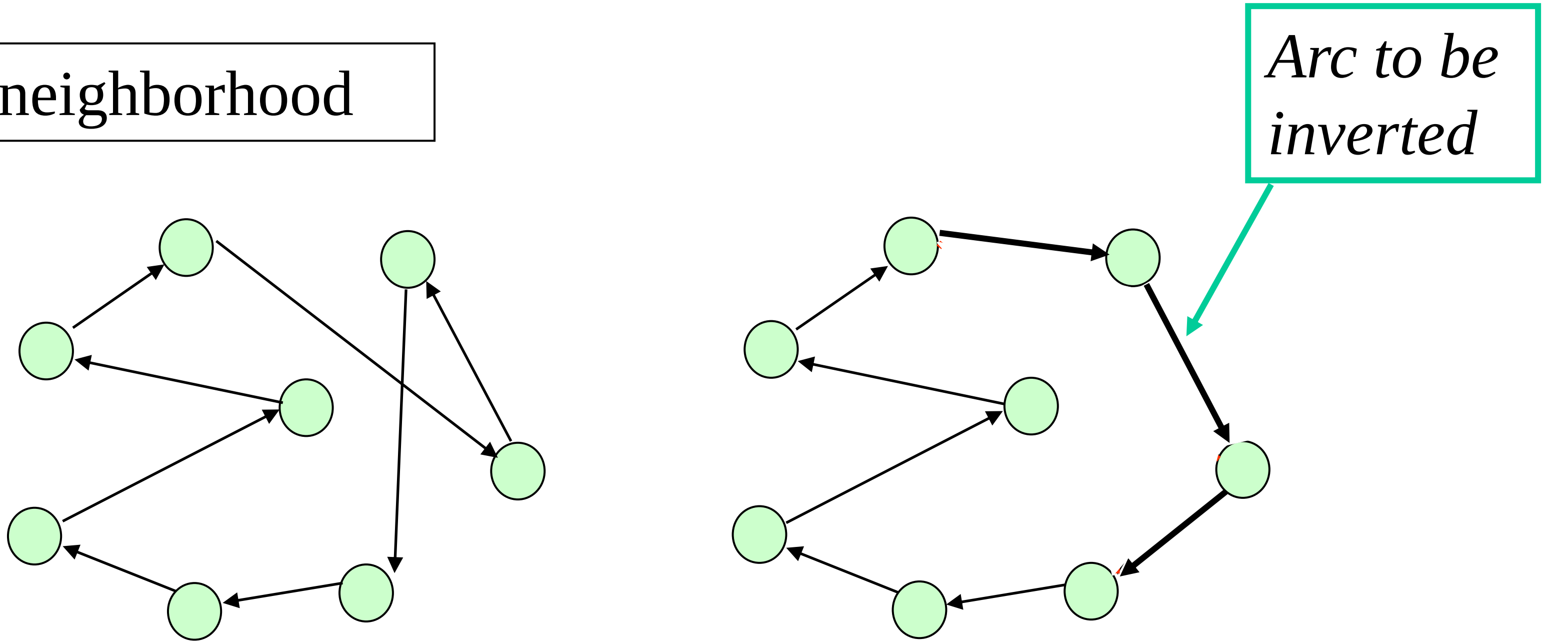
Remarks:

The 2-opt neighborhood is composed of $O(n^2)$ feasible solutions, one for each pair of removed edges.

The 3-opt neighborhood generates, on average, solutions of cost lower than those of the 2-opt neighborhood, but with a greater computational cost: it includes $O(n^3)$ feasible solution, three for each three edges removed

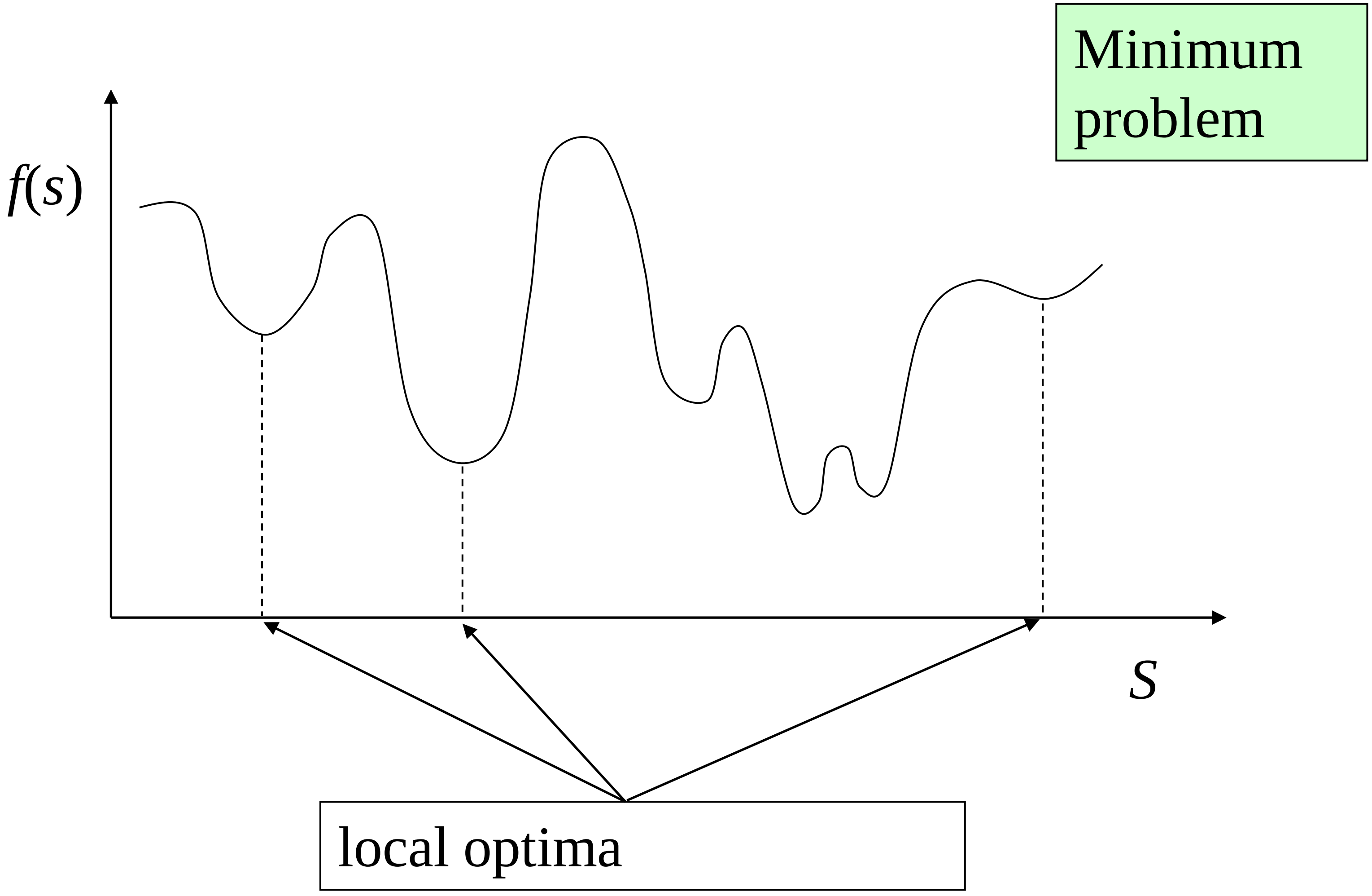
Asymmetric Traveling Salesman Problem (TSP)

2-opt neighborhood



The *move* consists in the ***swap*** of a pair of arcs with another pair and the ***inversion*** of the arcs of a part of the current path

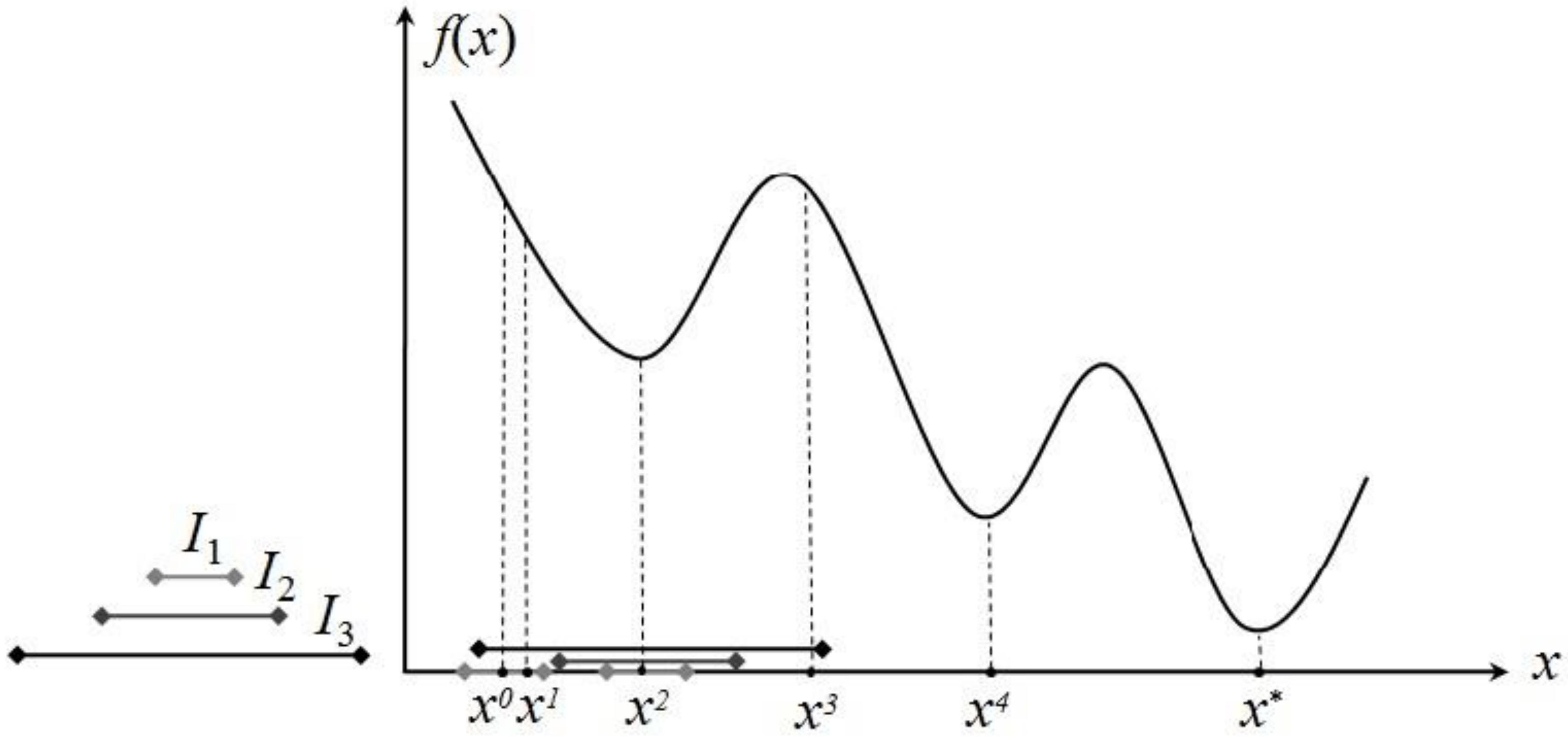
Metaheuristics algorithms



Variable Neighborhood Search (VNS)

- Developed in‘97 by P. Hansen and N. Mladenovic
- **Idea**: a solution that is a local optimum for a neighborhood could be not a local optimum for another neighborhood
- Family of neighborhoods I_k, with $k=1,\ldots,k_{max}$
- $I_k \subseteq I_{k+1}$

Variable Neighborhood Search (VNS)



Tabu Search (Glover 1990)

Primary Features of Tabu Search:

Adaptive memory - remembers features of good/bad solutions that you encounter).

Responsive exploration – exploration based on past exploration.

Tabu Search

Basic Algorithmic Features:

- Always move to the best available neighborhood solution, even if it is worse than the current solution.
- **Tabu list**: maintain a list of solution points that must be avoided (not allowed) or a list of move features that are not allowed.
- Update the tabu list based on some memory structure (short-term memory). Remove tabu moves after some time period has elapsed (*tenure*).
- Allow for exceptions from the tabu list (*aspiration criteria*).
- Expand the search area, modify tenure or size of tabu list.

Tabu Search pseudocode

Algorithm *Tabu search* (S, c, x^*):

```
begin
Let x' = Feasible(S);
Let x* = x';
Let TL={ x' };
Let k = 0,  stop = False;
while (stop = False) do
        while (k < max_no_improvement) do
                Let x' = arg min{c(x) : x ∈ I(x') \ TL}
                if c(x') < c(x*) then do
                        let x* = x';
                        let k = 0;
                else k=k+1;
                update (TL);
        end while
        diversification and/or intensification;
        if stop criterion is satisfied then stop = True
end while
end
```

Example: Tabu Search applied to TSP

$$C = \begin{pmatrix} 0 & 10 & 3 & 7 & 5 \\ 10 & 0 & 8 & 6 & 2 \\ 3 & 8 & 0 & 4 & 3 \\ 7 & 6 & 4 & 0 & 9 \\ 5 & 2 & 3 & 9 & 0 \end{pmatrix}$$

Starting solution by nearest neighborhood: 1, 3, 5, 2, 4, 1, cost 21

Its 2-opt neighborhood is:

Solution	***z***
1,3,5,2,4,1	21
1,5,3,2,4,1	29
1,2,5,3,4,1	26
1,3,2,5,4,1	29
1,3,4,2,5,1	20
1,2,4,5,3,1	31

Best solution 1, 3, 4, 2, 5, 1, cost 20

Example: Tabu Search applied to TSP

The 2-opt neighborhood of 1, 3, 4, 2, 5, 1 is:

Solution	***z***
1,3,4,2,5,1	20
1,3,5,2,4,1	21
1,3,2,4,5,1	31
1,4,3,2,5,1	26
1,2,4,3,5,1	28
1,3,4,5,2,1	28

Therefore 1, 3, 4, 2, 5, 1 is a local optimum

Instead TS selects 1, 4, 3, 2, 5, 1 although its cost (26) worsens the current solution

Example: Tabu Search applied to TSP

The 2-opt neighborhood of 1, 4, 3, 2, 5, 1 is:

Solution	***z***	
1,4,3,2,5,1	26	tabu
1,3,4,2,5,1	20	tabu
1,2,3,4,5,1	36	
1,4,3,5,2,1	26	
1,4,2,3,5,1	29	
1,4,5,2,3,1	29	

Therefore TS selects 1, 4, 3, 5, 2, 1 although it is not improving

Example: Tabu Search applied to TSP

The 2-opt neighborhood of 1, 4, 3, 5, 2, 1 is:

Solution	***z***	
1,4,3,5,2,1	26	tabu
1,4,3,2,5,1	26	tabu
1,3,4,5,2,1	28	
1,5,3,4,2,1	28	
1,4,5,3,2,1	37	
1,4,2,5,3,1	21	

Now TS selects 1, 4, 2, 5, 3, 1 that is improving!

Several metaheuristics

- Adaptive Large Neighborhood Search
- Greedy Adaptive Search Procedure (GRASP)
- Simulated Annealing
- Ant Colony Optimization
- Bee Colony Optimization
- Genetic Algorithms
- Memetic Algorithms
- ….
- Applications of Metaheuristic are almost uncountable and appear in many journals (e.g. «Journal of Heuristics») and specialized conferences e.g. Metaheuristics International Conference (MIC)
- Their success is due to the fact that they are general purpose method that do not require problem specific knowledge

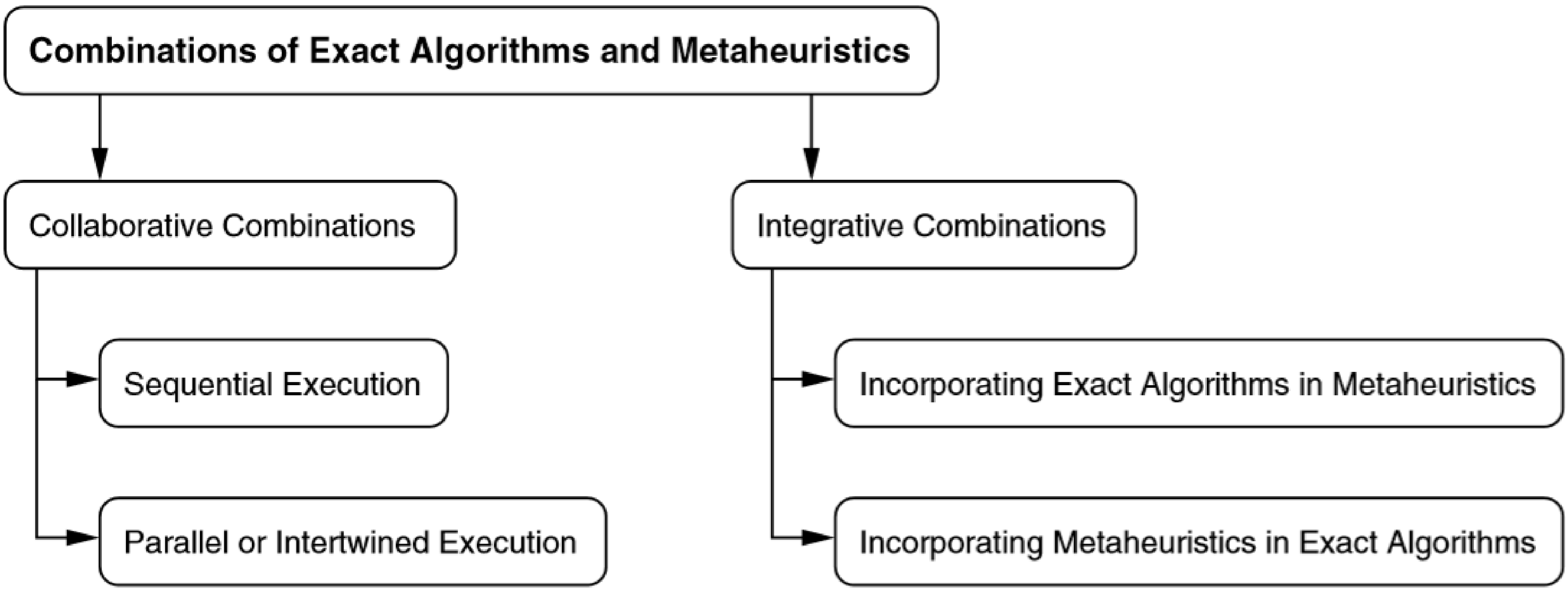
Matheuristics: general features

- Matheuristics are also called hybrid heuristics since combine the use of exact techniques with metaheuristic frameworks

- They are tailor-made algorithm (since exploit the math. structure of the problem)
  - **Advantage:** they have better performance compared to ‘general purpose’ metaheuristics
  - **Disadvantage:** they can be used only for a specific class of problems

- The performance concerns:
  1) The solution quality
  2) The computational time
  3) The robustness of the algorithm over a wide spectrum of instance types (e.g. to guarantee the algorithm can be used as optimization modules within decision support systems)

Matheuristics: a possible classification



J. Puchinger and G.R. Raidl, (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification.

Master-slave structure of Matheuristics

- Two different alternative possibilities:
  i. The metaheuristics acts at a higher level and controls the calls to the exact approach;
  ii. The exact technique acts as the master and calls and controls the use of the metaheuristic scheme

- In case i. the definition of the neighborhood follows the logic of a metaheuristic, while the exploration of the neighborhood is left to the exact approach (e.g. Corridor Method, large scale neighborhood search, local branching)

- Case ii. occurs e.g., in modern branch and cut solvers that exploits the potential of metaheuristics to quickly obtain good quality feasible solutions (useful for the pruning); or in order to find the first feasible solution, the *feasibility pump* matheuristic has been developed

Key questions designing Matheuristics

1. Which components should be “hybridized” to create an effective algorithm

2. Identification of the most effective exact methods to solve the COP (e.g., in Corridor Method: which exact method can effectively tackle the problem if of reduced size)

3. Size and boundaries of the neighborhood (they depend on the power of the exact method used)

4. Intensification-diversification tradeoff (e.g., the CM does not consider diversification, while RINS being based on the LP relaxations of the search tree put more emphasis in diversification)

Matheuristics based on linear relaxation

- The simplest matheuristic for a COP consists in rounding the solution of the linear (or continuous) relaxation of its ILP formulation

- In general this kind of approach is not good for COP with binary variables since rounding a fractional solution to 0/1 can introduce more error

- Nevertheless there are cases where this matheuristic works well even for ILP formulation with binary variables: e.g., the Minimum Weight Node Cover Problem (MWNC)

A rounding matheuristic for the MWNC

- Given an undirected graph G=(V,E) with a node cost function *c*, the Minimum Weight Node Cover Problem (MWNC) consists in finding a subset of vertices that covers i.e. touches each edge at least once and whose total cost is minimal.

$$\min z = \sum_{i=1}^{n} c_i x_i$$

$$x_i + x_j \geq 1 \; \forall [i,j] \in E$$

$$x_i \in \{0,1\} \text{ for } i = 1,..,n$$

- Let $\tilde{x}$ the optimal solution of the linear relaxation: $\forall\ [i,j] \in E,\ \widetilde{x_i} \geq 0.5 \text{ or } \widetilde{x_j} \geq 0.5$

- Therefore if we round up every $\widetilde{x_i} \geq 0.5$ and to 0 the others we obtain a feasible solution

- The value of this feasible solution, $\hat{z}$ is $\leq 2\tilde{z}$ being $\tilde{z}$ the optimal value of the LR

- Hence, $\hat{z} \leq 2\tilde{z} \leq 2z^*$, i.e., this is a 2-approximated algorithm!

Key questions for rounding matheuristics

1. What thresholds should be used for rounding?

2. What is the maximum (or average) error introduced by rounding?

3. What is the likelihood that a large number of variables will be 1 in a “typical” LP solution?

4. What is the likelihood that a large number of variable values will be close to 0 or 1 in a typical solution?

A relaxation based heuristic for the MAX-FS

- E. Amaldi, M. Bruglieri, G. Casale, (2008). A two-phase relaxation-based heuristic for the maximum feasible subsystem problem, *Computers & Operations Research,* vol. 35. issue 5, pp.1465-1482

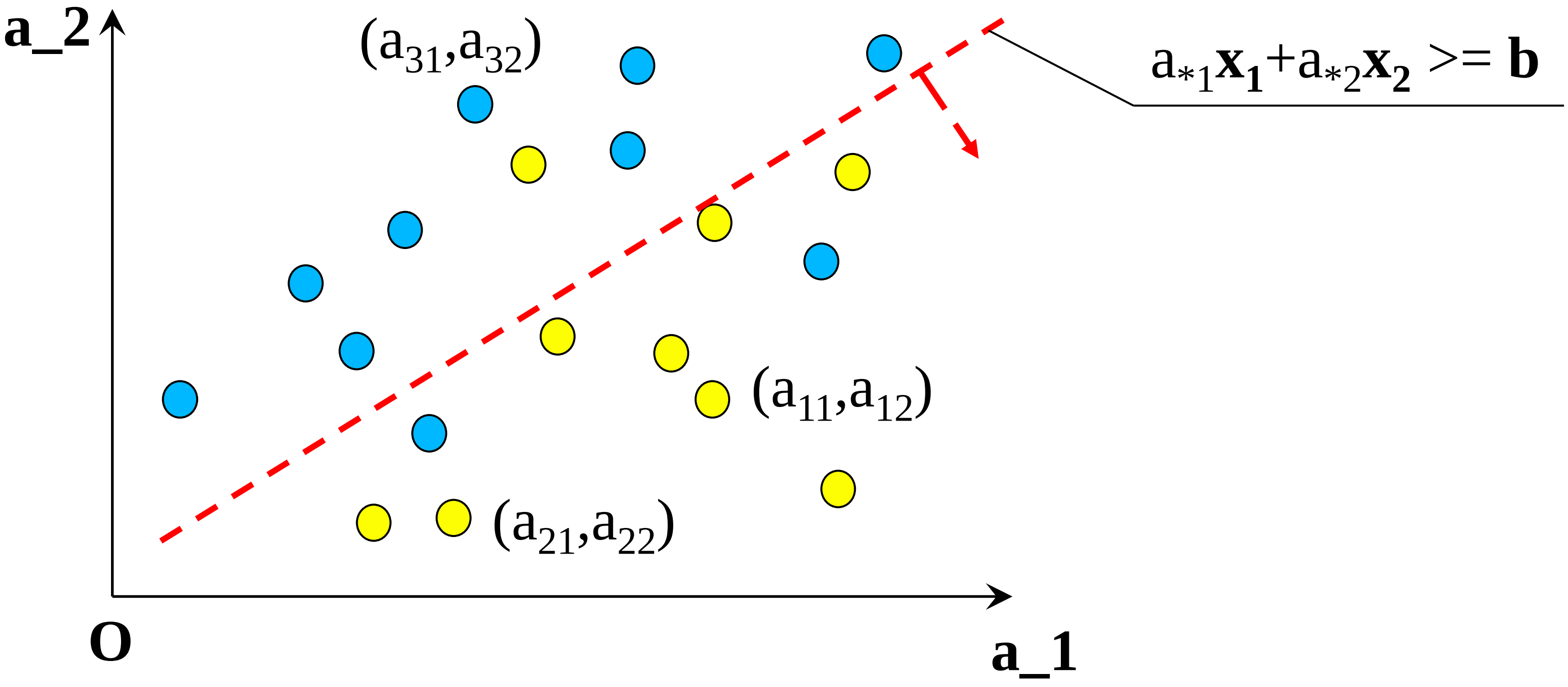
- **Max FS:** Given an infeasible $Ax \geq b$ with $A \in \Re^{m x n}$ and $b \in \Re^{m}$, find a **Max**imum **F**easible **S**ubsystem, i.e. a feasible subsystem containing as many inequalities as possible.

- We focus on the version where all variables x (or all but one) are bounded.

- **Discriminant analysis:** design optimal linear classifier (Glover '81, Mangasarian '92/'95)

- **Telecommunications:** determine antenna emission power so as to maximize coverage (Rossi et al. '01)

Linear discriminant analysis



Complexity and Approximability

- Max FS is strongly NP-hard (Sankaran ‘93)
- Max FS is approximable within 2 but does not admit a PTAS, unless P=NP (Amaldi & Kann‘95)

Dealing with Infeasibility

- **Easy to detect infeasibility** (phase 1 simplex)

- **Obstructions to feasibility**

IIS (Irridducibile Infeasible Subsystem):
Infeasible set of inequalities that becomes feasible if any of the inequalities is removed

$$x_1+x_2\geq 1$$
$$x_1\leq 0$$
$$x_2\leq 0$$

NB: Exponentially many IIS's

- **To recover feasibility**: find a **MAX FS** or equivalently a **MIN IIS Cover**

Algorithmic Approaches

Exact methods:

- MILP formulations ($a^i x \geq b_i - M(1-y_i)$ with $y_i \in \{0,1\}$)
- Partial IIS set covering formulation with dynamic IIS generation (Parker & Ryan '96)
- First Branch & Cut (Pfetsch '02)
- Combinatorial Benders'Cuts (Codato & Fischetti '04)

Filtering Heuristics

- *Chinneck's Algorithm*: iteratively remove a single constraint until the remaining subsystem is feasible
- The removed constraint is chosen using an *elastic program* E(S) associated to the infeasible system S

relation	elastic relation
$\sum_j a_{ij}x_j \geq b_i$	$\sum_j a_{ij}x_j + s_i \geq b_i$
$\sum_j a_{ij}x_j \leq b_i$	$\sum_j a_{ij}x_j - s_i \leq b_i$
$\sum_j a_{ij}x_j = b_i$	$\sum_j a_{ij}x_j + s'_i - s''_i = b_i,$

$$SINF := \min \sum_i s_i$$

Two Phase Relaxation-Based Heuristic

Bilinear Formulation

- Bilinear continuous formulation of the MAX FS:

$$
\begin{array}{llr}
\max & \sum_{i=1}^{m} y_i & \\
\text{s.t.} & y_i \sum_{j=1}^{n} a_{ij} x_j \geq y_i b_i & i = 1, \ldots, m \\
& l_j \leq x_j \leq u_j & j = 1, \ldots, n \\
& 0 \leq y_i \leq 1 & i = 1, \ldots, m.
\end{array}
$$

Linear Program with Equilibrium Constraints (LPEC)

Linearization of bilinear formulation

- Each bilinear term is replaced by a single variable

$$z_{ij} = y_i x_j$$

- The resulting formulation is thus:

$$\begin{array}{lll} \max & \sum_{i=1}^{m} y_i & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} z_{ij} \geq y_i b_i & i = 1, \ldots, m \quad (*) \\ & l_j \leq x_j \leq u_j & j = 1, \ldots, n \\ & 0 \leq y_i \leq 1 & i = 1, \ldots, m \\ & z_{ij} \geq 0 & j = 1, \ldots, n, \end{array}$$

- Nevertheless, the linearization involves a loss of information!

Assumptions

- Since each variable $x_j \in [l_j, u_j]$ we can assume w.l.o.g. $l_j = 0$:
  - if $u_j < 0$, replace x_j with $-x_j$
  - if $u_j \geq 0$ and $l_j \neq 0$, then
    - if $l_j < 0$ then $x_j = x_j^+ - x_j^-$
    - if $l_j > 0$ then $x_j = x_j^+ + l_j$
- Advantage: in constraints (*), we can replace z_{ij} with x_j, for all i and j s.t. $a_{ij} \geq 0$, since this helps to satisfy inequalities, being $x_j \geq 0$

Constraints on z_{ij}

- If $y_i \in \{0,1\}$ then $z_{ij} = y_i x_j$ if and only if:

$$y_i = 0 \Longrightarrow z_{ij} = 0 \qquad \text{for all } j = 1, \ldots, n \qquad \text{(C1)}$$

$$y_i = 1 \Longrightarrow z_{ij} = x_j \qquad \text{for all } j = 1, \ldots, n. \qquad \text{(C2)}$$

- Condition (C1) can be modelled as:

$$z_{ij} \le u_j y_i, \quad i = 1,\ldots,m,\ j = 1,\ldots,n,\ s.t.\ a_{ij} < 0$$

while (C2) as:

$$\begin{aligned} z_{ij} &\le x_j && i = 1, \ldots, m,\ j = 1, \ldots, n,\ \text{s.t. } a_{ij} < 0 \\ x_j - u_j (1 - y_i) &\le z_{ij} && i = 1, \ldots, m,\ j = 1, \ldots, n,\ \text{s.t. } a_{ij} < 0. \end{aligned}$$

Resulting linearization

$$
\begin{array}{lll}
\max & \sum_{i=1}^{m} y_i & \\
\text{s.t.} & \sum_{j:a_{ij}<0} a_{ij} z_{ij} + \sum_{j:a_{ij}\geq 0} a_{ij} x_j \geq y_i b_i & i = 1, \ldots, m \\
& z_{ij} \leq u_j y_i, & i = 1, \ldots, m, \ j = 1, \ldots, n, \ \text{s.t.} \ a_{ij} < 0 \\
& z_{ij} \leq x_j & i = 1, \ldots, m, \ j = 1, \ldots, n, \ \text{s.t.} \ a_{ij} < 0 \\
& x_j - u_j (1 - y_i) \leq z_{ij} & i = 1, \ldots, m, \ j = 1, \ldots, n, \ \text{s.t.} \ a_{ij} < 0 \\
& l_j \leq x_j \leq u_j & j = 1, \ldots, n \\
& 0 \leq y_i \leq 1 & i = 1, \ldots, m \\
& z_{ij} \geq 0 & i = 1, \ldots, m, \ j = 1, \ldots, n, \ \text{s.t.} \ a_{ij} < 0.
\end{array}
$$

Observations

- The set I of inequalities with y_i=1 is feasible
- Set I is not necessarily a subset of a MAX FS
- The inequalities corresponding to y_i<1 are not always inconsistent with the inequalities of I

Two Phase Algorithm

Phase 1:

Solve a relaxation of the MAX FS obtaining a solution $\tilde{y}$

Determine I_1={i: $\tilde{y}_i$=1, i=1 ,.., m}

Phase 2:

Solve an exact formulation of MAX FS fixing y_i=1 for all i $\in I_1$

Experimental Campaign

- 2-ph-bilin, 2-ph-bigM
- Exact-bigM
- Branch & Cut (Pfetsch ’02)
- CBC (Codato & Fischetti ’04)
- Filtering (Chinneck ’96)

- Time-limit 10000 sec on an Intel Xeon 2.80 Ghz
- Gaps with the best known optimal value (or the best known upper bound)

Instances

- **Random Instances** (Pfetsch's PhD thesis)
  - 28 groups each composed of 3 random instances
  - A and b have full density, m ≈ 20-100 and n ≈ 5-20
- **CBC-ML** (Codato et al. '04) :
  -Set of linear classification problem from
  the UCI Machine Learning repository
  - m ≈ 100-700 and n ≈ 10-40

- **ML:**
  - a different set of instances from the same repository

- **DVB** (Rossi et al. '01) **:**
  - sparse instances arising in Digital Video Broadcasts
  - m ≈ 1000-20000 and n ≈ 500
  - large difference in the coeff. values ranging between 10^{-11} and 10^{11}

Numerical results (CBC-ML)

	exact-bigM		*CBC*		*2-ph-bigM*				*2-ph-bilinear*				*Filtering*	
	FS	CPU	FS	CPU	FS		CPU		FS		CPU		FS	CPU
Instance					ph.1	ph 2	ph.1	ph.1+2	ph.1	ph.2	ph.1	ph.1+2		
Chorales-116	92	3559	92	550	46	92	0.1	22	58	92	3	11	92	14
Balloons76	66	7	66	0.1	52	66	0.1	0.1	52	66	0.1	1	66	4
BCW-367	359	365	359	1	333	359	0.1	0.3	338	359	93	93	358	5
BCW-683	673	6750	673	10	643	673	0.1	2	649	673	675	679	672	10
WPBC-194	189	2279	189	299	161	189	0.1	4	166	189	1025	1030	189	3
Breast-Cancer-400	376	71	376	0.1	374	376	0.1	0.2	374	376	0.1	3	374	13
Glass-163	150	3849	150	3	102	150	0.1	0.1	146	149	8	9	150	10
Horse-colic-151	146	592	146	12	128	146	0.1	0.1	130	146	82	84	146	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Chorales-134	103(*ub* : 113)	†	104	727	39	104	0.3	46	50	104	2	33	104	27
Chorales-107	80(*ub* : 85)	†	80	67	31	80	0.2	22	36	80	1	19	79	19
Bridges-132	108(*ub* : 121)	†	109	136	67	109	0.1	58	74	109	33	430	109	14
Mech-analysis-152	130(*ub* : 136)	†	131	139	86	131	0.2	12	117	131	6	6	128	16
Monks-tr-124	100(*ub* : 104)	†	100	56	50	100	0.1	22	55	100	3	24	97	17
Monks-tr-115	88(*ub* : 96)	†	88	487	25	88	0.1	61	49	87	2	56	88	24
Solar-flare-323	282(*ub* : 300)	†	285	3	241	284	0.1	4	254	284	94	96	281	45
Bv-os-376	367(*ub* : 369)	†	368	125	340	367	0.1	5	341	368	494	505	367	6
BusVan445	436(*ub* : 438)	†	437	102	411	437	0.1	4	412	437	320	363	437	5
Flags-169	159(*ub* : 163)	†	159	-	118	159	0.2	78	130	159	43	135	159	6
Horse-colic-253	240(*ub* : 248)	†	240	-	188	240	0.4	654	196	240	221	1275	240	15
Horse-colic-185	173(*ub* : 177)	†	173	-	137	173	0.1	42	145	173	128	272	172	9
Average				91.00	34.50%	0.34%		29.25	18.76%	0.33%		205.00	0.86%	9.44

Legend: †= time limit exceeded

Summary of average gaps

Testbed	*2-ph-bigM*		*2-ph-bilinear*		*2-ph-artificial*		*Filtering*
	ph.1	ph. 2	ph.1	ph. 2	ph.1	ph. 2	
Random	13.45%	2.16%	20.19%	**1.29%**	16.40%	3.47%	2.25%
CBC-ML	34.50%	0.34%	18.76%	**0.33%**	21.16%	0.56%	0.86%
ML	23.49%	7.22%	29.73%	7.38%	28.87%	7.53%	**7.11%**
DVB$^{\flat}$	1.73%	**0.95%**	1.96%	1.14%	16.02%	6.12%	1.33%
DVB$^{\natural}$	9.73%	7.26%	7.55%	**6.40%**	44.04%	15.74%	-

Legend: $\flat$=instances solved by all methods

$\natural$=instances solved by two-phase algorithms

Conclusions

- Simple 2-phase heuristic yields solutions with comparable quality of sophisticated exact methods within much lower CPU times
- Computational cost does not depend on the number of inequalities to be deleted to achieve feasibility (Filtering)
- Using LP relaxation of big-M formulation in Phase 1, drammatically reduces CPU times without substantially affect the solution quality

Better relaxations for Phase 1?

References

- E. Amaldi, M. Bruglieri, G. Casale, (2008). A two-phase relaxation-based heuristic for the maximum feasible subsystem problem, *Computers & Operations Research,* vol. 35. issue 5, pp.1465-1482
- M. O. Ball, (2011). Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16, pp. 21-38.
- M.Bruglieri, A. Colorni. Ricerca Operativa. Zanichelli, 2012.
- M. Caserta, S.Voβ, (2009). Metahueristics: intelligent problem solving. In: V. Maniezzo et al, (eds), Matheuristics, *Annals of Information Systems*, 10, pp. 1-38
- M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA, 1979.
- P. Hansen, V. Maniezzo, and S. Voß, (2009). Special issue on mathematical contributions to metaheuristics editorial. *Journal of Heuristics*, 15(3):197–199.
- G.L. Nemhauser, L.A. Wolsey. Integer and Combinatorial Optimization. John Wiley&Sons, 1999
- J. Puchinger and G.R. Raidl, (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. IWINAC 2005, LNCS 3562, pp. 41–53.
- G.R. Raidl, (2006). A unified view on hybrid metaheuristics. In F. Almeida, M.J. Blesa, C. Blum, J.M. Moreno-Vega, M.M. Perez, A. Roli, and M. Sampels, editors, Hybrid Metaheuristics, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer.
- M. Sniedovich and S. Voß, (2006). The corridor method: A dynamic programming inspired metaheuristic. *Control and Cybernetics*, 35:551–578.