MATHEURISTICS
FOR COMBINATORIAL
OPTIMIZATION PROBLEMS

Module 1- Lesson 1

Prof. Maurizio Bruglieri
Politecnico di Milano
Agenda

• **Lesson 1** (Monday 18th, 10.30-13):

• **Lesson 2** (Wednesday 20th, 10.30-13):
  Approximated heuristics. Dual heuristics.

• **Lesson 3** (Monday 25th, 10.30-13):

• **Lesson 4** (Wednesday 27th, 10.30-13):
  Decomposition based heuristics.
Combinatorial optimization problems

• A Combinatorial Optimization Problem (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: Traveler 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](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.
«I can’t find an efficient algorithm, I guess I’m just too dumb»
«I can’t find an efficient algorithm, because no such algorithm is possible!»
«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?

Matheuristics = math.prog. based heuristics

Mathematical Programming

Heuristics/Metaheuristics
Mathematical Programming formulation

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

• General form of a Mathematical Program:

\[
\min f(x) \\
\quad x \in X
\]

\(X \subseteq \mathbb{R}^n\) feasible solution set

\(f: X \rightarrow \mathbb{R}\) 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:

<table>
<thead>
<tr>
<th>Song</th>
<th>Appreciation</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light my fire</td>
<td>8</td>
<td>210</td>
</tr>
<tr>
<td>Fame</td>
<td>7</td>
<td>190</td>
</tr>
<tr>
<td>I will survive</td>
<td>8.5</td>
<td>235</td>
</tr>
<tr>
<td>Imagine</td>
<td>9</td>
<td>250</td>
</tr>
<tr>
<td>Let it be</td>
<td>7.5</td>
<td>200</td>
</tr>
<tr>
<td>I feel good</td>
<td>8</td>
<td>220</td>
</tr>
</tbody>
</table>

Parameters: \( n = \# \text{ songs} \), \( g_i = \text{appreciation of song } i \), \( w_i = \text{size of } i \), \( C = \text{CD capacity} \)

Decision variables: \( x_i = 1 \) if file \( i \) is selected for the CD, 0 otherwise

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

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

\( x_i \in \{0,1\} \) for \( i = 1, \ldots, n \)
The feasible region of an ILP is a subset $S \subseteq \mathbb{Z}^n$.

A **formulation** of $S \subseteq \mathbb{Z}^n$ is a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ such that $P \cap \mathbb{Z}^n = S$. 

![Graph showing the feasible region of an ILP as a polyhedron](image-url)
Mathematical Programming formulation

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

- $\text{conv}(S)$ is the ideal formulation

\[
\text{since } \begin{array}{c}
\min f(x) \\
\text{s.t. } x \in S
\end{array} \iff \begin{array}{c}
\min f(x) \\
\text{s.t. } x \in \text{conv}(S)
\end{array}
\]
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**

**Main idea:** 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:=Ø;
    while E ≠ ∅ do
        e:=element of E providing the best value of S ∪ {e};
        E := E – {e};
        if S ∪ {e} is feasible then S:= S ∪ {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

<table>
<thead>
<tr>
<th></th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
<th>Task 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empl. 1</td>
<td>15</td>
<td>22</td>
<td>18</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>Empl. 2</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>Empl. 3</td>
<td>12</td>
<td>30</td>
<td>13</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>Empl. 4</td>
<td>18</td>
<td>24</td>
<td>19</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>Empl. 5</td>
<td>13</td>
<td>23</td>
<td>14</td>
<td>32</td>
<td>25</td>
</tr>
</tbody>
</table>

**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 \( S \) represents 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

\[
m: S \rightarrow S
\]

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

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

$N(s)$ 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 $S$

**Definition**

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

$$N_l(s) = \{ \bar{s} \in S \mid d(\bar{s}, s) \leq l \}$$

where $d : S \times S \to \mathbb{R}^+$ 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:**

$s_1=\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$ and

$s_2=\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$

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 $N_1(s) = \{ \bar{s} \in S | d_H(\bar{s}, s) \leq 1 \}$ consists of all the feasible solutions that differ from $s$ in at most one component.
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.
Local search algorithm

\[Local\_Search(s);\]
begin \[s^*: = s; \text{ END} = \text{false};\]
repeat
\[s := \text{best solution in } N(s)\]
if \[f(s) < f(s^*)\] then
\[s^* := s;\]
else \[\text{ END} = \text{true};\]
until \[\text{not END};\]
return \[s^*;\]
end;
Local Optima

Definition

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

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

[Diagram showing a network before and after a move, with labeled nodes and edges]

The *move* is the swap of a pair of edges with another 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 edges 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?
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 solutions, 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

Minimum problem

Local optima
Variable Neighborhood Search (VNS)

- Developed in ‘97 by P. Hansen and N. Mladenovic

- **Idea**: a solution that is local optimal for a neighborhood could be not local optimal for another neighborhood

- Family of neighborhood $I_k$, with $k=1,\ldots,k_{\text{max}}$

- $I_k \subseteq I_{k+1}$
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 \((S, c, x^*)\):

1. begin 
2. Let \(x' = \text{Feasible}(S)\); 
3. Let \(x^* = x'\); 
4. Let \(TL=\{x'\}\); 
5. Let \(k = 0, \ stop = \text{False}\); 
6. while (\(stop = \text{False}\)) do 
7. while (\(k < \text{max\_no\_improvement}\)) do 
8. Let \(x' = \arg\min\{c(x) : x \in I(x') \setminus TL\}\) 
9. if \(c(x') < c(x^*)\) then do 
10. let \(x^* = x'\); 
11. let \(k = 0\); 
12. else \(k=k+1\); 
13. update \(TL\); 
14. end while 
15. diversification and/or intensification; 
16. if stop criterion is satisfied then \(stop = \text{True}\) 
17. end while 
18. 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:

<table>
<thead>
<tr>
<th>Solution</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,5,2,4,1</td>
<td>21</td>
</tr>
<tr>
<td>1,5,3,2,4,1</td>
<td>29</td>
</tr>
<tr>
<td>1,2,5,3,4,1</td>
<td>26</td>
</tr>
<tr>
<td>1,3,2,5,4,1</td>
<td>29</td>
</tr>
<tr>
<td>1,3,4,2,5,1</td>
<td>20</td>
</tr>
<tr>
<td>1,2,4,5,3,1</td>
<td>31</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Solution</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,4,2,5,1</td>
<td>20</td>
</tr>
<tr>
<td>1,3,5,2,4,1</td>
<td>21</td>
</tr>
<tr>
<td>1,3,2,4,5,1</td>
<td>31</td>
</tr>
<tr>
<td>1,4,3,2,5,1</td>
<td>26</td>
</tr>
<tr>
<td>1,2,4,3,5,1</td>
<td>28</td>
</tr>
<tr>
<td>1,3,4,5,2,1</td>
<td>28</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Solution</th>
<th>z</th>
<th>tabu</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4,3,5,2,1</td>
<td>26</td>
<td>tabu</td>
</tr>
<tr>
<td>1,4,3,2,5,1</td>
<td>26</td>
<td>tabu</td>
</tr>
<tr>
<td>1,3,4,5,2,1</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>1,5,3,4,2,1</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>1,4,5,3,2,1</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>1,4,2,5,3,1</td>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>

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

Combinations of Exact Algorithms and Metaheuristics

Collaborative Combinations
- Sequential Execution
- Parallel or Intertwined Execution

Integrative Combinations
- Incorporating Exact Algorithms in Metaheuristics
- Incorporating Metaheuristics in Exact Algorithms

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.

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

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

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

- Let $\tilde{x}$ the optimal solution of the linear relaxation:

  $\forall [i,j] \in E$, either $\tilde{x}_i \geq 0.5$ or $\tilde{x}_j \geq 0.5$

- Therefore if we round up every $\tilde{x}_i \geq 0.5$ 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


- **Max FS:** Given an infeasible $Ax \geq b$ with $A \in \mathbb{R}^{mxn}$ and $b \in \mathbb{R}^m$, find a **Maximum Feasible Subsystem**, 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

\[ a_{*1}x_1 + a_{*2}x_2 \geq b \]
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** (Irriducibile 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 $S$

\[
\begin{align*}
\sum_j a_{ij} x_j &\geq b_i \\
\sum_j a_{ij} x_j &\leq b_i \\
\sum_j a_{ij} x_j &= b_i
\end{align*}
\]

elastic relation
\[
\begin{align*}
\sum_j a_{ij} x_j + s_i &\geq b_i \\
\sum_j a_{ij} x_j - s_i &\leq b_i \\
\sum_j a_{ij} x_j + s_i' - s_i'' &= b_i
\end{align*}
\]

$SINF := \min \sum_i s_i$
Two Phase Relaxation-Based Heuristic
Bilinear Formulation

• Bilinear continuous formulation of the MAX FS:

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{m} y_i \\
\text{s.t.} & \quad y_i \sum_{j=1}^{n} a_{ij} x_j \geq y_i b_i \quad i = 1, \ldots, m \\
& \quad l_j \leq x_j \leq u_j \quad j = 1, \ldots, n \\
& \quad 0 \leq y_i \leq 1 \quad i = 1, \ldots, m.
\end{align*}
\]

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:

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

- 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 $z_{ij} = y_i \cdot x_j$ if and only if:
  
  $$y_i = 0 \implies z_{ij} = 0 \quad \text{for all } j = 1, \ldots, n \quad (C1)$$
  $$y_i = 1 \implies z_{ij} = x_j \quad \text{for all } j = 1, \ldots, n. \quad (C2)$$

- Condition (C1) can be modelled as:
  
  $$z_{ij} \leq u_j \cdot y_i, \quad i = 1, \ldots, m, \ j = 1, \ldots, n, \ s.t. \ a_{ij} < 0$$

while (C2) as:

$$z_{ij} \leq x_j \quad i = 1, \ldots, m, \ j = 1, \ldots, n, \ s.t. \ a_{ij} < 0$$
$$x_j - u_j(1 - y_i) \leq z_{ij} \quad i = 1, \ldots, m, \ j = 1, \ldots, n, \ s.t. \ a_{ij} < 0.$$
Resulting linearization

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{m} y_i \\
\text{s.t.} & \quad \sum_{j: a_{ij} < 0} a_{ij} z_{ij} + \sum_{j: a_{ij} \geq 0} a_{ij} x_j \geq y_i b_i \\
& \quad z_{ij} \leq u_j y_i, \\
& \quad z_{ij} \leq x_j \\
& \quad x_j - u_j (1 - y_i) \leq z_{ij} \\
& \quad l_j \leq x_j \leq u_j \\
& \quad 0 \leq y_i \leq 1 \\
& \quad z_{ij} \geq 0
\end{align*}
\]

\[i = 1, \ldots, m\]

\[i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad \text{s.t.} \quad a_{ij} < 0\]

\[i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad \text{s.t.} \quad a_{ij} < 0\]

\[i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad \text{s.t.} \quad a_{ij} < 0\]

\[j = 1, \ldots, n\]

\[i = 1, \ldots, m\]

\[i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad \text{s.t.} \quad a_{ij} < 0.\]
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 $I_1 = \{i: \tilde{y}_i = 1, \ i=1, \ldots, m\}$

**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, $m \approx 20-100$ and $n \approx 5-20$

- **CBC-ML** (Codato et al. ’04):
  - Set of linear classification problem from the UCI Machine Learning repository
  - $m \approx 100-700$ and $n \approx 10-40$

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

- **DVB** (Rossi et al. ’01):
  - sparse instances arising in Digital Video Broadcasts
  - $m \approx 1000-20000$ and $n \approx 500$
  - large difference in the coeff. values ranging between $10^{-11}$ and $10^{11}$
Numerical results (CBC-ML)

<table>
<thead>
<tr>
<th>Instance</th>
<th>exact-bigM</th>
<th>CBC</th>
<th>2-ph-bigM</th>
<th>Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FS</td>
<td>CPU</td>
<td>FS</td>
<td>CPU</td>
</tr>
<tr>
<td>Chorales-116</td>
<td>92</td>
<td>3559</td>
<td>92</td>
<td>550</td>
</tr>
<tr>
<td>Balloons76</td>
<td>66</td>
<td>7</td>
<td>66</td>
<td>0.1</td>
</tr>
<tr>
<td>BCW-367</td>
<td>359</td>
<td>365</td>
<td>359</td>
<td>1</td>
</tr>
<tr>
<td>BCW-683</td>
<td>673</td>
<td>6750</td>
<td>673</td>
<td>10</td>
</tr>
<tr>
<td>WPBC-194</td>
<td>189</td>
<td>2279</td>
<td>189</td>
<td>299</td>
</tr>
<tr>
<td>Breast-Cancer-400</td>
<td>376</td>
<td>71</td>
<td>376</td>
<td>0.1</td>
</tr>
<tr>
<td>Glass-163</td>
<td>150</td>
<td>3849</td>
<td>150</td>
<td>3</td>
</tr>
<tr>
<td>Horse-colic-151</td>
<td>146</td>
<td>592</td>
<td>146</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Chorales-134</td>
<td>103</td>
<td>(ub : 113)</td>
<td>104</td>
<td>727</td>
</tr>
<tr>
<td>Chorales-107</td>
<td>80</td>
<td>(ub : 85)</td>
<td>80</td>
<td>67</td>
</tr>
<tr>
<td>Bridges-132</td>
<td>108</td>
<td>(ub : 121)</td>
<td>109</td>
<td>136</td>
</tr>
<tr>
<td>Mech-analysis-152</td>
<td>130</td>
<td>(ub : 136)</td>
<td>131</td>
<td>139</td>
</tr>
<tr>
<td>Monks-tr-124</td>
<td>100</td>
<td>(ub : 104)</td>
<td>100</td>
<td>56</td>
</tr>
<tr>
<td>Monks-tr-115</td>
<td>88</td>
<td>(ub : 96)</td>
<td>88</td>
<td>487</td>
</tr>
<tr>
<td>Solar-flare-323</td>
<td>282</td>
<td>(ub : 300)</td>
<td>285</td>
<td>3</td>
</tr>
<tr>
<td>Bv-os-376</td>
<td>367</td>
<td>(ub : 369)</td>
<td>368</td>
<td>125</td>
</tr>
<tr>
<td>BvVan445</td>
<td>436</td>
<td>(ub : 438)</td>
<td>437</td>
<td>102</td>
</tr>
<tr>
<td>Flags-169</td>
<td>159</td>
<td>(ub : 163)</td>
<td>159</td>
<td>-</td>
</tr>
<tr>
<td>Horse-colic-253</td>
<td>240</td>
<td>(ub : 248)</td>
<td>240</td>
<td>-</td>
</tr>
<tr>
<td>Horse-colic-185</td>
<td>173</td>
<td>(ub : 177)</td>
<td>173</td>
<td>-</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td><strong>91.00</strong></td>
<td><strong>34.50%</strong></td>
</tr>
</tbody>
</table>
## Summary of average gaps

<table>
<thead>
<tr>
<th>Testbed</th>
<th>2-ph-bigM</th>
<th>2-ph-bilinear</th>
<th>2-ph-artificial</th>
<th>Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ph.1</td>
<td>ph. 2</td>
<td>ph.1</td>
<td>ph. 2</td>
</tr>
<tr>
<td>Random</td>
<td>13.45%</td>
<td>2.16%</td>
<td>20.19%</td>
<td>1.29%</td>
</tr>
<tr>
<td>CBC-ML</td>
<td>34.50%</td>
<td>0.34%</td>
<td>18.76%</td>
<td>0.33%</td>
</tr>
<tr>
<td>ML</td>
<td>23.49%</td>
<td>7.22%</td>
<td>29.73%</td>
<td>7.38%</td>
</tr>
<tr>
<td>DVB\textsuperscript{b}</td>
<td>1.73%</td>
<td>0.95%</td>
<td>1.96%</td>
<td>1.14%</td>
</tr>
<tr>
<td>DVB\textsuperscript{g}</td>
<td>9.73%</td>
<td>7.26%</td>
<td>7.55%</td>
<td>6.40%</td>
</tr>
</tbody>
</table>

Legend: \textsuperscript{b}=instances solved by all methods  
\textsuperscript{g}=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, dramatically reduces CPU times without substantially affect the solution quality

Better relaxations for Phase 1?
References