How to use your favorite MIP Solver:
modeling, solving, cannibalizing

Andrea Lodi

*University of Bologna, Italy*
andrea.lodi@unibo.it

January-February, 2012 © Universität Wien
Setting

- We consider a general Mixed Integer Program in the form:

\[
\max \{ c^T x : Ax \leq b, x \geq 0, x_j \in \mathbb{Z}, \forall j \in I \} 
\]  

(1)

where matrix A does not have a special structure.

- Thus, the problem is solved through branch-and-bound and the bounds are computed by iteratively solving the LP relaxations through a general-purpose LP solver.

- The course basically covers the MIP but we will try to discuss when possible how crucial is the LP component (the engine), and how much the whole framework is built on top the capability of effectively solving LPs.

- Roughly speaking, using the LP computation as a tool, MIP solvers integrate the branch-and-bound and the cutting plane algorithms through variations of the general branch-and-cut scheme [Padberg & Rinaldi 1987] developed in the context of the Traveling Salesman Problem (TSP).
Outline

1. The building blocks of a MIP solver.
   We will run over the first 50 exciting years of MIP by showing some crucial milestones and we will highlight the building blocks that are making nowadays solvers effective from both a performance and an application viewpoint.

2. How to use a MIP solver as a sophisticated (heuristic) framework.
   Nowadays MIP solvers should not be conceived as black-box exact tools. In fact, they provide countless options for their smart use as hybrid algorithmic frameworks, which thing might turn out especially interesting on the applied context. We will review some of those options and possible hybridizations, including some real-world applications.
Outline

1. The building blocks of a MIP solver.
   We will run over the first 50 exciting years of MIP by showing some crucial milestones and we will highlight the building blocks that are making nowadays solvers effective from both a performance and an application viewpoint.

2. How to use a MIP solver as a sophisticated (heuristic) framework.
   Nowadays MIP solvers should not be conceived as black-box exact tools. In fact, they provide countless options for their smart use as hybrid algorithmic frameworks, which thing might turn out especially interesting on the applied context. We will review some of those options and possible hybridizations, including some real-world applications.

3. Modeling and algorithmic tips to make a solver effective in practice.
   The capability of a solver to produce good, potentially optimal, solutions depends on the selection of the right model and the use of the right algorithmic tools the solver provides. We will discuss useful tips, from simple to sophisticated, which allow a smart use of a MIP solver.

Finally, we will show that this is NOT the end of the story and many challenges for MIP technology are still to be faced.
PART 2

1. The building blocks of a MIP solver

2. How to use a MIP solver as a sophisticated (heuristic) framework
PART 2

1. The building blocks of a MIP solver

2. How to use a MIP solver as a sophisticated (heuristic) framework

3. Modeling and algorithmic tips to make a solver effective in practice

- Outline:
  - The heuristic (dark) side of MIP solvers
1. The building blocks of a MIP solver

2. How to use a MIP solver as a sophisticated (heuristic) framework

3. Modeling and algorithmic tips to make a solver effective in practice

● Outline:
  – The heuristic (dark) side of MIP solvers
  – Heuristics in MIP
    1. Achieving Integer-Feasibility Quickly
    2. Reaching (quasi-)Optimality Quickly
    3. Analyzing Infeasible MIP solutions
PART 2

1. The building blocks of a MIP solver

2. How to use a MIP solver as a sophisticated (heuristic) framework

3. Modeling and algorithmic tips to make a solver effective in practice

- Outline:
  - The heuristic (dark) side of MIP solvers
  - Heuristics in MIP
    1. Achieving Integer-Feasibility Quickly
    2. Reaching (quasi-)Optimality Quickly
    3. Analyzing Infeasible MIP solutions
  - Using MIP heuristics in Applications
Are MIP solvers really “exact”?

• What has been presented in Part 1 is, apparently, a setting extremely far away from what we think is in the context of Metaheuristics and Hybrid Algorithms.
Are MIP solvers really “exact”? 

• What has been presented in Part 1 is, apparently, a setting extremely far away from what we think is in the context of Metaheuristics and Hybrid Algorithms. 

• Common sense says a MIP solver is a pure and exact algorithm (in contrast to a hybrid and heuristic one). 

• However, it is time to confute this viewpoint by showing that:
Are MIP solvers really “exact”?  

- What has been presented in Part 1 is, apparently, a setting extremely far away from what we think is in the context of Metaheuristics and Hybrid Algorithms.
- Common sense says a MIP solver is a pure and exact algorithm (in contrast to a hybrid and heuristic one).

However, it is time to confute this viewpoint by showing that:

1. MIP solvers are used for a large portion as heuristics
2. MIP solvers are heuristic in nature
3. Computation for $NP$-hard problems is inherently heuristic
4. Benchmarking is by design heuristic
5. All kind of heuristic decisions/techniques are hidden everywhere in MIP solvers
Are MIP solvers really “exact”? 

• What has been presented in Part 1 is, apparently, a setting extremely far away from what we think is in the context of Metaheuristics and Hybrid Algorithms.
• Common sense says a MIP solver is a pure and exact algorithm (in contrast to a hybrid and heuristic one).

• However, it is time to confute this viewpoint by showing that:

1. MIP solvers are used for a large portion as heuristics
2. MIP solvers are heuristic in nature
3. Computation for $\mathcal{NP}$-hard problems is inherently heuristic
4. Benchmarking is by design heuristic
5. All kind of heuristic decisions/techniques are hidden everywhere in MIP solvers

S: MIP solvers are open to different “worlds” and nowadays more and more hybrid algorithms.
Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces limits to the computation. The most classical of these limits is the time limit, but other limits are often applied, like number of solutions (sometimes just 1), percentage gap and number of nodes.
Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces limits to the computation. The most classical of these limits is the time limit, but other limits are often applied, like number of solutions (sometimes just 1), percentage gap and number of nodes. The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces limits to the computation. The most classical of these limits is the time limit, but other limits are often applied, like number of solutions (sometimes just 1), percentage gap and number of nodes. The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   – it is just a portion of the iterative decision process
   – the overall problem is highly complex and it is split a priori into pieces
   – it must be done online
   – . . .
Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces limits to the computation. The most classical of these limits is the time limit, but other limits are often applied, like number of solutions (sometimes just 1), percentage gap and number of nodes. The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   – it is just a portion of the iterative decision process
   – the overall problem is highly complex and it is split a priori into pieces
   – it must be done online
   – . . .

2. Any MIP solver works with tolerances.
Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces limits to the computation. The most classical of these limits is the time limit, but other limits are often applied, like number of solutions (sometimes just 1), percentage gap and number of nodes. The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split a priori into pieces
   - it must be done online
   - ...

2. Any MIP solver works with tolerances. The tolerance is both in the solution of the LPs and on the branching side.
Trivial Facts (1. and 2.)

Some of the points anticipated above are trivial:

1. The classical user of a MIP solver enforces limits to the computation. The most classical of these limits is the time limit, but other limits are often applied, like number of solutions (sometimes just 1), percentage gap and number of nodes.

   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split a priori into pieces
   - it must be done online
   - . . .

2. Any MIP solver works with tolerances.

   The tolerance is both in the solution of the LPs and on the branching side.
   It is pretty tight for feasibility, thus good solvers certify their solutions as “really feasible”.
Some of the points anticipated above are trivial:

1. The classical user of a MIP solver **enforces limits** to the computation. The most classical of these limits is the **time limit**, but other limits are often applied, like **number of solutions** (sometimes just 1), percentage **gap** and number of **nodes**.
   The computational resources are limited and sometimes solving MIPs does not require/allow looking for optimality because:
   - it is just a portion of the iterative decision process
   - the overall problem is highly complex and it is split **a priori** into pieces
   - it must be done **online**
   - . . .

2. Any MIP solver works with **tolerances**.
   The tolerance is both in the solution of the LPs and on the branching side.
   It is pretty **tight for feasibility**, thus good solvers certify their solutions as “really feasible”.
   It is **less strict for optimality**. A popular default value for that is 0.01% which for special applications is far from acceptable [Koch et al. 2011].
   (Ever noticed the number of nodes left to explore at the end of a run?)
Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.
Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.
Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:
3. MIP $\mathcal{NP}$-hardness.

By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

– bad algorithmic decisions (discussed later)
3. MIP $\mathcal{NP}$-hardness.

By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

- bad algorithmic decisions (discussed later)

- performance variability:
  
  * a change in performance
  * for the same problem (or problems in the same family)
  * created by a change in the solver or in the environment
  * that seems “performance neutral”.
Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{NP}$-hardness.

By definition there is always in the tree a polynomial path to the optimal solution. However, unless $\mathcal{P} = \mathcal{NP}$, in the worst-case, the path followed will be exponentially long.

In other words, branch-and-bound algorithms HEURISTICALLY explore the tree and such an exploration can be unlucky because of:

- bad algorithmic decisions (discussed later)

- performance variability:
  
  * a change in performance
  * for the same problem (or problems in the same family)
  * created by a change in the solver or in the environment
  * that seems “performance neutral”.

- ...
Example: 10 teams, CPLEX 11, Linux

Tried aggregator 1 time.
MIP Presolve eliminated 20 rows and 425 columns.
Reduced MIP has 210 rows, 1600 columns, and 9600 nonzeros.
Presolve time = 0.01 sec.
Clique table members: 170.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.05 sec.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Node</th>
<th>Left</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Integer</th>
<th>Best Node</th>
<th>ItCnt</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>917.0000</td>
<td>140</td>
<td>917.0000</td>
<td>1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>924.0000</td>
<td>165</td>
<td>Cuts: 50</td>
<td>1969</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>924.0000</td>
<td>167</td>
<td>Cuts: 17</td>
<td>2348</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>924.0000</td>
<td>175</td>
<td>Cliquess: 14</td>
<td>2731</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
<td>0</td>
<td>924.0000</td>
<td>924.0000</td>
<td>2731</td>
<td>0.00%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clique cuts applied: 16
Zero-half cuts applied: 3
Gomory fractional cuts applied: 1

Solution pool: 1 solution saved.

MIP - Integer optimal solution: Objective = 9.240000000e+02
Solution time = 0.41 sec. Iterations = 2731 Nodes = 0
Example: 10 teams, CPLEX 11, AIX

Tried aggregator 1 time.
MIP Presolve eliminated 20 rows and 425 columns.
Reduced MIP has 210 rows, 1600 columns, and 9600 nonzeros.
Presolve time = 0.00 sec.
Clique table members: 170.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.18 sec.

| Nodes | Cuts/
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Leaf</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>* 100+</td>
<td>96</td>
</tr>
<tr>
<td>1000</td>
<td>520</td>
</tr>
<tr>
<td>* 1425</td>
<td>0</td>
</tr>
</tbody>
</table>

Clique cuts applied: 12
Zero-half cuts applied: 4
Gomory fractional cuts applied: 2

Solution pool: 2 solutions saved.

MIP - Integer optimal solution: Objective = 9.2400000000e+02
Solution time = 41.39 sec. Iterations = 122948 Nodes = 1426
Slightly-less Trivial Facts: ④.

④. Benchmarking of (commercial) MIP solvers.
4. Benchmarking of (commercial) MIP solvers.

The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.
4. Benchmarking of (commercial) MIP solvers.

The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

Any new idea is tested on the entire testset and in order to “make it into the solver” it must
Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

Any new idea is tested on the entire testset and in order to “make it into the solver” it must

– improve on the subset of problems to which the idea applies, and

– not deteriorate (too much) on the rest of the problems, generally the easy ones
**Slightly-less Trivial Facts: 4.**

4. **Benchmarking of (commercial) MIP solvers.**

The testsets of MIP solvers, in particular the commercial ones, are made of thousands of instances. Instances are classified into categories from very easy to very difficult.

Any new idea is tested on the entire testset and in order to “make it into the solver” it must

- improve on the subset of problems to which the idea applies, and

- not deteriorate (too much) on the rest of the problems, generally the easy ones.

Because of the MIP \( \mathcal{NP} \)-hardness, it is both theoretically and practically hard to recognize problems as good or bad for an idea, then such an idea must be **heuristically “weakened”** to accomplish simultaneously the two given goals.
5. MIP Solvers, nowadays key features

- We have just discussed:
5. MIP Solvers, nowadays key features

- We have just discussed:
  
  A Preprocessing:
  - probing, bound strengthening, propagation
5. MIP Solvers, nowadays key features

- We have just discussed:
  
  A Preprocessing:
  - probing, bound strengthening, propagation
  
  B Cutting plane generation:
  - Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, \( \{0, \frac{1}{2}\} \)-cuts, ...
5. MIP Solvers, nowadays key features

- We have just discussed:

  A  **Preprocessing:**
     probing, bound strengthening, propagation

  B  **Cutting plane generation:**
     Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, \( \{0, \frac{1}{2}\} \)-cuts, 
     ...

  C  **Sophisticated branching strategies:**
     strong branching, pseudo-cost branching, diving and hybrids
5. MIP Solvers, nowadays key features

• We have just discussed:

A Preprocessing:
   probing, bound strengthening, propagation

B Cutting plane generation:
   Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, \{0, \frac{1}{2}\}-cuts, 
   . . .

C Sophisticated branching strategies:
   strong branching, pseudo-cost branching, diving and hybrids

D Primal heuristics:
   rounding heuristics (from easy to complex), local search, metaheuristics, . . .
5. MIP Solvers, nowadays key features

- We have just discussed:

  A **Preprocessing:**
  - probing, bound strengthening, propagation

  B **Cutting plane generation:**
  - Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, \( \{0, \frac{1}{2}\}\)-cuts, ...

  C **Sophisticated branching strategies:**
  - strong branching, pseudo-cost branching, diving and hybrids

  D **Primal heuristics:**
  - rounding heuristics (from easy to complex), local search, metaheuristics, ...
5.A: Preprocessing

- Many of the techniques used in preprocessing are called in the *Constraint Programming* context propagation algorithms:
5.A: Preprocessing

- Many of the techniques used in preprocessing are called in the *Constraint Programming* context propagation algorithms:

- A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
5.A: Preprocessing

• Many of the techniques used in preprocessing are called in the *Constraint Programming* context propagation algorithms:

- A **global constraint** defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.

- Moreover, a global constraint contains an algorithm that prunes (filters) values from the variable domains so as to reduce as much as possible the search space.
5.A: Preprocessing

• Many of the techniques used in preprocessing are called in the *Constraint Programming* context propagation algorithms:

  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
  - Moreover, a global constraint contains an algorithm that prunes (filters) values from the variable domains so as to reduce as much as possible the search space.

• MIPs do not explicitly contain global constraints, thus, the propagation is applied by LOCALLY comparing constraints/variables, a structurally heuristic process.
5. A: Preprocessing

• Many of the techniques used in preprocessing are called in the *Constraint Programming* context propagation algorithms:

- A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.

- Moreover, a global constraint contains an algorithm that prunes (filters) values from the variable domains so as to reduce as much as possible the search space.

• MIPs do not explicitly contain global constraints, thus, the propagation is applied by **LOCALLY comparing constraints/variables**, a structurally heuristic process.

• Indeed, random permutations of rows/columns of the MIP generally lead to worse performance of the solvers mostly because of reduced preprocessing effectiveness.
5.B: Cutting Planes
• Current MIP cutting plane technology is based on the following almost trivial two-step heuristic:
Current MIP cutting plane technology is based on the following almost trivial two-step heuristic:

- HEURISTICALLY aggregate the entire MIP into a mixed-integer set of one single (!) row
- apply the cut separation procedure to such a mixed-integer set (Gomory fractional, Gomory Mixed-Integer, MIR, \(\{0, \frac{1}{2}\}, \ldots\) )
• Current MIP cutting plane technology is based on the following almost trivial two-step heuristic:
  – HEURISTICALLY aggregate the entire MIP into a mixed-integer set of one single (!) row
  – apply the cut separation procedure to such a mixed-integer set
    (Gomory fractional, Gomory Mixed-Integer, MIR, \{0, \frac{1}{2}\}, \ldots)

• The most recent heuristic decision that appeared to be highly crucial is the cut selection, i.e.,
  which among all possible separated cutting planes should be added?
5.C: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented by commercial and non-commercial MIP solvers.
The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented by commercial and non-commercial MIP solvers.

“Bad” decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.
5.C: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented by commercial and non-commercial MIP solvers.

- “Bad” decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- As discussed “strong branching” techniques are extensively applied in MIP solvers.
5. C: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented by commercial and non-commercial MIP solvers.

- "Bad" decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- As discussed “strong branching” techniques are extensively applied in MIP solvers.

- However, the computational effort required by using “fully” such a technique would be too high (at every decision point, all variable-value assignments would need to be evaluated).
• The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented by commercial and non-commercial MIP solvers.

• “Bad” decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

• As discussed “strong branching” techniques are extensively applied in MIP solvers.

• However, the computational effort required by using “fully” such a technique would be too high (at every decision point, all variable-value assignments would need to be evaluated). Thus, two heuristic criteria are applied:
  – only a subset of these assignments are evaluated, and
  – each LP is not solved to optimality but within a given iteration limit.
5. C: Branching

- The branching phase is by construction (probably) the most delicate part of the branch-and-cut framework currently implemented by commercial and non-commercial MIP solvers.

- “Bad” decisions at early stages of the search, i.e., high levels in the tree, result in the exponential increase of the tree itself.

- As discussed “strong branching” techniques are extensively applied in MIP solvers.

- However, the computational effort required by using “fully” such a technique would be too high (at every decision point, all variable-value assignments would need to be evaluated). Thus, two heuristic criteria are applied:
  - only a subset of these assignments are evaluated, and
  - each LP is not solved to optimality but within a given iteration limit.

- The heuristic side of branching is not limited to the above criteria and has an impact in almost all branching components, as for example in the decision on how to break ties.
This is of course the most trivial context in which “real” heuristics play a central role in the MIP solvers.
5.D: Primal Heuristics

- This is of course the most trivial context in which “real” heuristics play a central role in the MIP solvers.

- As discussed, the integration of the primal heuristics in the MIP solvers had a huge impact in terms of user perception, and such integration is becoming tighter and tighter.
5.D: Primal Heuristics

• This is of course the most trivial context in which “real” heuristics play a central role in the MIP solvers.

• As discussed, the integration of the primal heuristics in the MIP solvers had a huge impact in terms of user perception, and such integration is becoming tighter and tighter.

• However, the most surprising impact of primal heuristics in the solver is on MIPping.
5.D: Primal Heuristics

• This is of course the most trivial context in which “real” heuristics play a central role in the MIP solvers.

• As discussed, the integration of the primal heuristics in the MIP solvers had a huge impact in terms of user perception, and such integration is becoming tighter and tighter.

• However, the most surprising impact of primal heuristics in the solver is on MIPping.

   Indeed, the sub-MIPs used to find better solutions and/or to generate cuts are NEVER solved to optimality: the full integration of sophisticated heuristics in the solvers allows to count on the fact that nested calls of the same solvers could produce heuristic solutions fast!
Overall: does Cplex v.x dominate Cplex v.y \((x > y)\)?

- Let’s go back and look at the Cplex numbers.
Overall: does Cplex v.x dominate Cplex v.y \((x > y)\)?

• Let’s go back and look at the Cplex numbers.

<table>
<thead>
<tr>
<th>Cplex versions</th>
<th>year</th>
<th>better</th>
<th>worse</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.0</td>
<td>2007</td>
<td>0</td>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>10.0</td>
<td>2005</td>
<td>201</td>
<td>650</td>
<td>1.91</td>
</tr>
<tr>
<td>9.0</td>
<td>2003</td>
<td>142</td>
<td>793</td>
<td>2.73</td>
</tr>
<tr>
<td>8.0</td>
<td>2002</td>
<td>117</td>
<td>856</td>
<td>3.56</td>
</tr>
<tr>
<td>7.1</td>
<td>2001</td>
<td>63</td>
<td>930</td>
<td>4.59</td>
</tr>
<tr>
<td>6.5</td>
<td>1999</td>
<td>71</td>
<td>997</td>
<td>7.47</td>
</tr>
<tr>
<td>6.0</td>
<td>1998</td>
<td>55</td>
<td>1060</td>
<td>21.30</td>
</tr>
<tr>
<td>5.0</td>
<td>1997</td>
<td>45</td>
<td>1069</td>
<td>22.57</td>
</tr>
<tr>
<td>4.0</td>
<td>1995</td>
<td>37</td>
<td>1089</td>
<td>26.29</td>
</tr>
<tr>
<td>3.0</td>
<td>1994</td>
<td>34</td>
<td>1107</td>
<td>34.63</td>
</tr>
<tr>
<td>2.1</td>
<td>1993</td>
<td>13</td>
<td>1137</td>
<td>56.16</td>
</tr>
<tr>
<td>1.2</td>
<td>1991</td>
<td>17</td>
<td>1132</td>
<td>67.90</td>
</tr>
</tbody>
</table>
Overall: does Cplex v.\(x\) dominate Cplex v.\(y\) \((x > y)\)?

- Let’s go back and look at the Cplex numbers.

<table>
<thead>
<tr>
<th>Cplex versions</th>
<th>year</th>
<th>better</th>
<th>worse</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.0</td>
<td>2007</td>
<td>0</td>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>10.0</td>
<td>2005</td>
<td>201</td>
<td>650</td>
<td>1.91</td>
</tr>
<tr>
<td>9.0</td>
<td>2003</td>
<td>142</td>
<td>793</td>
<td>2.73</td>
</tr>
<tr>
<td>8.0</td>
<td>2002</td>
<td>117</td>
<td>856</td>
<td>3.56</td>
</tr>
<tr>
<td>7.1</td>
<td>2001</td>
<td>63</td>
<td>930</td>
<td>4.59</td>
</tr>
<tr>
<td>6.5</td>
<td>1999</td>
<td>71</td>
<td>997</td>
<td>7.47</td>
</tr>
<tr>
<td>6.0</td>
<td>1998</td>
<td>55</td>
<td>1060</td>
<td>21.30</td>
</tr>
<tr>
<td>5.0</td>
<td>1997</td>
<td>45</td>
<td>1069</td>
<td>22.57</td>
</tr>
<tr>
<td>4.0</td>
<td>1995</td>
<td>37</td>
<td>1089</td>
<td>26.29</td>
</tr>
<tr>
<td>3.0</td>
<td>1994</td>
<td>34</td>
<td>1107</td>
<td>34.63</td>
</tr>
<tr>
<td>2.1</td>
<td>1993</td>
<td>13</td>
<td>1137</td>
<td>56.16</td>
</tr>
<tr>
<td>1.2</td>
<td>1991</td>
<td>17</td>
<td>1132</td>
<td>67.90</td>
</tr>
</tbody>
</table>

- There are 17 instances on which Cplex 1.2 (1991) is at least 10% faster than Cplex 11.0 (2007)!
Summary: MIP solvers as Hybrid Algorithms

- Not only MIP solvers
Summary: MIP solvers as Hybrid Algorithms

- Not only MIP solvers
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
Summary: MIP solvers as Hybrid Algorithms

- Not only MIP solvers
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
Summary: MIP solvers as Hybrid Algorithms

- Not only MIP solvers
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components
Summary: MIP solvers as Hybrid Algorithms

● Not only MIP solvers

   – are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)

   – are developed and tested using heuristic criteria

   – use heuristic decisions in each of their basic components

● but also incorporate
Summary: MIP solvers as Hybrid Algorithms

- Not only MIP solvers
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components

- but also incorporate
  - heuristic algorithms which are not only used to find good solutions but also as building blocks of sophisticated algorithmic strategies like MIPping
Summary: MIP solvers as Hybrid Algorithms

- Not only MIP solvers
  - are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  - are developed and tested using heuristic criteria
  - use heuristic decisions in each of their basic components

- but also incorporate
  - heuristic algorithms which are not only used to find good solutions but also as building blocks of sophisticated algorithmic strategies like MIPping
  - propagation algorithms from Constraint Programming.
Summary: MIP solvers as Hybrid Algorithms

• Not only MIP solvers
  – are used as heuristics (they are in several important cases the best heuristic approaches even in the case of structured problems, e.g., set covering type problems)
  – are developed and tested using heuristic criteria
  – use heuristic decisions in each of their basic components

• but also incorporate
  – heuristic algorithms which are not only used to find good solutions but also as building blocks of sophisticated algorithmic strategies like MIPping
  – propagation algorithms from Constraint Programming.

• In other words, MIP solvers are open frameworks that already are, and will more and more be, sophisticated hybrids.
Heuristics in MIP

The role of (primal) heuristics in MIP solvers is associated with three distinct aspects.

1. Achieving Integer-Feasibility Quickly.
   Finding a first feasible solution is sometimes the main issue when solving a MIP. This is true theoretically because the feasibility problem for MIP is $\mathcal{NP}$-complete, but also from the user’s perspective (as discussed) the solver needs to provide a feasible solution as quick as possible.
Heuristics in MIP

• The role of (primal) heuristics in MIP solvers is associated with three distinct aspects.

1. Achieving Integer-Feasibility Quickly.
   Finding a first feasible solution is sometimes the main issue when solving a MIP. This is true theoretically because the feasibility problem for MIP is $\mathcal{NP}$-complete, but also from the user’s perspective (as discussed) the solver needs to provide a feasible solution as quick as possible.

2. Reaching (quasi-)Optimality Quickly.
   Of course, once a feasible solution is there, the challenge becomes getting better and better ones, and local search heuristics come into the play for that.
Heuristics in MIP

• The role of (primal) heuristics in MIP solvers is associated with three distinct aspects.

1. Achieving Integer-Feasibility Quickly.
   Finding a first feasible solution is sometimes the main issue when solving a MIP. This is true theoretically because the feasibility problem for MIP is $\mathcal{NP}$-complete, but also from the user’s perspective (as discussed) the solver needs to provide a feasible solution as quick as possible.

2. Reaching (quasi-)Optimality Quickly.
   Of course, once a feasible solution is there, the challenge becomes getting better and better ones, and local search heuristics come into the play for that.

3. Analyzing Infeasible MIP solutions.
   During the enumeration tree a large amount of (slightly) infeasible solutions is encountered, either as infeasible nodes in the tree itself, or as a result of the application to a primal heuristic. It is then possible to use heuristics to repair these solutions.
Achieving Integer-Feasibility Quickly

• Let us denote by $\mathcal{B}$ the set of binary variables among the integer-constrained ones in $I$, i.e., $\mathcal{B} \subseteq I$.

• We restrict our attention, for the sake of simplicity, to MIPs in the form (1) in which the all integer-constrained variables are in fact binary, i.e., $I = \mathcal{B}$.

• Let us also denote the continuous relaxation on (1) as

$$P := \{Ax \leq b, x \geq 0\}. \quad (2)$$
Achieving Integer-Feasibility Quickly

• Let us denote by $B$ the set of binary variables among the integer-constrained ones in $I$, i.e., $B \subseteq I$.

• We restrict our attention, for the sake of simplicity, to MIPs in the form (1) in which the all integer-constrained variables are in fact binary, i.e., $I = B$.

• Let us also denote the continuous relaxation on (1) as

$$P := \{Ax \leq b, x \geq 0\}. \quad (2)$$

• We look for an initial feasible solution through an algorithm called Feasibility Pump (FP) [Fischetti, Glover & Lodi 2005].
Feasibility Pump: the basic scheme

- We start from any \( x^* \in P \), and define its rounding \( \tilde{x} \).
Feasibility Pump: the basic scheme

• We start from any \( x^* \in P \), and define its rounding \( \tilde{x} \).

• At each iteration we look for a point \( x^* \in P \) which is as close as possible to the current \( \tilde{x} \) by solving the problem:

\[
\min \{ \Delta(x, \tilde{x}) : x \in P \}
\]

Assuming \( \Delta(x, \tilde{x}) \) is chosen appropriately, it is an easily solvable LP problem.
Feasibility Pump: the basic scheme

- We start from any \( x^* \in P \), and define its rounding \( \tilde{x} \).
- At each iteration we look for a point \( x^* \in P \) which is as close as possible to the current \( \tilde{x} \) by solving the problem:

\[
\min \{ \Delta(x, \tilde{x}) : x \in P \}
\]

Assuming \( \Delta(x, \tilde{x}) \) is chosen appropriately, it is an easily solvable LP problem.
- If \( x_j^* \) is integral \( \forall j \in B \), then \( x^* \) is a feasible MIP solution and we are done.
Feasibility Pump: the basic scheme

• We start from any $x^* \in P$, and define its rounding $\tilde{x}$.

• At each iteration we look for a point $x^* \in P$ which is as close as possible to the current $\tilde{x}$ by solving the problem:

$$\min \{ \Delta(x, \tilde{x}) : x \in P \}$$

Assuming $\Delta(x, \tilde{x})$ is chosen appropriately, it is an easily solvable LP problem.

• If $x_j^*$ is integral $\forall j \in \mathcal{B}$, then $x^*$ is a feasible MIP solution and we are done.

• Otherwise, we replace $\tilde{x}$ by the rounding of $x^*$, and repeat.
Feasibility Pump: the basic scheme

- We start from any $x^* \in P$, and define its rounding $\tilde{x}$.
- At each iteration we look for a point $x^* \in P$ which is as close as possible to the current $\tilde{x}$ by solving the problem:

$$\min \{ \Delta(x, \tilde{x}) : x \in P \}$$

Assuming $\Delta(x, \tilde{x})$ is chosen appropriately, it is an easily solvable LP problem.
- If $x^*_j$ is integral $\forall j \in B$, then $x^*$ is a feasible MIP solution and we are done.
- Otherwise, we replace $\tilde{x}$ by the rounding of $x^*$, and repeat.

- From a geometric point of view, this simple heuristic generates two hopefully convergent trajectories of points $x^*$ and $\tilde{x}$ that satisfy feasibility in a partial but complementary way:
Feasibility Pump: the basic scheme

- We start from any $x^* \in P$, and define its rounding $\tilde{x}$.

- At each iteration we look for a point $x^* \in P$ which is as close as possible to the current $\tilde{x}$ by solving the problem:

  $$\min \{ \Delta(x, \tilde{x}) : x \in P \}$$

  Assuming $\Delta(x, \tilde{x})$ is chosen appropriately, it is an easily solvable LP problem.

- If $x_j^*$ is integral $\forall j \in B$, then $x^*$ is a feasible MIP solution and we are done.

- Otherwise, we replace $\tilde{x}$ by the rounding of $x^*$, and repeat.

- From a geometric point of view, this simple heuristic generates two hopefully convergent trajectories of points $x^*$ and $\tilde{x}$ that satisfy feasibility in a partial but complementary way:
  1. one, $x^*$, satisfies the linear constraints,
Feasibility Pump: the basic scheme

- We start from any $x^* \in P$, and define its rounding $\tilde{x}$.
- At each iteration we look for a point $x^* \in P$ which is as close as possible to the current $\tilde{x}$ by solving the problem:

$$\min\{\Delta(x, \tilde{x}) : x \in P\}$$

Assuming $\Delta(x, \tilde{x})$ is chosen appropriately, it is an easily solvable LP problem.
- If $x_j^*$ is integral $\forall j \in B$, then $x^*$ is a feasible MIP solution and we are done.
- Otherwise, we replace $\tilde{x}$ by the rounding of $x^*$, and repeat.

- From a geometric point of view, this simple heuristic generates two hopefully convergent trajectories of points $x^*$ and $\tilde{x}$ that satisfy feasibility in a partial but complementary way:
  1. one, $x^*$, satisfies the linear constraints,
  2. the other, $\tilde{x}$, the integer requirement.
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration.
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \tilde{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \widetilde{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each iteration
FP: Plot of the infeasibility measure $\Delta(x^*, \tilde{x})$ at each iteration
**FP: Definition of** $\Delta(x^*, \tilde{x})$

- We consider the $L_1$-norm distance between a generic point $x \in P$ and a given integer $\tilde{x}$:

$$\Delta(x, \tilde{x}) = \sum_{j \in \mathcal{B}} |x_j - \tilde{x}_j|$$

The continuous variables $x_j$ with $j \not\in \mathcal{B}$, if any, do not contribute to this function.
FP: Definition of $\Delta(x^*, \tilde{x})$

- We consider the $L_1$-norm distance between a generic point $x \in P$ and a given integer $\tilde{x}$:

$$\Delta(x, \tilde{x}) = \sum_{j \in B} |x_j - \tilde{x}_j|$$

The continuous variables $x_j$ with $j \not\in B$, if any, do not contribute to this function.

- In the case of a binary MIP:

$$\Delta(x, \tilde{x}) := \sum_{j \in B: \tilde{x}_j = 0} x_j + \sum_{j \in B: \tilde{x}_j = 1} (1 - x_j) \quad (3)$$
FP: Definition of $\Delta(x^*, \tilde{x})$

- We consider the $L_1$-norm distance between a generic point $x \in P$ and a given integer $\tilde{x}$:

\[
\Delta(x, \tilde{x}) = \sum_{j \in B} |x_j - \tilde{x}_j|
\]

The continuous variables $x_j$ with $j \notin B$, if any, do not contribute to this function.

- In the case of a binary MIP:

\[
\Delta(x, \tilde{x}) := \sum_{j \in B: \tilde{x}_j = 0} x_j + \sum_{j \in B: \tilde{x}_j = 1} (1 - x_j) \tag{3}
\]

- Given an integer $\tilde{x}$, the closest point $x^* \in P$ can therefore be determined by solving the LP

\[
\min\{\Delta(x, \tilde{x}) : x \in P\}. \tag{4}
\]
FP: Definition of $\Delta(x^*, \tilde{x})$

- We consider the $L_1$-norm distance between a generic point $x \in P$ and a given integer $\tilde{x}$:

$$\Delta(x, \tilde{x}) = \sum_{j \in B} |x_j - \tilde{x}_j|$$

The continuous variables $x_j$ with $j \not\in B$, if any, do not contribute to this function.

- In the case of a binary MIP:

$$\Delta(x, \tilde{x}) := \sum_{j \in B: \tilde{x}_j = 0} x_j + \sum_{j \in B: \tilde{x}_j = 1} (1 - x_j) \quad (3)$$

- Given an integer $\tilde{x}$, the closest point $x^* \in P$ can therefore be determined by solving the LP

$$\min\{\Delta(x, \tilde{x}) : x \in P\}. \quad (4)$$

- We like to see such a distance as a difference of pressure between the two complementary infeasibility of $x^*$ and $\tilde{x}$, that we try to reduce by pumping the integrality of $\tilde{x}$ into $x^*$.
**FP: Definition of $\Delta(x^*, \tilde{x})$**

- We consider the $L_1$-norm distance between a generic point $x \in P$ and a given integer $\tilde{x}$:

$$\Delta(x, \tilde{x}) = \sum_{j \in B} |x_j - \tilde{x}_j|$$

The **continuous variables** $x_j$ with $j \not\in B$, if any, do not contribute to this function.

- In the case of a **binary MIP**:

$$\Delta(x, \tilde{x}) := \sum_{j \in B: \tilde{x}_j = 0} x_j + \sum_{j \in B: \tilde{x}_j = 1} (1 - x_j) \quad (3)$$

- Given an integer $\tilde{x}$, the **closest point** $x^* \in P$ can therefore be determined by solving the LP

$$\min \{ \Delta(x, \tilde{x}) : x \in P \}. \quad (4)$$

- We like to see such a distance as a **difference of pressure** between the two complementary infeasibility of $x^*$ and $\tilde{x}$, that we try to reduce by **pumping** the integrality of $\tilde{x}$ into $x^*$.

- On the other hand, it is clearly a **measure of vicinity**, therefore a neighborhood.
FP: A first implementation

- MAIN PROBLEM, stalling issues:
  as soon as $\Delta(x^*, \tilde{x})$ is not reduced when replacing $\tilde{x}$ by $x^*$.

If $\Delta(x^*, \tilde{x}) > 0$ we still want to modify $\tilde{x}$, even if this increases its distance from $x^*$.

Hence, we reverse the rounding of some variables $x^*_j$, $j \in B$ chosen so as to minimize the increase in the current value of $\Delta(x^*, \tilde{x})$. 


FP: A first implementation

- MAIN PROBLEM, stalling issues:
  as soon as $\Delta(x^*, \tilde{x})$ is not reduced when replacing $\tilde{x}$ by $x^*$.

If $\Delta(x^*, \tilde{x}) > 0$ we still want to modify $\tilde{x}$, even if this increases its distance from $x^*$.

Hence, we reverse the rounding of some variables $x_j^*$, $j \in \mathcal{B}$ chosen so as to minimize the increase in the current value of $\Delta(x^*, \tilde{x})$.

1. initialize nIT := 0 and $x^* := \arg\max\{c^T x : x \in P\}$;
2. if $x^*$ is integer, return $(x^*)$;
3. let $\tilde{x} := [x^*]$ (= rounding of $x^*$);
4. while (time < TL) do
5.   let nIT := nIT + 1 and compute $x^* := \arg\min\{\Delta(x, \tilde{x}) : x \in P\}$;
6.   if $x^*$ is integer, return $(x^*)$;
7.   if $\exists j \in \mathcal{B} : [x_j^*] \neq \tilde{x}_j$ then
8.     $\tilde{x} := [x^*]$
   else
9.     flip the $TT = \text{rand}(T/2, 3T/2)$ entries $\tilde{x}_j$ ($j \in \mathcal{B}$) with highest $|x_j^* - \tilde{x}_j|$ 
10. endif
11. enddo
FP: Plot of the infeasibility measure $\Delta(x^*, \bar{x})$ at each pumping cycle
FP: better dealing with the objective function

• After the initialization of the algorithm in which the optimal solution $x^*$ of the continuous relaxation is computed, the original objective function $c^T x$ is replaced by the distance function $\Delta(x, \tilde{x})$.

• It is clear that if the number of FP iterations is large, especially if some random steps are applied, the trajectory can go very “far away” from the initial $x^*$, with potentially poor values of the original objective function.

• In other words, the search is not guided anymore by $c^T x$. 
**FP: better dealing with the objective function**

- **After the initialization** of the algorithm in which the optimal solution \( x^* \) of the continuous relaxation is computed, the original objective function \( c^T x \) is replaced by the distance function \( \Delta(x, \tilde{x}) \).

- **It is clear that** if the number of FP iterations is large, especially if some random steps are applied, the trajectory can go very “far away” from the initial \( x^* \), with potentially poor values of the original objective function.

- **In other words**, the search is not guided anymore by \( c^T x \).

- **The above issue can be partially corrected** by taking a convex combination of the \( \Delta(x, \tilde{x}) \) and \( c^T x \) objective functions [Achterberg & Berthold 2007].

- **Precisely**, the combination used is

  \[
  \Delta_\alpha(x, \tilde{x}) := (1 - \alpha) \Delta(x, \tilde{x}) - \alpha \frac{\sqrt{|B|}}{||c||_2} c^T x \quad \text{with} \quad \alpha \in [0, 1]
  \]

  (5)

  where \( \alpha \) geometrically decreases at every iteration so as to give more and more emphasis on feasibility with respect to optimality.
FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $I$) requires to redefine the objective function $\Delta(x, \tilde{x})$ appropriately.
FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $I$) requires to redefine the objective function $\Delta(x, \tilde{x})$ appropriately. Indeed, for a variable, say $x_j$, which is not sit at a bound, say either $\ell_j$ or $u_j$, in an integer infeasible solution $\tilde{x}$, it is less trivial to penalize its move.
FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in \( I \)) requires to redefine the objective function \( \Delta(x, \tilde{x}) \) appropriately. Indeed, for a variable, say \( x_j \), which is not sit at a bound, say either \( \ell_j \) or \( u_j \), in an integer infeasible solution \( \tilde{x} \), it is less trivial to penalize its move.

- A possible updated definition [Bertacco, Fischetti & Lodi 2007] is

\[
\Delta(x, \tilde{x}) := \sum_{j \in I: \bar{x}_j = \ell_j} (x_j - \ell_j) + \sum_{j \in I: \bar{x}_j = u_j} (u_j - x_j) + \sum_{j \in I: \ell_j < \bar{x}_j < u_j} d_j
\]

(6)

where the artificial variables \( d_j(=|x_j - \bar{x}_j|) \) satisfy the additional constraints

\[
d_j \geq x_j - \bar{x}_j \quad \text{and} \quad d_j \geq \bar{x}_j - x_j.
\]
FP: dealing with general integer variables

- The extension of the FP algorithm from 0-1 MIPs to general integers (variables in $I$) requires to redefine the objective function $\Delta(x, \widetilde{x})$ appropriately. Indeed, for a variable, say $x_j$, which is not sit at a bound, say either $\ell_j$ or $u_j$, in an integer infeasible solution $\widetilde{x}$, it is less trivial to penalize its move.

- A possible updated definition [Bertacco, Fischetti & Lodi 2007] is

$$\Delta(x, \widetilde{x}) := \sum_{j \in I: \widetilde{x}_j = \ell_j} (x_j - \ell_j) + \sum_{j \in I: \widetilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in I: \ell_j < \widetilde{x}_j < u_j} d_j$$  \hspace{1cm} (6)$$

where the artificial variables $d_j(= |x_j - \widetilde{x}_j|)$ satisfy the additional constraints

$$d_j \geq x_j - \widetilde{x}_j \text{ and } d_j \geq \widetilde{x}_j - x_j.$$ 

- In addition, the FP procedure is split into three parts:
  1. **Binary first**: the integrality requirement of the variables in $I \setminus B$ is relaxed.
  2. **The general integer**: the requirement is reinstalled.
  3. A truncated enumeration phase is performed by solving the MIP with the original constraint and the objective function (6) which uses as $\widetilde{x}$ the “least infeasible” integer solution obtained so far.
The Mixed-Integer NON Linear case

• We consider here a Mixed Integer Non Linear Program of the form:

\[
\begin{aligned}
\text{MINLP} \quad & \min f(x, y) \\
\text{s.t.} \quad & g(x, y) \leq b \\
& x \in \mathbb{Z}^{n_1} \\
& y \in \mathbb{R}^{n_2}
\end{aligned}
\]

where \( f \) is a function from \( \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \) to \( \mathbb{R} \) and \( g \) is a function from \( \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \) to \( \mathbb{R}^m \). We assume the feasible region \( g(x, y) \leq b \) to be convex but we allow the single functions \( g_i \) to be nonconvex.
The Mixed-Integer NON Linear case

• We consider here a Mixed Integer Non Linear Program of the form:

\[
\begin{align*}
\text{MINLP} & \quad \left\{ \begin{array}{l}
\min f(x, y) \\
\text{s.t. :} \\
\quad g(x, y) \leq b \\
\quad x \in \mathbb{Z}^{n_1} \\
\quad y \in \mathbb{R}^{n_2}
\end{array} \right.
\end{align*}
\]

where \( f \) is a function from \( \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \) to \( \mathbb{R} \) and \( g \) is a function from \( \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \) to \( \mathbb{R}^m \). We assume the feasible region \( g(x, y) \leq b \) to be convex but we allow the single functions \( g_i \) to be nonconvex.

• The Outer Approximation (OA) technique linearizes the constraints of the continuous relaxation of MINLP to build a mixed integer linear relaxation of MINLP [Duran & Grossmann, 1986].
The Mixed-Integer NON Linear case

- We consider here a Mixed Integer Non Linear Program of the form:

\[
\text{MINLP} \begin{cases}
\min f(x, y) \\
\text{s.t.: } \\
g(x, y) \leq b \\
x \in \mathbb{Z}^{n_1} \\
y \in \mathbb{R}^{n_2}
\end{cases}
\]

where \( f \) is a function from \( \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \) to \( \mathbb{R} \) and \( g \) is a function from \( \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \) to \( \mathbb{R}^m \). We assume the feasible region \( g(x, y) \leq b \) to be convex but we allow the single functions \( g_i \) to be nonconvex.

- The Outer Approximation (OA) technique linearizes the constraints of the continuous relaxation of MINLP to build a mixed integer linear relaxation of MINLP [Duran & Grossmann, 1986].

- The idea [Bonami, Cornuéjols, Lodi & Margot 2009] is to combine such a linearization technique with a FP-type algorithm to obtain feasible solutions for MINLPs. Let us denote such an algorithm as FP-NLP.
FP–NLP: the basic scheme

• We start by any feasible solution of the continuous relaxation of MINLP, \((\bar{x}^0, \bar{y}^0)\).
FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, \((\bar{x}^0, \bar{y}^0)\).
- In each iteration \(i \geq 1\), we compute \((\hat{x}^i, \hat{y}^i)\) by finding the closest point to \(\bar{x}^{i-1}\) (using the \(L_1\) norm) on the current outer approximation \((OA)^i\) of the problem.
FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, \((\vec{x}^0, \vec{y}^0)\).
- In each iteration \(i \geq 1\), we compute \((\hat{x}^i, \hat{y}^i)\) by finding the closest point to \(\overline{x}^{i-1}\) (using the \(L_1\) norm) on the current outer approximation \((OA)^i\) of the problem:

\[
(FOA)^i \begin{cases}
\min \|x - \overline{x}^{i-1}\|_1 \\
g(\overline{x}^k, \overline{y}^k) + Jg(\overline{x}^k, \overline{y}^k) \left( \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \overline{x}^k \\ \overline{y}^k \end{pmatrix} \right) \leq b \\
x \in \mathbb{Z}^{n_1} \\
y \in \mathbb{R}^{n_2}.
\end{cases}
\]

A. Lodi, How to use your favorite MIP Solver
FP-NLP: the basic scheme

- We start by any feasible solution of the continuous relaxation of MINLP, $(\overline{x}^0, \overline{y}^0)$.
- In each iteration $i \geq 1$, we compute $(\hat{x}^i, \hat{y}^i)$ by finding the closest point to $\overline{x}^{i-1}$ (using the $L_1$ norm) on the current outer approximation $(OA)^i$ of the problem

\[
(FOA)^i \begin{cases} 
\min \|x - \overline{x}^{i-1}\|_1 \\
g(\overline{x}^k, \overline{y}^k) + Jg(\overline{x}^k, \overline{y}^k) \left( \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \overline{x}^k \\ \overline{y}^k \end{pmatrix} \right) \leq b \quad \forall k = 0, \ldots, i - 1 \\
x \in \mathbb{Z}^{n_1} \\
y \in \mathbb{R}^{n_2}.
\end{cases}
\]

- We then compute the next $(\overline{x}^i, \overline{y}^i)$ by solving the NLP
FP-NLP: the basic scheme

• We start by any feasible solution of the continuous relaxation of MINLP, \((\bar{x}^0, \bar{y}^0)\).
• In each iteration \(i \geq 1\), we compute \((\hat{x}^i, \hat{y}^i)\) by finding the closest point to \(\bar{x}^{i-1}\) (using the \(L_1\) norm) on the current outer approximation \((OA)^i\) of the problem

\[
(FOA)^i \begin{cases}
\min \|x - \bar{x}^{i-1}\|_1 \\
g(\bar{x}^k, \bar{y}^k) + Jg(\bar{x}^k, \bar{y}^k) \left( \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \bar{x}^k \\ \bar{y}^k \end{pmatrix} \right) \leq b & \forall k = 0, \ldots, i - 1 \\
x \in \mathbb{Z}^{n_1} \\
y \in \mathbb{R}^{n_2}.
\end{cases}
\]

• We then compute the next \((\bar{x}^i, \bar{y}^i)\) by solving the NLP

\[
(FP-NLP)^i \begin{cases}
\min \|x - \hat{x}^i\|_2 \\
g(x, y) \leq b \\
x \in \mathbb{R}^{n_1} \\
y \in \mathbb{R}^{n_2}.
\end{cases}
\]

• The procedure iterates between solving \((FOA)^i\) and \((FP-NLP)^i\) until either a feasible solution of MINLP is found or \((FOA)^i\) becomes infeasible.
FP-NLP: a pictorial explanation
FP-NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point \((\bar{x}^i, \bar{y}^i)\) the method cannot cycle:
FP–NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point \((\bar{x}^i, \bar{y}^i)\) the method cannot cycle:

  **Thm:** In the basic FP, let \((\hat{x}^i, \hat{y}^i)\) be an optimal solution of \((FOA)^i\) and \((\bar{x}^i, \bar{y}^i)\) an optimal solution of \((FP-NLP)^i\). If the constraint qualification for \((FP-NLP)^i\) holds at \((\bar{x}^i, \bar{y}^i)\), then \(\bar{x}^i \neq \bar{x}^k\) for all \(k = 0, \ldots, i - 1\).
FP–NLP: theoretical results

- The next theorem shows that if constraint qualifications hold at each point \((\bar{x}^i, \bar{y}^i)\) the method cannot cycle:

  **Thm:** In the basic FP, let \((\hat{x}^i, \hat{y}^i)\) be an optimal solution of \((FOA)^i\) and \((\bar{x}^i, \bar{y}^i)\) an optimal solution of \((FP-NLP)^i\). If the constraint qualification for \((FP - NLP)^i\) holds at \((\bar{x}^i, \bar{y}^i)\), then \(\bar{x}^i \neq \bar{x}^k\) for all \(k = 0, \ldots, i - 1\).

- Otherwise, when the point \((\bar{x}^i, \bar{y}^i)\) is not integer feasible and does not satisfy the constraint qualifications of \((FP-NLP)^i\), let \(h(x) = ||x - \hat{x}^i||_2\).
**FP-NLP: theoretical results**

- The next theorem shows that if constraint qualifications hold at each point \((\overline{x}^i, \overline{y}^i)\) the method cannot cycle:

  **Thm:** In the basic FP, let \((\hat{x}^i, \hat{y}^i)\) be an optimal solution of \((\text{FOA})^i\) and \((\overline{x}^i, \overline{y}^i)\) an optimal solution of \((\text{FP-NLP})^i\). If the constraint qualification for \((\text{FP} – \text{NLP})^i\) holds at \((\overline{x}^i, \overline{y}^i)\), then \(\overline{x}^i \neq \overline{x}^k\) for all \(k = 0, \ldots, i - 1\).

- Otherwise, when the point \((\overline{x}^i, \overline{y}^i)\) is not integer feasible and does not satisfy the constraint qualifications of \((\text{FP-NLP})^i\), let \(h(x) = ||x - \hat{x}^i||_2\).

- Then, we add, at iteration \(k\), the following inequality to \((\text{FOA})^i\):

  \[
  (\overline{x}^k - \hat{x}^k)^T(x - \overline{x}^k) \geq 0
  \]  

  (7)
FP-NLP: theoretical results

• The next theorem shows that if constraint qualifications hold at each point \((\bar{x}^i, \bar{y}^i)\) the method cannot cycle:

Thm: In the basic FP, let \((\hat{x}^i, \hat{y}^i)\) be an optimal solution of \((FOA)^i\) and \((\bar{x}^i, \bar{y}^i)\) an optimal solution of \((FP-NLP)^i\). If the constraint qualification for \((FP - NLP)^i\) holds at \((\bar{x}^i, \bar{y}^i)\), then \(\bar{x}^i \neq \bar{x}^k\) for all \(k = 0, \ldots, i - 1\).

• Otherwise, when the point \((\bar{x}^i, \bar{y}^i)\) is not integer feasible and does not satisfy the constraint qualifications of \((FP-NLP)^i\), let \(h(x) = ||x - \hat{x}^i||_2\).

• Then, we add, at iteration \(k\), the following inequality to \((FOA)^i\):

\[
(\bar{x}^k - \hat{x}^k)^T(x - \bar{x}^k) \geq 0
\]  

(7)

• This inequality is valid for the continuous relaxation of MINLP and therefore can be added in the outer approximation constraints in \((FOA)^i\).
**FP-NLP: theoretical results**

- The next theorem shows that if constraint qualifications hold at each point \((\bar{x}^i, \bar{y}^i)\) the method cannot cycle:

  **Thm:** *In the basic FP, let \((\hat{x}^i, \hat{y}^i)\) be an optimal solution of \((FOA)^i\) and \((\bar{x}^i, \bar{y}^i)\) an optimal solution of \((FP-NLP)^i\). If the constraint qualification for \((FP - NLP)^i\) holds at \((\bar{x}^i, \bar{y}^i)\), then \(\bar{x}^i \neq \bar{x}^k\) for all \(k = 0, \ldots, i - 1\).*

- Otherwise, when the point \((\bar{x}^i, \bar{y}^i)\) is not integer feasible and does not satisfy the constraint qualifications of \((FP-NLP)^i\), let \(h(x) = ||x - \hat{x}^i||_2\).
- Then, we add, at iteration \(k\), the following inequality to \((FOA)^i\):

  \[
  (\bar{x}^k - \hat{x}^k)^T (x - \bar{x}^k) \geq 0 \tag{7}
  \]

  - This inequality is valid for the continuous relaxation of MINLP and therefore can be added in the outer approximation constraints in \((FOA)^i\).
- Then, we can prove that:

  **Thm:** *If the integer variables \(x\) are bounded, the algorithm enhanced by constraints \((7)\) terminates in a finite number of iterations.*
Reaching (quasi-)Optimality Quickly

- We again restrict our initial attention to the special case in which $I = B$.

- We now assume to have a feasible solution, $\bar{x}$ at hand, so-called reference solution (generally the incumbent, i.e., the best solution encountered so far).
Reaching (quasi-)Optimality Quickly

- We again restrict our initial attention to the special case in which $I = B$.

- We now assume to have a feasible solution, $\bar{x}$ at hand, so-called reference solution (generally the incumbent, i.e., the best solution encountered so far).

- We the look for better and better solutions through the Local Branching algorithm (LB) [Fischetti & Lodi 2002].
**The Local Branching framework**

- Given reference solution \( \bar{x} \), let \( \bar{S} := \{ j \in \mathcal{B} : \bar{x}_j = 1 \} \) denote the binary support of \( \bar{x} \).
The Local Branching framework

• Given reference solution $\bar{x}$, let $\overline{S} := \{ j \in B : \bar{x}_j = 1 \}$ denote the binary support of $\bar{x}$.

• For a given positive integer parameter $k$, we define the $k$-OPT neighborhood $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (1) satisfying the additional local branching constraint

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in B \setminus \overline{S}} x_j$$
The Local Branching framework

• Given reference solution $\bar{x}$, let $\overline{S} := \{ j \in B : \bar{x}_j = 1 \}$ denote the binary support of $\bar{x}$.

• For a given positive integer parameter $k$, we define the $k$-OPT neighborhood $N(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (1) satisfying the additional local branching constraint

$$
\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in B \setminus \overline{S}} x_j \leq k
$$

where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.
The Local Branching framework

- Given reference solution $\bar{x}$, let $\bar{S} := \{j \in B : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.

- For a given positive integer parameter $k$, we define the $k$-OPT neighborhood $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (1) satisfying the additional local branching constraint

$$\Delta(x, \bar{x}) := \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in B \setminus \bar{S}} x_j \leq k$$

(8)

where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.

- Constraint (8) imposes a maximum Hamming distance of $k$ among the feasible neighbors of $\bar{x}$ and can be used as a branching criterion within an enumerative scheme for (1):
The Local Branching framework

- Given reference solution \( \bar{x} \), let \( \bar{S} := \{ j \in \mathcal{B} : \bar{x}_j = 1 \} \) denote the binary support of \( \bar{x} \).

- For a given positive integer parameter \( k \), we define the \( k\text{-OPT neighborhood} \) \( \mathcal{N}(\bar{x}, k) \) of \( \bar{x} \) as the set of the feasible solutions of (1) satisfying the additional \textit{local branching constraint}:

\[
\Delta(x, \bar{x}) := \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \bar{S}} x_j \leq k
\]  

(8)

where the two terms in left-hand side count the number of binary variables flipping their value (with respect to \( \bar{x} \)) either from 1 to 0 or from 0 to 1, respectively.

- Constraint (8) imposes a maximum Hamming distance of \( k \) among the feasible neighbors of \( \bar{x} \) and can be used as a branching criterion within an enumerative scheme for (1):

\[
\Delta(x, \bar{x}) \leq k \quad \text{(left branch)}
\]
The Local Branching framework

- Given reference solution \( \bar{x} \), let \( \overline{S} := \{ j \in B : \bar{x}_j = 1 \} \) denote the binary support of \( \bar{x} \).

- For a given positive integer parameter \( k \), we define the \( k\text{-OPT neighborhood} \mathcal{N}(\bar{x}, k) \) of \( \bar{x} \) as the set of the feasible solutions of (1) satisfying the additional local branching constraint

\[
\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in B \setminus \overline{S}} x_j \leq k
\]  

(8)

where the two terms in left-hand side count the number of binary variables flipping their value (with respect to \( \bar{x} \)) either from 1 to 0 or from 0 to 1, respectively.

- Constraint (8) imposes a maximum Hamming distance of \( k \) among the feasible neighbors of \( \bar{x} \) and can be used as a branching criterion within an enumerative scheme for (1):

\[
\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \quad \text{or} \quad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)}
\]

(9)
The Local Branching framework

- Given reference solution $\bar{x}$, let $\overline{S} := \{j \in B : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$.

- For a given positive integer parameter $k$, we define the $k$-OPT neighborhood $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (1) satisfying the additional local branching constraint

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in B \setminus \overline{S}} x_j \leq k$$

where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.

- Constraint (8) imposes a maximum Hamming distance of $k$ among the feasible neighbors of $\bar{x}$ and can be used as a branching criterion within an enumerative scheme for (1):

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \quad \text{or} \quad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)}$$

- The neighborhoods defined by the local branching constraints can be searched by using, as a black-box, a MIP solver.
**LB: the basic scheme**

1. initial solution $\bar{x}^1$
LB: the basic scheme

1. initial solution $\bar{x}^1$

$\Delta(x, \bar{x}^1) \leq k$

2.
**LB: the basic scheme**

1. Initial solution $\bar{x}^1$

$$\Delta(x, \bar{x}^1) \leq k$$

2. Improved solution $\bar{x}^2$
**LB: the basic scheme**

![Diagram]

- **1. Initial solution** $\bar{x}^1$
  - $\Delta(x, \bar{x}^1) \leq k$

- **2. Improved solution** $\bar{x}^2$
  - $\Delta(x, \bar{x}^1) \geq k + 1$
**LB: the basic scheme**

1. Initial solution $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]

2. Improved solution $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]

3. 

A. Lodi, How to use your favorite MIP Solver
**LB: the basic scheme**

Initial solution $\bar{x}^1$

$$\Delta(x, \bar{x}^1) \leq k$$

Improved solution $\bar{x}^2$

$$\Delta(x, \bar{x}^2) \leq k$$

Improved solution $\bar{x}^3$

$$\Delta(x, \bar{x}^1) \geq k + 1$$
**LB: the basic scheme**

1. **Initial solution** $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]

2. **Improved solution** $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]
   \[ \Delta(x, \bar{x}^2) \geq k + 1 \]

3. **Improved solution** $\bar{x}^3$
   \[ \Delta(x, \bar{x}^3) \leq k \]
   \[ \Delta(x, \bar{x}^3) \geq k + 1 \]
**LB: the basic scheme**

1. Initial solution $\bar{x}^1$
   - $\Delta(x, \bar{x}^1) \leq k$
   - $\Delta(x, \bar{x}^1) \geq k + 1$

2. Improved solution $\bar{x}^2$
   - $\Delta(x, \bar{x}^2) \leq k$
   - $\Delta(x, \bar{x}^2) \geq k + 1$

3. Improved solution $\bar{x}^3$
   - $\Delta(x, \bar{x}^3) \leq k$
   - $\Delta(x, \bar{x}^3) \geq k + 1$
LB: the basic scheme

1. Initial solution $\bar{x}^1$

$\Delta(x, \bar{x}^1) \leq k$

$\Delta(x, \bar{x}^1) \geq k + 1$

2. Improved solution $\bar{x}^2$

$\Delta(x, \bar{x}^2) \leq k$

$\Delta(x, \bar{x}^2) \geq k + 1$

3. Improved solution $\bar{x}^3$

$\Delta(x, \bar{x}^3) \leq k$

4. No improved solution

$\Delta(x, \bar{x}^3) \leq k$
**LB: the basic scheme**

1. **Initial solution** $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]
   \[ \Delta(x, \bar{x}^1) \geq k + 1 \]

2. **Improved solution** $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]
   \[ \Delta(x, \bar{x}^2) \geq k + 1 \]

3. **Improved solution** $\bar{x}^3$
   \[ \Delta(x, \bar{x}^3) \leq k \]
   \[ \Delta(x, \bar{x}^3) \geq k + 1 \]

4. **No improved solution**

A. Lodi, How to use your favorite MIP Solver
**LB: the basic scheme**

1. Initial solution $\bar{x}^1$
   - $\Delta(x, \bar{x}^1) \leq k$
   - $\Delta(x, \bar{x}^1) \geq k + 1$

2. Improved solution $\bar{x}^2$
   - $\Delta(x, \bar{x}^2) \leq k$
   - $\Delta(x, \bar{x}^2) \geq k + 1$

3. Improved solution $\bar{x}^3$
   - $\Delta(x, \bar{x}^3) \leq k$
   - $\Delta(x, \bar{x}^3) \geq k + 1$

4. No improved solution

5. No improved solution
Solving MIP instance \texttt{tr24-15} (solution value vs. CPU seconds)
Solving MIP instance tr24-15 (solution value vs. CPU seconds)
Solving MIP instance tr24-15 (solution value vs. CPU seconds)
Solving MIP instance tr24-15 (solution value vs. CPU seconds)
**LB: the heuristic version, time limit at nodes**

- The previous scheme can be enhanced from a heuristic viewpoint in two ways.
LB: the heuristic version, time limit at nodes

- The previous scheme can be enhanced from a heuristic viewpoint in two ways.
  - Imposing a time/node limit on the left-branch nodes.
The previous scheme can be enhanced from a heuristic viewpoint in two ways.

- Imposing a time/node limit on the left-branch nodes.

In some cases, the exact solution of the left-branch node can be too time consuming for the value of the parameter $k$ at hand. Hence, from the point of view of a heuristic, it is reasonable to impose a time/node limit for the left-branch computation.
The previous scheme can be enhanced from a heuristic viewpoint in two ways.

- Imposing a time/node limit on the left-branch nodes.

In some cases, the exact solution of the left-branch node can be too time consuming for the value of the parameter $k$ at hand. Hence, from the point of view of a heuristic, it is reasonable to impose a time/node limit for the left-branch computation.

In case the time limit is exceeded, we have two cases:

* **Case (a):**
  
  If the incumbent solution has been improved, we backtrack to the father node and create a new left-branch node associated with the new incumbent solution, without modifying the value of parameter $k$. 

```
The previous scheme can be enhanced from a heuristic viewpoint in two ways.

– Imposing a time/node limit on the left-branch nodes.

In some cases, the exact solution of the left-branch node can be too time consuming for the value of the parameter $k$ at hand. Hence, from the point of view of a heuristic, it is reasonable to impose a time/node limit for the left-branch computation.

In case the time limit is exceeded, we have two cases:

* **Case (a):**
  
  If the incumbent solution has been improved, we backtrack to the father node and create a new left-branch node associated with the new incumbent solution, without modifying the value of parameter $k$.

* **Case (b):**
  
  If the time limit is reached with no improved solution, instead, we reduce the size of the neighborhood in an attempt to speed-up its exploration. This is obtained by reducing the right-hand side term by, e.g., $\lceil k/2 \rceil$. 

Working with a node time limit: case (a)

1. initial solution $\bar{x}^1$
Working with a node time limit: case (a)

\[ \Delta(x, \bar{x}^1) \leq k \]

Initial solution \( \bar{x}^1 \)
Working with a node time limit: case (a)

1. initial solution $\bar{x}^1$

2. $\Delta(x, \bar{x}^1) \leq k$

improved solution $\bar{x}^2$
Working with a node time limit: case (a)

1. initial solution \( \bar{x}^1 \)

\[ \Delta(x, \bar{x}^1) \leq k \]

\[ \Delta(x, \bar{x}^1) \geq k + 1 \]

2. T

\[ \Delta(x, \bar{x}^2) \leq k \]

improved solution \( \bar{x}^2 \)

3. T

\[ \Delta(x, \bar{x}^2) \leq k \]
Working with a node time limit: case (a)

\[ \Delta(x, \bar{x}^1) \leq k \]
\[ \Delta(x, \bar{x}^1) \geq k + 1 \]

\[ \Delta(x, \bar{x}^2) \leq k \]

improved solution \( \bar{x}^2 \)

\[ \Delta(x, \bar{x}^3) \leq k \]

time limit reached, improved solution \( \bar{x}^3 \)
Working with a node time limit: case (a)

1. Initial solution $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]

2. Improved solution $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]
   \[ \Delta(x, \bar{x}^1) \geq k + 1 \]

3. Time limit reached, improved solution $\bar{x}^3$
   \[ \Delta(x, \bar{x}^3) \leq k \]
Working with a node time limit: case (a)

1. Initial solution $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]
   \[ \Delta(x, \bar{x}^1) \geq k + 1 \]

2. Improved solution $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]

3. Time limit reached, improved solution $\bar{x}^3$
   \[ \Delta(x, \bar{x}^3) \leq k \]

3'. Improved solution $\bar{x}^4$
   \[ \Delta(x, \bar{x}^3) \leq k \]
Working with a node time limit: case (a)

1. Initial solution $\bar{x}^1$
   - $\Delta(x, \bar{x}^1) \leq k$
   - $\Delta(x, \bar{x}^1) \geq k + 1$

2. Improved solution $\bar{x}^2$
   - $\Delta(x, \bar{x}^2) \leq k$

3. Time limit reached, improved solution $\bar{x}^3$
   - $\Delta(x, \bar{x}^3) \leq k$

3'. Improved solution $\bar{x}^4$
   - $\Delta(x, \bar{x}^3) \geq k + 1$
Working with a node time limit: case (b)

1. initial solution $x^1$
Working with a node time limit: case (b)

\[ \Delta(x, \bar{x}^1) \leq k \]

initial solution \( \bar{x}^1 \)
Working with a node time limit: case (b)

\[ \Delta(x, \bar{x}^1) \leq k \]

initial solution \( \bar{x}^1 \)

improved solution \( \bar{x}^2 \)
Working with a node time limit: case (b)

- Initial solution $\bar{x}^1$
  - $\Delta(x, \bar{x}^1) \leq k$
  - $\Delta(x, \bar{x}^1) \geq k + 1$

- Improved solution $\bar{x}^2$
  - $\Delta(x, \bar{x}^2) \leq k$
Working with a node time limit: case (b)

1. Initial solution $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]

2. Improved solution $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]

3. Time limit reached, no improved solution
   \[ \Delta(x, \bar{x}^1) \geq k + 1 \]
Working with a node time limit: case (b)

1. Initial solution $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]
   \[ \Delta(x, \bar{x}^1) \geq k + 1 \]

2. Improved solution $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]

3. Time limit reached, no improved solution
   \[ \Delta(x, \bar{x}^2) \leq \lfloor \frac{k}{2} \rfloor \]
Working with a node time limit: case (b)

1. Initial solution $\bar{x}^1$
   - $\Delta(x, \bar{x}^1) \leq k$
   - $\Delta(x, \bar{x}^1) \geq k + 1$

2. Improved solution $\bar{x}^2$
   - $\Delta(x, \bar{x}^2) \leq k$

3. Time limit reached, no improved solution

3'. Improved solution $\bar{x}^3$
   - $\Delta(x, \bar{x}^2) \leq \lfloor \frac{k}{2} \rfloor$
Working with a node time limit: case (b)

1. initial solution $\bar{x}^1$
   \[ \Delta(x, \bar{x}^1) \leq k \]

2. improved solution $\bar{x}^2$
   \[ \Delta(x, \bar{x}^2) \leq k \]

3. time limit reached, no improved solution
   \[ \Delta(x, \bar{x}^2) \leq \lfloor \frac{k}{2} \rfloor \]

4. improved solution $\bar{x}^3$
   \[ \Delta(x, \bar{x}^2) \geq \lfloor \frac{k}{2} \rfloor + 1 \]

...
LB: the heuristic version, diversification

- The second enhancement from a heuristic viewpoint is
  - Diversification
LB: the heuristic version, diversification

- The second enhancement from a heuristic viewpoint is
  - Diversification

  Two mechanisms are applied in the spirit of metaheuristic techniques.
The second enhancement from a heuristic viewpoint is

- Diversification

Two mechanisms are applied in the spirit of metaheuristic techniques.

* Soft diversification.

We first apply a “soft” action consisting in enlarging the current neighborhood by increasing its size by, e.g., \( \lceil k/2 \rceil \).

A new “left-branch” is then explored and in case no improved solution is found even in the enlarged neighborhood (within the time limit), we apply a stronger action in the spirit of Variable Neighborhood Search [Mladenović & Hansen 1997].
LB: the heuristic version, diversification

- The second enhancement from a heuristic viewpoint is
  - Diversification

  Two mechanisms are applied in the spirit of metaheuristic techniques.
  
  * Soft diversification.
    
    We first apply a “soft” action consisting in enlarging the current neighborhood by increasing its size by, e.g., \( \lceil \frac{k}{2} \rceil \).

    A new “left-branch” is then explored and in case no improved solution is found even in the enlarged neighborhood (within the time limit), we apply a stronger action in the spirit of Variable Neighborhood Search [Mladenović & Hansen 1997].

  * Strong diversification.

    We look for a solution (typically worse than the incumbent one) which is not “too far” from the current reference solution.

    We apply tactical branching to the current problem amended by \( \Delta(x, \bar{x}^i) \leq k + 2\lceil \frac{k}{2} \rceil \), but without imposing any upper bound on the optimal solution value.

    The exploration is aborted as soon as the first feasible solution is found.

    This solution (typically worse than the current best one) is used as the new reference one.
LB: solution value vs. CPU seconds for instance B1C1S1
LB: solution value vs. CPU seconds for instance B1C1S1
LB: dealing with general integer variables

- Local branching is based on the assumption that $B \neq \emptyset$, i.e., there is a set of binary variables, and moreover, this set is highly important.

- According to the computational experience, this is true even in case of MIPs involving general integer variables, in that the 0-1 variables (which are often associated with big-M terms) are often largely responsible for the difficulty of the model.
**LB: dealing with general integer variables**

- Local branching is based on the assumption that $\mathcal{B} \neq \emptyset$, i.e., there is a set of binary variables, and moreover, this set is highly important.

- According to the computational experience, this is true even in case of MIPs involving general integer variables, in that the 0-1 variables (which are often associated with big-M terms) are often largely responsible for the difficulty of the model.

- However, in the general case of integer variables $x_j$ with lower and upper bounds $\ell_j$ and $u_j$, respectively, the local branching constraint can be written as

  \[
  \Delta_1(x, \bar{x}) := \sum_{j \in I : \bar{x}_j = \ell_j} \mu_j (x_j - \ell_j) + \sum_{j \in I : \bar{x}_j = u_j} \mu_j (u_j - x_j) + \sum_{j \in I : \ell_j < \bar{x}_j < u_j} \mu_j (x_j^+ + x_j^-) \leq k
  \]

  where weights $\mu_j$ are defined, e.g., as $\mu_j = 1/(u_j - \ell_j)$ $\forall j \in I$, while the variation terms $x_j^+$ and $x_j^-$ require the introduction into the MIP model of additional constraints of the form

  \[x_j = \bar{x}_j + x_j^+ - x_j^-, \quad x_j^+ \geq 0, \quad x_j^- \geq 0, \quad \forall j \in I : \ell_j < \bar{x}_j < u_j.\]
LB: implementation within a MIP solver

- The (final) scheme above enhances (or at least enhanced back in 2002) the heuristic behavior of the MIP solver at the price of giving up optimality.

- On the other hand, it is easy to see that an alternative implementation would be within the branch-and-cut tree of a MIP solver, in the line of a classical local search improvement procedure.


**LB: implementation within a MIP solver**

- The (final) scheme above enhances (or at least enhanced back in 2002) the heuristic behavior of the MIP solver at the price of giving up optimality.

- On the other hand, it is easy to see that an alternative implementation would be within the branch-and-cut tree of a MIP solver, in the line of a classical local search improvement procedure.

More precisely, whenever a new feasible solution has been found, an alternative MIP amended by the LB constraint \((MIP\text{ping})\) is searched by branch-and-cut for a fixed number of nodes in the aim of improving such a solution.
The (final) scheme above enhances (or at least enhanced back in 2002) the heuristic behavior of the MIP solver at the price of giving up optimality.

On the other hand, it is easy to see that an alternative implementation would be within the branch-and-cut tree of a MIP solver, in the line of a classical local search improvement procedure.

More precisely, whenever a new feasible solution has been found, an alternative MIP amended by the LB constraint (MIPping) is searched by branch-and-cut for a fixed number of nodes in the aim of improving such a solution.

This is indeed the way in which it is implemented in Cplex since version 9.
Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the solution of the continuous relaxation, say $x^*$, at a given node of the branch-and-bound tree [Danna, Rothberg & Le Pape 2005].

- Then, $\bar{x}$ and $x^*$ are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.
Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the solution of the continuous relaxation, say $x^*$, at a given node of the branch-and-bound tree [Danna, Rothberg & Le Pape 2005].

- Then, $\bar{x}$ and $x^*$ are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.

- The advantages of RINS with respect to LB are essentially two.
Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, \( \bar{x} \), and the solution of the continuous relaxation, say \( x^* \), at a given node of the branch-and-bound tree [Danna, Rothberg & Le Pape 2005].

- Then, \( \bar{x} \) and \( x^* \) are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, \( \bar{x} \) is updated in the rest of the tree.

- The advantages of RINS with respect to LB are essentially two.
  1. On the one side, because the solution of the continuous relaxation \( x^* \) changes at each branch-and-bound node, RINS can be invoked more often, potentially every node, thus leading, in principle, to faster improvements.
Relaxation Induced Neighborhood Search (RINS)

- In the spirit of solving an associated MIP to improve the incumbent, an enhancement has been proposed which takes simultaneously into account both the incumbent solution, $\bar{x}$, and the solution of the continuous relaxation, say $x^*$, at a given node of the branch-and-bound tree [Danna, Rothberg & Le Pape 2005].

- Then, $\bar{x}$ and $x^*$ are compared and all the binary variables which assume the same value are hard-fixed in an associated MIP.

- This associated MIP is then solved by using the MIP solver as a black-box, and in case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.

- The advantages of RINS with respect to LB are essentially two.
  1. On the one side, because the solution of the continuous relaxation $x^*$ changes at each branch-and-bound node, RINS can be invoked more often, potentially every node, thus leading, in principle, to faster improvements.
  2. Moreover, and more crucially, the sub-MIPs explored by RINS are generally smaller than those of LB because the former fixes variables (thus removing them from the MIP) while the latter imposes a soft fixing based on the addition of a linear inequality.
Analyzing (and Repairing) Infeasible MIP solutions

- Sometimes waiting to have a fully feasible solution before starting a local search approach, for example, within a branch-and-bound tree, is unnecessary.
Analyzing (and Repairing) Infeasible MIP solutions

- Sometimes waiting to have a fully feasible solution before starting a local search approach, for example, within a branch-and-bound tree, is unnecessary.

- In addition, during the exploration of the enumeration tree, one can collect solutions which, although “slightly” infeasible, are potentially of good quality.

- These solutions are (generally) simply discarded while a careful use of local search could lead to repair them and driving to feasibility.
Analyzing (and Repairing) Infeasible MIP solutions

- Sometimes waiting to have a fully feasible solution before starting a local search approach, for example, within a branch-and-bound tree, is unnecessary.

- In addition, during the exploration of the enumeration tree, one can collect solutions which, although “slightly” infeasible, are potentially of good quality.

- These solutions are (generally) simply discarded while a careful use of local search could lead to repair them and driving to feasibility.

- One flexible way of doing this is combining FP and LB [Fischetti & Lodi 2008].
Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\tilde{x}$ with minimum distance $\Delta$ to a feasible solution $x^*$ of the continuous relaxation of (1) is stored.
Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\tilde{x}$ with minimum distance $\Delta$ to a feasible solution $x^*$ of the continuous relaxation of (1) is stored.

- Besides the value of $\Delta(x, \tilde{x})$, the infeasibility of $\tilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

- Let us call $T$ the set of the indices of the violated constraints.
Combining FP and LB

- FP is executed for a limited number of iterations and the integer (infeasible) solution $\tilde{x}$ with minimum distance $\Delta$ to a feasible solution $x^*$ of the continuous relaxation of (1) is stored.

- Besides the value of $\Delta(x, \tilde{x})$, the infeasibility of $\tilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

- Let us call $T$ the set of the indices of the violated constraints.

- Clearly, in the spirit of phase 1 of the Simplex algorithm, one can modify the original MIP to make such a $\tilde{x}$ feasible by
Combining FP and LB

• FP is executed for a limited number of iterations and the integer (infeasible) solution \( \tilde{x} \) with minimum distance \( \Delta \) to a feasible solution \( x^* \) of the continuous relaxation of (1) is stored.

• Besides the value of \( \Delta(x, \tilde{x}) \), the infeasibility of \( \tilde{x} \) can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

• Let us call \( T \) the set of the indices of the violated constraints.

• Clearly, in the spirit of phase 1 of the Simplex algorithm, one can modify the original MIP to make such a \( \tilde{x} \) feasible by

1. subtracting a continuous non-negative artificial variable \( s_i \) to each constraint \( i \in T \), and
Combining FP and LB

• FP is executed for a limited number of iterations and the integer (infeasible) solution $\tilde{x}$ with minimum distance $\Delta$ to a feasible solution $x^*$ of the continuous relaxation of (1) is stored.

• Besides the value of $\Delta(x, \tilde{x})$, the infeasibility of $\tilde{x}$ can be combinatorially measured in terms of the number of constraints of the original MIP which violates.

• Let us call $T$ the set of the indices of the violated constraints.

• Clearly, in the spirit of phase 1 of the Simplex algorithm, one can modify the original MIP to make such a $\tilde{x}$ feasible by

  1. subtracting a continuous non-negative artificial variable $s_i$ to each constraint $i \in T$, and
  2. adding the constraint $s_i \leq My_i$ with $y_i \in \{0, 1\}$.

• The artificial variable $s_i$ is used to satisfy the constraint but if and only if the corresponding $y_i$ variable is set to 1 ($M$ is a classical big-M coefficient).
Combining FP and LB (cont.d)

- LB starts by using $\tilde{x}$ as a reference solution and replacing the original objective function with

$$\min \sum_{i \in T} y_i$$

- The solution $\tilde{x}$ is clearly feasible for such a new MIP and its objective function value is $|T|$. 
Combining FP and LB (cont.d)

• LB starts by using $\tilde{x}$ as a reference solution and replacing the original objective function with

$$\min \sum_{i \in T} y_i$$

• The solution $\tilde{x}$ is clearly feasible for such a new MIP and its objective function value is $|T|$.

• Hence, in a first phase, LB attempts to improve the current infeasible solution by reducing the number of infeasible constraints in the spirit of the first phase of the Simplex algorithm.
Combining FP and LB (cont.d)

- LB starts by using $\tilde{x}$ as a reference solution and replacing the original objective function with
  \[
  \min \sum_{i \in T} y_i
  \]

- The solution $\tilde{x}$ is clearly feasible for such a new MIP and its objective function value is $|T|$.

- Hence, in a first phase, LB attempts to improve the current infeasible solution by reducing the number of infeasible constraints in the spirit of the first phase of the Simplex algorithm.

- In the second phase, once a feasible solution has been found, the original objective function is then restored and LB takes care of improving the quality of such a solution.
Using MIP heuristics in Applications: Matheuristics

- Recently, a new name/concept in the world of heuristic and metaheuristic techniques has been introduced [Maniezzo 2006], namely matheuristics.
Using MIP heuristics in Applications: Matheuristics

• Recently, a new name/concept in the world of heuristic and metaheuristic techniques has been introduced [Maniezzo 2006], namely matheuristics.

• From Wikipedia:
  “Matheuristics are optimization algorithms made by the interoperation of metaheuristics and mathematical programming (generally MIP) techniques. An essential feature is the exploitation in some part of the algorithms of features derived from the mathematical model of the problems of interest, thus the definition “model-based metaheuristics” appearing in the title of some events of the conference series dedicated to matheuristics”

http://astarte.csr.unibo.it/Matheuristics/
Using MIP heuristics in Applications: Matheuristics

- Recently, a new name/concept in the world of heuristic and metaheuristic techniques has been introduced [Maniezzo 2006], namely matheuristics.

- From Wikipedia:
  "Matheuristics are optimization algorithms made by the interoperation of metaheuristics and mathematical programming (generally MIP) techniques. An essential feature is the exploitation in some part of the algorithms of features derived from the mathematical model of the problems of interest, thus the definition “model-based metaheuristics” appearing in the title of some events of the conference series dedicated to matheuristics"

  [http://astarte.csr.unibo.it/Matheuristics/](http://astarte.csr.unibo.it/Matheuristics/)

- Of course, the use of MIP for heuristic solution of optimization problems is much older and much more widespread than matheuristics, but this is not the case for metaheuristics.

- Even the idea of designing MIP methods specifically for finding heuristic solutions has innovative traits, when opposed to exact methods that turn into heuristics when enough computational resources are not available.
Matheuristics (cont.d)

- Two main types of algorithms have been devised:
  
  1. MIP as a subroutine for known metaheuristics, and  
  2. MIP as a paradigm for new metaheuristics.
Matheuristics (cont.d)

- Two main types of algorithms have been devised:

  1. MIP as a subroutine for known metaheuristics, and
  2. MIP as a paradigm for new metaheuristics.

- In both cases a common key step prescribes to collect possible components of the problem solution and to include them in a MIP formulation, often as columns of a set-partitioning formulation, possibly with additional constraints.

- The resulting restricted MIP formulation is then solved in an exact or heuristic way.
MIP as a subroutine for known metaheuristics

• The use of MIP subroutines within metaheuristics has been essentially tried in all contexts. A few examples are:
MIP as a subroutine for known metaheuristics

- The use of MIP subroutines within metaheuristics has been essentially tried in all contexts. A few examples are:
  
  - Genetic and Evolutionary algorithms:
    fixing the solution parts common to both parents and optimizing the rest [Rothberg 2007].
MIP as a subroutine for known metaheuristics

- The use of MIP subroutines within metaheuristics has been essentially tried in all contexts. A few examples are:

  - Genetic and Evolutionary algorithms:
    fixing the solution parts common to both parents and optimizing the rest [Rothberg 2007].

  - Variable Neighborhood Search:
    relaxations of the neighborhoods are solved to decide the ordering in which they are explored [Puchinger & Raidl 2008]; combination of Variable Neighborhood Search and RINS [Mladenovic et al. 2009]; a combination of an integer multicommodity flow optimization component and a variable neighborhood search component is used in scheduling of transportation of concrete [Schmid, Doerner, Hartl, Savelsbergh & Stoecher 2009].
• The use of MIP subroutines within metaheuristics has been essentially tried in all contexts. A few examples are:

  – **Genetic and Evolutionary algorithms:**
    fixing the solution parts common to both parents and optimizing the rest [Rothberg 2007].

  – **Variable Neighborhood Search:**
    relaxations of the neighborhoods are solved to decide the ordering in which they are explored [Puchinger & Raidl 2008]; combination of Variable Neighborhood Search and RINS [Mladenovic et al. 2009]; a combination of an integer multicommodity flow optimization component and a variable neighborhood search component is used in scheduling of transportation of concrete [Schmid, Doerner, Hartl, Savelsbergh & Stoecher 2009].

  – **Ant Colony Optimization:**
    lower bounds are used to assess the likelihood of success of the possible ants moves [Maniezzo 1999]; other applications combine Ant Colony Optimization and partial enumeration, precisely Beam-Search [Blum 2005].
MIP as a subroutine for known metaheuristics (cont.d)

- More examples are:
MIP as a subroutine for known metaheuristics (cont.d)

● More examples are:

  – **GRASP:**
    in an application to a particular transfer line balancing problem, MIP is used both in the construction and in the improvement phase [Dolgui, Eremeev & Guschinskaya 2009].
MIP as a subroutine for known metaheuristics (cont.d)

- More examples are:

  - **GRASP:**
    in an application to a particular transfer line balancing problem, MIP is used both in the construction and in the improvement phase [Dolgui, Eremeev & Guschinskaya 2009].

  - **Tabu Search:**
    a special vehicle routing problem in which a perfect $b$-matching is used to define the candidate sets to be passed to a granular tabu search algorithm (i.e., a TS where only a subset of neighbors is checked) [Ulrich-Ngueveu, Prins & Wolfler-Calvo 2009].
MIP as a subroutine for known metaheuristics (cont.d)

- More examples are:
  
  - **GRASP:**
    in an application to a particular transfer line balancing problem, MIP is used both in the construction and in the improvement phase [Dolgui, Eremeev & Guschinskaya 2009].

  - **Tabu Search:**
    a special vehicle routing problem in which a perfect $b$-matching is used to define the candidate sets to be passed to a granular tabu search algorithm (i.e., a TS where only a subset of neighbors is checked) [Ulrich-Ngueveu, Prins & Wolfler-Calvo 2009].

  - **Path Relinking:**
    neighborhoods of the current incumbent solution are defined by using information of the previous solutions and explored by MIP [Beasley 2006].
MIP as a subroutine for known metaheuristics (cont.d)

More examples are:

- **GRASP:**
  in an application to a particular transfer line balancing problem, MIP is used both in the construction and in the improvement phase [Dolgui, Eremeev & Guschinskaya 2009].

- **Tabu Search:**
  a special vehicle routing problem in which a perfect $b$-matching is used to define the candidate sets to be passed to a granular tabu search algorithm (i.e., a TS where only a subset of neighbors is checked) [Ulrich-Ngueveu, Prins & Wolfler-Calvo 2009].

- **Path Relinking:**
  neighborhoods of the current incumbent solution are defined by using information of the previous solutions and explored by MIP [Beasley 2006].

- **Large Neighborhood Search:**
  an hybrid framework combines Large Neighborhood Search and Column Generation with columns generated heuristically through Variable Neighborhood Search [Parragh & Schmid 2011].
MIP as a paradigm for “new” metaheuristics

• Here we consider to be new a metaheuristic that could not exist without the MIP contribution.
MIP as a paradigm for “new” metaheuristics

• Here we consider to be new a metaheuristic that could not exist without the MIP contribution.

• Some of these algorithms are probably the roots of matheuristics, for example

  – Very Large Neighborhood Search [Ahuja, Orlin & Sharma 2000],
  – Dynasearch [Congram, Potts & van de Velde 2002],
  – Local Branching [Fischetti & Lodi 2002],

were the first methods to explore exponentially-large neighborhoods by means of Mathematical Programming techniques, thus considerably extending the context of local search heuristics, and giving rise to a new type of metaheuristics.
MIP as a paradigm for “new” metaheuristics

• Here we consider to be new a metaheuristic that could not exist without the MIP contribution.

• Some of these algorithms are probably the roots of matheuristics, for example
  – Very Large Neighborhood Search [Ahuja, Orlin & Sharma 2000],
  – Dynasearch [Congram, Potts & van de Velde 2002],
  – Local Branching [Fischetti & Lodi 2002],

were the first methods to explore exponentially-large neighborhoods by means of Mathematical Programming techniques, thus considerably extending the context of local search heuristics, and giving rise to a new type of metaheuristics.

• Differently from the others, LB has been the first approach to use a general-purpose solver (specifically a MIP solver) for the neighborhood exploration, instead of relying in the exploitation of its special structure both in the design and solution.
MIP as a paradigm for “new” metaheuristics (cont.d)

• More recent attempts on this line are:

  – Corridor Method [Sniedovich & Voss 2006]:
    An exponential-size neighborhood is defined according to its exploration method (be it a MIP solver, dynamic programming or other), and it is implemented by imposing exogenous constraints on the original problem. This defines a corridor around an incumbent solution along which the solver is forced to move. Stacking of container terminals in a yard.
MIP as a paradigm for “new” metaheuristics (cont.d)

- More recent attempts on this line are:

  - **Corridor Method** [Sniedovich & Voss 2006]:
    An exponential-size neighborhood is defined according to its exploration method (be it a MIP solver, dynamic programming or other), and it is implemented by imposing exogenous constraints on the original problem. This defines a corridor around an incumbent solution along which the solver is forced to move. Stacking of container terminals in a yard.

  - **Decomposition Techniques and Metaheuristics** [Boschetti & Maniezzo 2009]:
    The basic idea is that classical decomposition methods, like Lagrangean, Benders, Dantzig-Wolfe, can be effectively used as frameworks for heuristic approaches in contrast to their original root in exact optimization. Peer-to-Peer Network Design.
**MIP as a paradigm for “new” metaheuristics (cont.d)**

- More recent attempts on this line are:
  
  - **Corridor Method** [Sniedovich & Voss 2006]:
    An exponential-size neighborhood is defined according to its exploration method (be it a MIP solver, dynamic programming or other), and it is implemented by imposing exogenous constraints on the original problem. This defines a corridor around an incumbent solution along which the solver is forced to move. Stacking of container terminals in a yard.

  - **Decomposition Techniques and Metaheuristics** [Boschetti & Maniez 2009]:
    The basic idea is that classical decomposition methods, like Lagrangean, Benders, Dantzig-Wolfe, can be effectively used as frameworks for heuristic approaches in contrast to their original root in exact optimization. Peer-to-Peer Network Design.

  - **Forward & Backward** [Bartolini, Maniez & Mingozzi 2008]:
    The algorithm generates a sequence of gradually improving solutions by exploiting at each iteration the knowledge gained in previous ones. Specifically, the algorithm builds enumerative trees and stores at each tree level a set of promising partial solutions used to drive the tree exploration in the next iteration. Design of telecommunication networks.
Few more (recent) real-world applications

• Integrated resource scheduling in a maritime logistics facility [Chen & Lau 2011]
  Decomposition into a machine scheduling problem and an equipment allocation problem, the
  latter solved by MIP.

• Nesting with defects (leather garment and furniture industry) [Baldacci et al. 2011]
  Lagrangean Decomposition and a Guided Local Search hybrid.

• Helicopter Transport of Oil Rig Crews at Petrobras [Menezes et al. 2010]
  Heuristic column generation approach with a network flow model to optimally assign passengers
  to selected routes.

• Operational management of water-supply networks [Maniezzo et al. 2010]
  Managerial policies of a water network using state-of-the-art algorithms from graph theory,
  generalized flows or flows with gains.

• Traffic Counter Location [Bochetti et al. 2011]
  Lagrangean Metaheuristic approach.