Problem-solving by Mixed-Integer Programming

Andrea Lodi

University of Bologna, Italy
IBM-Unibo Excellence Center in Mathematical Optimization
andrea.lodi@unibo.it

ACP Summer School on **Practical Constraint Programming**, Bologna (Italy)
June 19, 2014

Setting

- We consider a general Mixed Integer Program in the form

$$\max\{c^T x : Ax \leq b, x \geq 0,\ x_j \in \mathbb{Z}, \forall j \in I\} \tag{1}$$

where matrix A does not have a special structure.

Setting

- We consider a general Mixed Integer Program in the form

$$\max\{c^T x : Ax \leq b, x \geq 0,\ x_j \in \mathbb{Z}, \forall j \in I\} \tag{1}$$

  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.

Setting

- We consider a general Mixed Integer Program in the form

$$\max\{c^T x : Ax \leq b, x \geq 0,\ x_j \in \mathbb{Z}, \forall j \in I\} \tag{1}$$

  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 lecture 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.

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.

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. Heuristic nature and capability of MIP solvers.
   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.

MIP Evolution, early days

- Despite quite some work on basically all aspects of IP and in particular on cutting planes, the early days of general-purpose MIP solvers were mainly devoted to develop fast and reliable LP solvers used within good branch-and-bound schemes.

MIP Evolution, early days

- Despite quite some work on basically all aspects of IP and in particular on cutting planes, the early days of general-purpose MIP solvers were mainly devoted to develop fast and reliable LP solvers used within good branch-and-bound schemes.
- Remarkable exceptions are:
  - 1983 Crowder, Johnson & Padberg: PIPX, pure 0/1 MIPs
  - 1987 Van Roy & Wolsey: MPSARX, mixed 0/1 MIPs

MIP Evolution, early days

- Despite quite some work on basically all aspects of IP and in particular on cutting planes, the early days of general-purpose MIP solvers were mainly devoted to develop fast and reliable LP solvers used within good branch-and-bound schemes.

- Remarkable exceptions are:
  - 1983 Crowder, Johnson & Padberg: PIPX, pure 0/1 MIPs

  - 1987 Van Roy & Wolsey: MPSARX, mixed 0/1 MIPs

- When do the early days end?
  Or equivalently, when does the current generation of MIP solvers appear?

MIP Evolution, early days

- Despite quite some work on basically all aspects of IP and in particular on cutting planes, the early days of general-purpose MIP solvers were mainly devoted to develop fast and reliable LP solvers used within good branch-and-bound schemes.
- Remarkable exceptions are:
  - 1983 Crowder, Johnson & Padberg: PIPX, pure 0/1 MIPs
  - 1987 Van Roy & Wolsey: MPSARX, mixed 0/1 MIPs
- When do the early days end?
  Or equivalently, when does the current generation of MIP solvers appear?
- It looks like a major (crucial) step to get to nowadays MIP solvers has been the ultimate proof that cutting plane generation in conjunction with branching could work in general, i.e., after the success in the TSP context:
  - 1994 Balas, Ceria & Cornuéjols: lift-and-project
  - 1996 Balas, Ceria, Cornuéjols & Natraj: Gomory cuts revisited

MIP Evolution, Cplex numbers

- Bob Bixby & Tobias Achterberg performed the following interesting experiment comparing Cplex versions from Cplex 1.2 (the first one with MIP capability) up to Cplex 11.0.

MIP Evolution, Cplex numbers

- Bob Bixby & Tobias Achterberg performed the following interesting experiment comparing Cplex versions from Cplex 1.2 (the first one with MIP capability) up to Cplex 11.0.
- 1,734 MIP instances, time limit of 30,000 CPU seconds, computing times as geometric means normalized wrt Cplex 11.0 (equivalent if within 10%).

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

MIP Evolution, Cplex numbers

- Bob Bixby & Tobias Achterberg performed the following interesting experiment comparing Cplex versions from Cplex 1.2 (the first one with MIP capability) up to Cplex 11.0.
- 1,734 MIP instances, time limit of 30,000 CPU seconds, computing times as geometric means normalized wrt Cplex 11.0 (equivalent if within 10%).

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

- Does anybody know which was the key feature of Cplex v. 6.5?

MIP Evolution, Cutting Planes

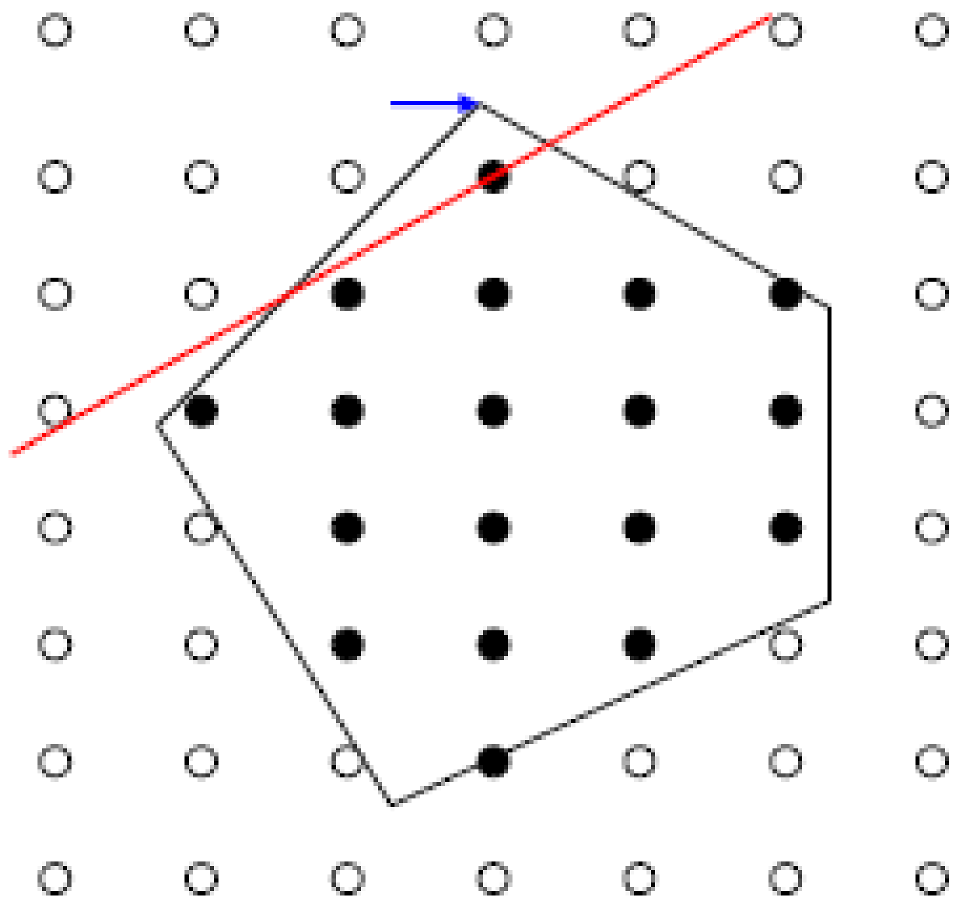


Figure 1: Strengthening the LP relaxation by cutting planes.

MIP Evolution, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):

MIP Evolution, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):
  - Preprocessing:
    probing, bound strengthening, propagation

MIP Evolution, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):
  - Preprocessing:
    probing, bound strengthening, propagation
  - Cutting plane generation:
    Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, . . .

MIP Evolution, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):

  - Preprocessing:
    probing, bound strengthening, propagation

  - Cutting plane generation:
    Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, . . .

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

MIP Evolution, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):

  – Preprocessing:
  probing, bound strengthening, propagation

  – Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, . . .

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

  – Primal heuristics:
  rounding heuristics (from easy to complex), local search, . . .

MIP Evolution, nowadays key features

- The current generation of MIP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):

  – Preprocessing:
  probing, bound strengthening, propagation

  – Cutting plane generation:
  Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, . . .

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

  – Primal heuristics:
  rounding heuristics (from easy to complex), local search, . . .

- Moreover, the MIP computation has reached such an effective and stable quality to allow the solution of sub-MIPs in the algorithmic process, the MIPping approach [Fischetti & Lodi 2002]. These sub-MIPs are solved both for cutting plane generation and in the primal heuristic context.

MIP Building Blocks: Preprocessing/Presolving

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.
- This is generally done without "changing" the set of optimal solutions of the problem at hand, a notable exception being symmetry breaking reductions.

MIP Building Blocks: Preprocessing/Presolving

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.
- This is generally done without "changing" the set of optimal solutions of the problem at hand, a notable exception being symmetry breaking reductions.

- There are two different venues for preprocessing.
  1. Model preprocessing:
     MIP models often contain some "garbage", i.e., redundant or weak information slowing down the solution process by forcing the solver to perform useless operations.
     This is especially true for models originating from real-world applications and created by using modeling languages.

MIP Building Blocks: Preprocessing/Presolving

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.
- This is generally done without "changing" the set of optimal solutions of the problem at hand, a notable exception being symmetry breaking reductions.

- There are two different venues for preprocessing.
  1. Model preprocessing:
     MIP models often contain some "garbage", i.e., redundant or weak information slowing down the solution process by forcing the solver to perform useless operations.
     This is especially true for models originating from real-world applications and created by using modeling languages. There are two types of sources of inefficiency:
     First, the models are unnecessary large and thus harder to manage. This is the case in which there are redundant/parallel constraints or variables that are already fixed and nevertheless appear in the model as additional constraints.

MIP Building Blocks: Preprocessing/Presolving

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.
- This is generally done without "changing" the set of optimal solutions of the problem at hand, a notable exception being symmetry breaking reductions.

- There are two different venues for preprocessing.
  1. Model preprocessing:
     MIP models often contain some "garbage", i.e., redundant or weak information slowing down the solution process by forcing the solver to perform useless operations.
     This is especially true for models originating from real-world applications and created by using modeling languages. There are two types of sources of inefficiency:
     First, the models are unnecessary large and thus harder to manage. This is the case in which there are redundant/parallel constraints or variables that are already fixed and nevertheless appear in the model as additional constraints.
     Second, the variable bounds can be unnecessary large or the constraints could have been written in a loose way, for example with coefficients weaker than they could possibly be.

MIP Building Blocks: Preprocessing/Presolving

- In the preprocessing phase a MIP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.
- This is generally done without "changing" the set of optimal solutions of the problem at hand, a notable exception being symmetry breaking reductions.

- There are two different venues for preprocessing.
  1. Model preprocessing:
     MIP models often contain some "garbage", i.e., redundant or weak information slowing down the solution process by forcing the solver to perform useless operations.
     This is especially true for models originating from real-world applications and created by using modeling languages. There are two types of sources of inefficiency:
     First, the models are unnecessary large and thus harder to manage. This is the case in which there are redundant/parallel constraints or variables that are already fixed and nevertheless appear in the model as additional constraints.
     Second, the variable bounds can be unnecessary large or the constraints could have been written in a loose way, for example with coefficients weaker than they could possibly be.

     Thus, modern MIP solvers have the capability of cleaning up and strengthen a model so as to create a presolved instance on which the MIP technology is then applied.

MIP Building Blocks: Preprocessing/Presolving (cont.d)

2. Algorithmic preprocessing:
   more sophisticated presolve mechanisms are also able to discover important implications and sub-structures that might be of fundamental importance later on in the computation for both branching purposes and cutting plane generation.

MIP Building Blocks: Preprocessing/Presolving (cont.d)

2. Algorithmic preprocessing:
   more sophisticated presolve mechanisms are also able to discover important implications and sub-structures that might be of fundamental importance later on in the computation for both branching purposes and cutting plane generation.

   As an example, the presolve phase determines the clique table or conflict graph, i.e., groups of binary variables such that no more than one can be non-zero at the same time.
   The so-called conflict graph is then fundamental to separate clique inequalities [Johnson and Padberg 1982] written as

$$\sum_{j \in Q} x_j \leq 1 \tag{2}$$

   where Q denotes a subset of (indices of) binary variables such that at most one of them can be non-zero.

MIP Building Blocks: Preprocessing/Presolving (cont.d)

2. Algorithmic preprocessing:
   more sophisticated presolve mechanisms are also able to discover important implications and sub-structures that might be of fundamental importance later on in the computation for both branching purposes and cutting plane generation.

   As an example, the presolve phase determines the clique table or conflict graph, i.e., groups of binary variables such that no more than one can be non-zero at the same time.
   The so-called conflict graph is then fundamental to separate clique inequalities [Johnson and Padberg 1982] written as

$$\sum_{j \in Q} x_j \leq 1 \tag{2}$$

   where Q denotes a subset of (indices of) binary variables such that at most one of them can be non-zero.

   Finally, the lower and upper bounds on the objective function and the solution of LPs can be used to perform even stronger reduction (known as probing) with the aim of fixing variables.

MIP Building Blocks: Preprocessing/Presolving (cont.d)

2. Algorithmic preprocessing:
   more sophisticated presolve mechanisms are also able to discover important implications and sub-structures that might be of fundamental importance later on in the computation for both branching purposes and cutting plane generation.

   As an example, the presolve phase determines the clique table or conflict graph, i.e., groups of binary variables such that no more than one can be non-zero at the same time.
   The so-called conflict graph is then fundamental to separate clique inequalities [Johnson and Padberg 1982] written as

$$\sum_{j \in Q} x_j \leq 1 \tag{2}$$

   where Q denotes a subset of (indices of) binary variables such that at most one of them can be non-zero.

   Finally, the lower and upper bounds on the objective function and the solution of LPs can be used to perform even stronger reduction (known as probing) with the aim of fixing variables.

   This is currently the component of MIP solvers in which the integration with CP techniques has advanced the most.

MIP Building Blocks: Cutting Planes

- From what has been discussed before, it is clear that cutting planes are a crucial components of MIP solvers.

MIP Building Blocks: Cutting Planes

- From what has been discussed before, it is clear that cutting planes are a crucial components of MIP solvers.

- Given the MIP (1), we are mainly interested in the two sets

$$S := \{Ax \leq b, x \geq 0,\ x_j \in \mathbb{Z}, \forall j \in I\} \tag{3}$$

and

$$P := \{Ax \leq b, x \geq 0\}. \tag{4}$$

MIP Building Blocks: Cutting Planes

- From what has been discussed before, it is clear that cutting planes are a crucial components of MIP solvers.

- Given the MIP (1), we are mainly interested in the two sets

$$S := \{Ax \leq b, x \geq 0,\ x_j \in \mathbb{Z}, \forall j \in I\} \tag{3}$$

and

$$P := \{Ax \leq b, x \geq 0\}. \tag{4}$$

- Generality: We are interested in general-purpose cutting planes, those that can be derived without assuming any special structure for the polyhedron P.
- Validity: An inequality $\alpha x \leq \beta$ is said to be valid for S if it is satisfied by all $x \in S$.
- Obtaining a valid inequality for a continuous set: Given P, any valid inequality for it is obtained as $uAx \leq \beta$, where $u \in \mathbb{R}^m_+$ and $\beta \geq ub$. (Farkas Lemma)

MIP Building Blocks: Cutting Planes (cont.d)

- Separation:
  Given a family of valid inequalities $\mathcal{F}$ and a solution $x^* \in P \setminus S$, the **Separation problem for** $\mathcal{F}$ is defined as

  Find an inequality $\alpha x \leq \beta$ of $\mathcal{F}$ valid for S such that $\alpha x^* > \beta$ or show that none exists.

- Iterative strengthening
  1. solve the problem $\{\max c^T x \ : \ x \in P\}$ and get x^*
  2. if $x^* \in S$ then **STOP**
  3. solve the separation problem, add $\alpha x \leq \beta$ to P and go to 1.

- (Almost) all cutting plane classes in the arsenal of nowadays MIP solvers belong to the family of split cuts, i.e., they are separated by exploiting in some way (from easy to complex) a disjunction on the integer variables.

MIP Building Blocks: Cutting Planes (cont.d)

- A basic rounding argument:
  If $x \in \mathbb{Z}$ and $x \leq f$ $f \notin \mathbb{Z}$, then $x \leq \lfloor f \rfloor$.

MIP Building Blocks: Cutting Planes (cont.d)

- A basic rounding argument:
  If $x \in \mathbb{Z}$ and $x \leq f$ $f \notin \mathbb{Z}$, then $x \leq \lfloor f \rfloor$.

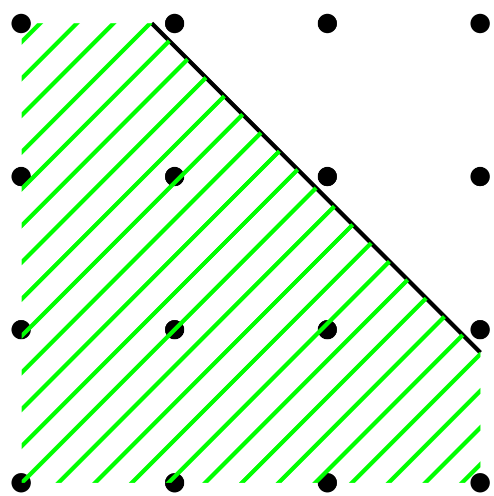
- Using rounding:
  Consider an inequality $\alpha x \leq \beta$ such that $\alpha_j \in \mathbb{Z},\ j = 1, \ldots, n$ in the pure integer case $I = \{1, \ldots, n\}$. If $\alpha x \leq \beta$, then $\alpha x \leq \lfloor \beta \rfloor$ is valid as well.

MIP Building Blocks: Cutting Planes (cont.d)

- A basic rounding argument:
  If $x \in \mathbb{Z}$ and $x \leq f$ $f \notin \mathbb{Z}$, then $x \leq \lfloor f \rfloor$.

- Using rounding:
  Consider an inequality $\alpha x \leq \beta$ such that $\alpha_j \in \mathbb{Z},\ j = 1, \ldots, n$ in the pure integer case $I = \{1, \ldots, n\}$. If $\alpha x \leq \beta$, then $\alpha x \leq \lfloor \beta \rfloor$ is valid as well.

- Example:
  $x \in \mathbb{Z}^2$ such that $x_1 + x_2 \leq 1.9$

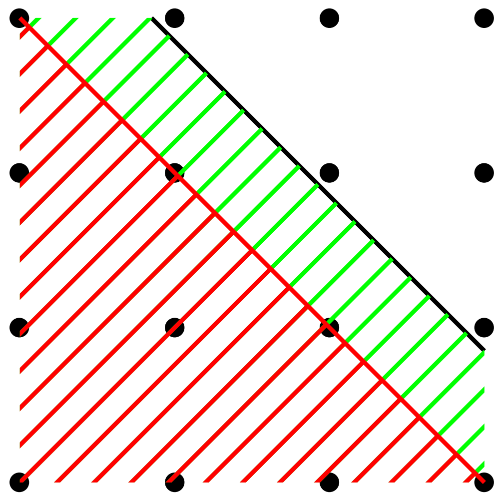


MIP Building Blocks: Cutting Planes (cont.d)

- A basic rounding argument:
  If $x \in \mathbb{Z}$ and $x \leq f$ $f \notin \mathbb{Z}$, then $x \leq \lfloor f \rfloor$.

- Using rounding:
  Consider an inequality $\alpha x \leq \beta$ such that $\alpha_j \in \mathbb{Z},\ j = 1, \ldots, n$ in the pure integer case $I = \{1, \ldots, n\}$. If $\alpha x \leq \beta$, then $\alpha x \leq \lfloor \beta \rfloor$ is valid as well.

- Example:
  $x \in \mathbb{Z}^2$ such that $x_1 + x_2 \leq 1.9 \Rightarrow\ x_1 + x_2 \leq \lfloor 1.9 \rfloor = 1$



MIP Building Blocks: Cutting Planes (cont.d)

- Theorem [Gomory 1958, Chvátal 1973]:
  If $x \in \mathbb{Z}^n$ satisfies $Ax \leq b$, then the inequality $uAx \leq \lfloor ub \rfloor$ is valid for S for all $u \geq 0$ such that $uA \in \mathbb{Z}^n$.

MIP Building Blocks: Cutting Planes (cont.d)

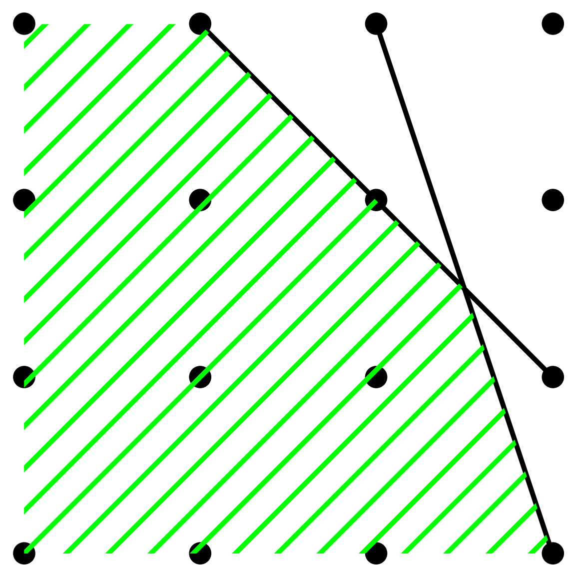
- Theorem [Gomory 1958, Chvátal 1973]:

  If $x \in \mathbb{Z}^n$ satisfies $Ax \leq b$, then the inequality $uAx \leq \lfloor ub \rfloor$ is valid for S for all $u \geq 0$ such that $uA \in \mathbb{Z}^n$.

  Example:

  Consider the polyhedron given by the two inequalities

$$x_1 + x_2 \leq 2$$
$$3x_1 + x_2 \leq 5$$



MIP Building Blocks: Cutting Planes (cont.d)

- Theorem [Gomory 1958, Chvátal 1973]:

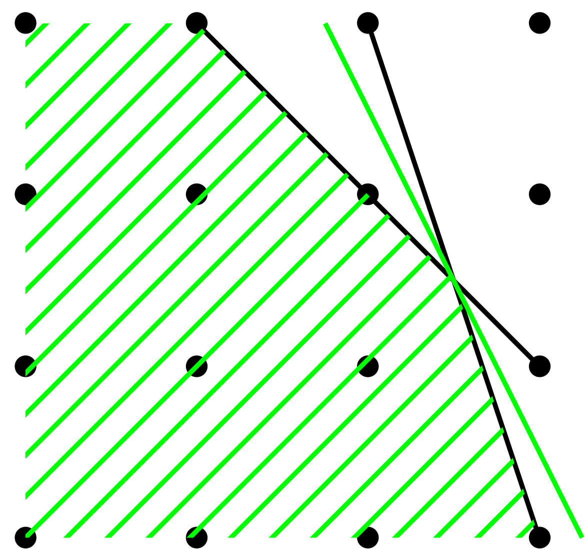
  If $x \in \mathbb{Z}^n$ satisfies $Ax \leq b$, then the inequality $uAx \leq \lfloor ub \rfloor$ is valid for S for all $u \geq 0$ such that $uA \in \mathbb{Z}^m$.

  Example:

  Consider the polyhedron given by the two inequalities

  $$x_1 + x_2 \leq 2$$
  $$3x_1 + x_2 \leq 5$$

  Let $u_1 = u_2 = \frac{1}{2}$, $\Rightarrow 2x_1 + x_2 \leq 3.5$



MIP Building Blocks: Cutting Planes (cont.d)

- Theorem [Gomory 1958, Chvátal 1973]:

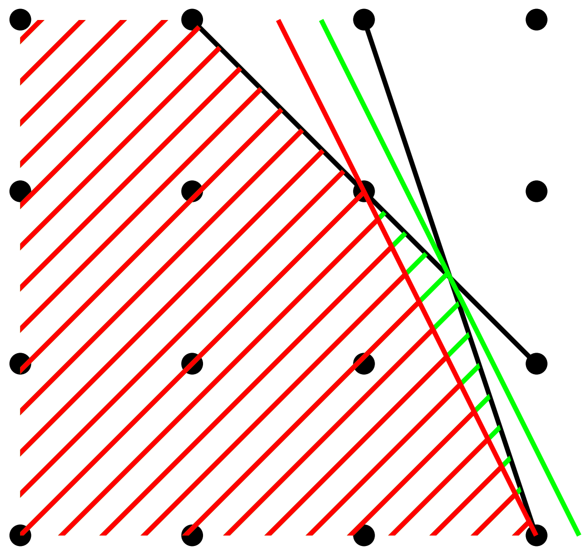
  If $x \in \mathbb{Z}^n$ satisfies $Ax \leq b$, then the inequality $uAx \leq \lfloor ub \rfloor$ is valid for S for all $u \geq 0$ such that $uA \in \mathbb{Z}^m$.

Example:
Consider the polyhedron given by the two inequalities

$$x_1 + x_2 \leq 2$$
$$3x_1 + x_2 \leq 5$$

Let $u_1 = u_2 = \frac{1}{2}$, $\Rightarrow 2x_1 + x_2 \leq 3.5$

and rounding we obtain $2x_1 + x_2 \leq 3$



MIP Building Blocks: Branching

- In its basic version the branch-and-bound algorithm [Land & Doig 1960] iteratively partitions the solution space into sub-MIPs (the children nodes) that have the same theoretical complexity of the originating MIP (the father node, or the root node if it is the initial MIP).

MIP Building Blocks: Branching

- In its basic version the branch-and-bound algorithm [Land & Doig 1960] iteratively partitions the solution space into sub-MIPs (the children nodes) that have the same theoretical complexity of the originating MIP (the father node, or the root node if it is the initial MIP).

- Usually, for MIP solvers the branching creates two children by using the rounding of the solution of the LP relaxation value of a fractional variable, say x_j, constrained to be integral

$$x_j \leq \lfloor x_j^* \rfloor \quad \textbf{OR} \quad x_j \geq \lfloor x_j^* \rfloor + 1. \tag{5}$$

MIP Building Blocks: Branching

- In its basic version the branch-and-bound algorithm [Land & Doig 1960] iteratively partitions the solution space into sub-MIPs (the children nodes) that have the same theoretical complexity of the originating MIP (the father node, or the root node if it is the initial MIP).

- Usually, for MIP solvers the branching creates two children by using the rounding of the solution of the LP relaxation value of a fractional variable, say x_j, constrained to be integral

$$x_j \leq \lfloor x_j^* \rfloor \quad \textbf{OR} \quad x_j \geq \lfloor x_j^* \rfloor + 1. \tag{5}$$

- On the two children, left (or "down") branch and right (or "up") branch, the integrality requirement on the variables $x_j, \forall j \in I$ is relaxed (again) and the LP relaxation is solved.

MIP Building Blocks: Branching

- In its basic version the branch-and-bound algorithm [Land & Doig 1960] iteratively partitions the solution space into sub-MIPs (the children nodes) that have the same theoretical complexity of the originating MIP (the father node, or the root node if it is the initial MIP).

- Usually, for MIP solvers the branching creates two children by using the rounding of the solution of the LP relaxation value of a fractional variable, say x_j, constrained to be integral

$$x_j \leq \lfloor x_j^* \rfloor \quad \textbf{OR} \quad x_j \geq \lfloor x_j^* \rfloor + 1. \tag{5}$$

- On the two children, left (or "down") branch and right (or "up") branch, the integrality requirement on the variables $x_j, \forall j \in I$ is relaxed (again) and the LP relaxation is solved.

- Sub-MIPs become smaller and smaller due to the partition mechanism (basically some of the decisions are taken) and eventually the LP relaxation is directly integral (or infeasible).

MIP Building Blocks: Branching

- In its basic version the branch-and-bound algorithm [Land & Doig 1960] iteratively partitions the solution space into sub-MIPs (the children nodes) that have the same theoretical complexity of the originating MIP (the father node, or the root node if it is the initial MIP).

- Usually, for MIP solvers the branching creates two children by using the rounding of the solution of the LP relaxation value of a fractional variable, say x_j, constrained to be integral

$$x_j \leq \lfloor x_j^* \rfloor \quad \textbf{OR} \quad x_j \geq \lfloor x_j^* \rfloor + 1. \tag{5}$$

- On the two children, left (or "down") branch and right (or "up") branch, the integrality requirement on the variables $x_j, \forall j \in I$ is relaxed (again) and the LP relaxation is solved.

- Sub-MIPs become smaller and smaller due to the partition mechanism (basically some of the decisions are taken) and eventually the LP relaxation is directly integral (or infeasible).

- In addition, the LP relaxation is solved at every node to decide if the node itself is worthwhile to be further partitioned: if the LP relaxation value is already not better (bigger) than the incumbent, the node can be safely fathomed.

MIP Building Blocks: Branching (cont.d)

- Of course, the basic idea of the splitting a node does not require that branching is performed as in (5): i.e., more than two children could be created, and one can branch on more general hyperplanes, or, in general, on any other disjunctive condition.

MIP Building Blocks: Branching (cont.d)

- Of course, the basic idea of the splitting a node does not require that branching is performed as in (5): i.e., more than two children could be created, and one can branch on more general hyperplanes, or, in general, on any other disjunctive condition.

- The reason why variable branching (5) is the most popular (and this situation is not likely to change anytime soon, at least for MIP solvers) is that it takes full advantage of the ability of the Simplex algorithm to recompute the optimal solution of the LP relaxation if only variable bounds (possibly one) have changed.

MIP Building Blocks: Branching (cont.d)

- Of course, the basic idea of the splitting a node does not require that branching is performed as in (5): i.e., more than two children could be created, and one can branch on more general hyperplanes, or, in general, on any other disjunctive condition.

- The reason why variable branching (5) is the most popular (and this situation is not likely to change anytime soon, at least for MIP solvers) is that it takes full advantage of the ability of the Simplex algorithm to recompute the optimal solution of the LP relaxation if only variable bounds (possibly one) have changed.

- In fact, on average, for a single LP solution Interior Point algorithms perform better than the Simplex algorithm [Rothberg 2010], which is in turn (currently) unbeatable in the iterative context.

MIP Building Blocks: Branching (cont.d)

- Of course, the basic idea of the splitting a node does not require that branching is performed as in (5): i.e., more than two children could be created, and one can branch on more general hyperplanes, or, in general, on any other disjunctive condition.

- The reason why variable branching (5) is the most popular (and this situation is not likely to change anytime soon, at least for MIP solvers) is that it takes full advantage of the ability of the Simplex algorithm to recompute the optimal solution of the LP relaxation if only variable bounds (possibly one) have changed.

- In fact, on average, for a single LP solution Interior Point algorithms perform better than the Simplex algorithm [Rothberg 2010], which is in turn (currently) unbeatable in the iterative context.

- The described branch-and-bound framework requires two independent and important decisions at any step: Node and Variable selection.

MIP Building Blocks: Branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.

MIP Building Blocks: Branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.
   All other techniques, more or less sophisticated, are basically hybrids around these two ideas.

MIP Building Blocks: Branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.
   All other techniques, more or less sophisticated, are basically hybrids around these two ideas.

2. Variable selection:
   The variable selection problem is the one of deciding how to partition the current node, i.e., on which variable to branch on in order to create the two children.

MIP Building Blocks: Branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.
   All other techniques, more or less sophisticated, are basically hybrids around these two ideas.

2. Variable selection:
   The variable selection problem is the one of deciding how to partition the current node, i.e., on which variable to branch on in order to create the two children.

   For a long time, a classical choice has been branching on the most fractional variable, i.e., in the 0-1 case the closest to 0.5.

MIP Building Blocks: Branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.
   All other techniques, more or less sophisticated, are basically hybrids around these two ideas.

2. Variable selection:
   The variable selection problem is the one of deciding how to partition the current node, i.e., on which variable to branch on in order to create the two children.

   For a long time, a classical choice has been branching on the most fractional variable, i.e., in the 0-1 case the closest to 0.5.

   That rule has been computationally shown to be worse than a complete random choice [Achterberg et al. 2005]. However, it is of course very easy to evaluate.

MIP Building Blocks: Branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.
   All other techniques, more or less sophisticated, are basically hybrids around these two ideas.

2. Variable selection:
   The variable selection problem is the one of deciding how to partition the current node, i.e., on which variable to branch on in order to create the two children.

   For a long time, a classical choice has been branching on the most fractional variable, i.e., in the 0-1 case the closest to 0.5.

   That rule has been computationally shown to be worse than a complete random choice [Achterberg et al. 2005]. However, it is of course very easy to evaluate.

   In order to devise stronger criteria one has to do much more work.

MIP Building Blocks: Branching (cont.d)

2. Variable selection (cont.d):
   The extreme is the so called strong branching technique [Applegate et al. 2007; Linderoth & Savelsbergh 1999].

MIP Building Blocks: Branching (cont.d)

2. Variable selection (cont.d):
   The extreme is the so called strong branching technique [Applegate et al. 2007; Linderoth & Savelsbergh 1999].

   In its full version, at any node one has to simulate branch on each candidate fractional variable and select the one on which the improvement (decrease) in the bound on the left branch times the one on the right branch is the maximum.

   Of course, this is in general computationally unpractical (discussed later) but all MIP solvers implement lighter versions of this scheme.

MIP Building Blocks: Branching (cont.d)

2. Variable selection (cont.d):
   The extreme is the so called strong branching technique [Applegate et al. 2007; Linderoth & Savelsbergh 1999].

   In its full version, at any node one has to simulate branch on each candidate fractional variable and select the one on which the improvement (decrease) in the bound on the left branch times the one on the right branch is the maximum.
   Of course, this is in general computationally unpractical (discussed later) but all MIP solvers implement lighter versions of this scheme.

   Another sophisticated technique is pseudocost branching [Benichouet al. 1971] that keeps a history of the success (in terms of the change in the LP relaxation value) of the branchings already performed on each variable as an indication of the quality of the variable itself.

MIP Building Blocks: Branching (cont.d)

2. Variable selection (cont.d):
   The extreme is the so called strong branching technique [Applegate et al. 2007; Linderoth & Savelsbergh 1999].

   In its full version, at any node one has to simulate branch on each candidate fractional variable and select the one on which the improvement (decrease) in the bound on the left branch times the one on the right branch is the maximum.
   Of course, this is in general computationally unpractical (discussed later) but all MIP solvers implement lighter versions of this scheme.

   Another sophisticated technique is pseudocost branching [Benichouet al. 1971] that keeps a history of the success (in terms of the change in the LP relaxation value) of the branchings already performed on each variable as an indication of the quality of the variable itself.

   The most recent effective and sophisticated method, called reliability branching [Achterberg et al. 2005], integrates strong and pseudocost branchings by defining a reliability threshold, i.e., a level below which the information of the pseudocosts is not considered accurate enough and some strong branching is performed.

MIP Building Blocks: Primal Heuristics

- The last 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solutions early in the tree.

MIP Building Blocks: Primal Heuristics

- The last 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solutions early in the tree.

- However, a very meaningful experiment [Danna 2007] has shown that even the knowledge of the optimal solution from the beginning of the search improves on average the running time of a MIP solver **only** by a factor of 2.

MIP Building Blocks: Primal Heuristics

- The last 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solutions early in the tree.

- However, a very meaningful experiment [Danna 2007] has shown that even the knowledge of the optimal solution from the beginning of the search improves on average the running time of a MIP solver **only** by a factor of 2.

- In other words, heuristics largely impact on the user perception of the quality of a solver, and are fundamental in the real-world context.

MIP Building Blocks: Primal Heuristics

- The last 10 years have seen a tremendous improvement in the capability of primal heuristics to find very good (almost optimal) solutions early in the tree.

- However, a very meaningful experiment [Danna 2007] has shown that even the knowledge of the optimal solution from the beginning of the search improves on average the running time of a MIP solver **only** by a factor of 2.

- In other words, heuristics largely impact on the user perception of the quality of a solver, and are fundamental in the real-world context.

- The primal heuristics implemented in the solvers go from very light and easy, as variations of the classical rounding of the LP solution, to much more heavy and complex, like local search and metaheuristics.

- Details on these latter classes of heuristics will be discussed in the second part of the lecture.

MIP Solvers: exploiting multiple cores

- The branch-and-bound algorithm is a natural one to parallelize, as nodes of the search tree may be processed independently.

MIP Solvers: exploiting multiple cores

- The branch-and-bound algorithm is a natural one to parallelize, as nodes of the search tree may be processed independently.
- The two types of parallel MIP research can be loosely categorized based on the type of parallel computing architecture used:
  1. Distributed-memory architectures rely on message passing to communicate results of the algorithm.
  2. Shared-memory computers communicate information among CPU's by reading from and writing to a common memory pool.

MIP Solvers: exploiting multiple cores

- The branch-and-bound algorithm is a natural one to parallelize, as nodes of the search tree may be processed independently.
- The two types of parallel MIP research can be loosely categorized based on the type of parallel computing architecture used:
  1. Distributed-memory architectures rely on message passing to communicate results of the algorithm.
  2. Shared-memory computers communicate information among CPU's by reading from and writing to a common memory pool.
- In parallel branch-and-bound, the order in which node computations are completed can have a significant impact on performance, and often lead to anomalous behavior: one can run the same instance, with the same parameter settings, and achieve very different results in terms of nodes evaluated and CPU time.

MIP Solvers: exploiting multiple cores

- The branch-and-bound algorithm is a natural one to parallelize, as nodes of the search tree may be processed independently.
- The two types of parallel MIP research can be loosely categorized based on the type of parallel computing architecture used:
  1. Distributed-memory architectures rely on message passing to communicate results of the algorithm.
  2. Shared-memory computers communicate information among CPU's by reading from and writing to a common memory pool.
- In parallel branch-and-bound, the order in which node computations are completed can have a significant impact on performance, and often lead to anomalous behavior: one can run the same instance, with the same parameter settings, and achieve very different results in terms of nodes evaluated and CPU time.
- To combat this undesirable behavior, modern (shared-memory-based) MIP software has introduced appropriate synchronization points in the algorithm to ensure reproducible behavior in a parallel environment. Some overhead is introduced by these synchronization mechanisms.

MIP Solvers: exploiting multiple cores

- The branch-and-bound algorithm is a natural one to parallelize, as nodes of the search tree may be processed independently.
- The two types of parallel MIP research can be loosely categorized based on the type of parallel computing architecture used:
  1. Distributed-memory architectures rely on message passing to communicate results of the algorithm.
  2. Shared-memory computers communicate information among CPU's by reading from and writing to a common memory pool.
- In parallel branch-and-bound, the order in which node computations are completed can have a significant impact on performance, and often lead to anomalous behavior: one can run the same instance, with the same parameter settings, and achieve very different results in terms of nodes evaluated and CPU time.
- To combat this undesirable behavior, modern (shared-memory-based) MIP software has introduced appropriate synchronization points in the algorithm to ensure reproducible behavior in a parallel environment. Some overhead is introduced by these synchronization mechanisms.

- However, the most intriguing development associated with the availability of multiple cores is the fact of exploiting them for doing different "things", not different nodes. In other words, to run different algorithmic strategies on different cores and/or use them to learn what is the best.

MIP Evolution, a development viewpoint

- Solving a MIP to optimality is only one aspect of using a MIP solver for applications, sometimes not the most important one (discussed later).

MIP Evolution, a development viewpoint

- Solving a MIP to optimality is only one aspect of using a MIP solver for applications, sometimes not the most important one (discussed later).
  Nowadays MIP solvers include useful tools for complex algorithmic design and data and model analysis. Some of them are:

MIP Evolution, a development viewpoint

- Solving a MIP to optimality is only one aspect of using a MIP solver for applications, sometimes not the most important one (discussed later).
  Nowadays MIP solvers include useful tools for complex algorithmic design and data and model analysis. Some of them are:
  - automatic tuning of the parameters:
    the number of parameters (corresponding to different algorithmic options) makes the hand-tuning complex but it guarantees great flexibility

MIP Evolution, a development viewpoint

- Solving a MIP to optimality is only one aspect of using a MIP solver for applications, sometimes not the most important one (discussed later).
  Nowadays MIP solvers include useful tools for complex algorithmic design and data and model analysis. Some of them are:

  – automatic tuning of the parameters:
    the number of parameters (corresponding to different algorithmic options) makes the hand-tuning complex but it guarantees great flexibility

  – multiple solutions:
    allow flexibility and support for decision making and, as side effect, improve primal heuristics

MIP Evolution, a development viewpoint

- Solving a MIP to optimality is only one aspect of using a MIP solver for applications, sometimes not the most important one (discussed later).
  Nowadays MIP solvers include useful tools for complex algorithmic design and data and model analysis. Some of them are:

  - automatic tuning of the parameters:
    the number of parameters (corresponding to different algorithmic options) makes the hand-tuning complex but it guarantees great flexibility

  - multiple solutions:
    allow flexibility and support for decision making and, as side effect, improve primal heuristics

  - detection of sources of infeasibility in the models:
    real-world models are often over constrained and sources of infeasibility must be removed [Amaldi et al. 2003; Chinneck 2001]

MIP Evolution, a development viewpoint

- Solving a MIP to optimality is only one aspect of using a MIP solver for applications, sometimes not the most important one (discussed later).
  Nowadays MIP solvers include useful tools for complex algorithmic design and data and model analysis. Some of them are:

  - automatic tuning of the parameters:
    the number of parameters (corresponding to different algorithmic options) makes the hand-tuning complex but it guarantees great flexibility

  - multiple solutions:
    allow flexibility and support for decision making and, as side effect, improve primal heuristics

  - detection of sources of infeasibility in the models:
    real-world models are often over constrained and sources of infeasibility must be removed [Amaldi et al. 2003; Chinneck 2001]

  - callbacks:
    allow flexibility to accommodate the user code so as to take advantage of specific knowledge

MIP Software

- We have already discussed about the historical path that led to nowadays solvers.

MIP Software

- We have already discussed about the historical path that led to nowadays solvers.

- An important aspect of the design of software for solving MIPs is the user interface. The range of purposes for MIP software is quite large, thus the need of a large number of user interfaces.

MIP Software

- We have already discussed about the historical path that led to nowadays solvers.

- An important aspect of the design of software for solving MIPs is the user interface. The range of purposes for MIP software is quite large, thus the need of a large number of user interfaces.

- In general, users may wish
  1. to solve MIPs using the solver as a "black box" (so-called interactive use),
  2. to call the software from a third-party package (like a modeling language as AMPL), or
  3. to embed the solver into custom applications, which would require software to have a callable library.

MIP Software

- We have already discussed about the historical path that led to nowadays solvers.
- An important aspect of the design of software for solving MIPs is the user interface. The range of purposes for MIP software is quite large, thus the need of a large number of user interfaces.
- In general, users may wish
  1. to solve MIPs using the solver as a "black box" (so-called interactive use),
  2. to call the software from a third-party package (like a modeling language as AMPL), or
  3. to embed the solver into custom applications, which would require software to have a callable library.
- Finally, the user may wish to adapt certain aspects of the algorithm, and, as already discussed, this can be achieved by callback functions, or, when the source code is available, through abstract interfaces.

MIP Commercial Software

1. Cplex

Version	12.6
Website	`http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`
Interfaces	C, C++, Java, `.NET`, Matlab, Python, Microsoft Excel

- Cplex is owned and distributed by IBM.
- A special search algorithm, called *dynamic search* can be used instead of branch-and-cut.
- Cplex is moving to Mixed Integer Non-Linear Programming MINLP, being already able to solve a large portion of quadratic and quadratically-constrained Mixed Integer Programs.

2. Gurobi

Version	5.6
Website	`www.gurobi.com`
Interfaces	C, C++, Java, Python, `.NET`, Matlab

- Gurobi Optimizer contains a relatively new MIP solver that was built from scratch to exploit modern multi-core processing technology.
- Gurobi is also available “on demand” using the Amazon Elastic Compute Cloud.

MIP Commercial Software (cont.d)

3. LINDO

Version	8.0
Website	`www.lindo.com`
Interfaces	C, Visual Basic, Matlab, Ox

- LINDO Systems offers a MIP solver as part of its LINDO API.

4. Mosek

Version	7.0
Website	`www.mosek.com`
Interfaces	C, C++, Java, .NET, Python

- MOSEK ApS is a company specializing in generic mathematical optimization software.
- Mosek suite is especially powerful for MINLP and is available through GAMS.

5. XPRESS-MP

Version	7.7
Website	`http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx`
Interfaces	C, C++, Java, .NET, VBA

- A unique feature of XPRESS-MP is that it offers an option to branch into general (split) disjunctions, or to search for special structures on which to branch.

MIP Noncommercial Software

1. BLIS

License	Common Public License
Version	0.93
Website	`https://projects.coin-or.org/CHiPPS`
Language	C++

- Open-source MIP solver available as part of the COIN-OR project.
- Built on top of the COIN-OR High-Performance Parallel Search Framework (CHiPPS), it runs on a distributed memory platforms.
- LPs are solved using the COIN-OR linear programming Solver (Clp).

2. CBC

License	Common Public License
Version	2.8.7
Website	`https://projects.coin-or.org/Cbc`
Language	C++

- Open-source MIP solver distributed under the COIN-OR project and built from many COIN components, including the COIN-OR Clp.

MIP Noncommercial Software (cont.d)

3. GLPK

License	GNU General Public License (GPL)
Version	4.52
Website	`http://www.gnu.org/software/glpk/`
Language	C

- The software distinguishes itself through the large number of community-built interfaces available.

4. lp_solve

License	GNU lesser general public license (LGPL)
Version	5.5.2
Website	`http://lpsolve.sourceforge.net/5.5/`
Language	C

- Open source linear and integer programming solver.

MIP Noncommercial Software (cont.d)

5. MINTO

License	Given as library only
Version	3.1
Website	http://coral.ie.lehigh.edu/minto/
Language	C

- Black-box solver and solver framework for MIP.
- Primary development of the software was done in the 1990's: a whole generation of MIP researchers has been trained with MINTO!

6. SCIP

License	ZIB Academic License
Version	3.0.1
Website	http://scip.zib.de/
Language	C

- Developed and distributed by a team of researchers at Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB).
- SCIP is also a framework for Constraint Integer Programming and branch-cut-and-price, allowing the user significant control of the algorithm.
- Current benchmarks indicate that SCIP is likely the fastest noncommercial MIP solver.

MIP Noncommercial Software (cont.d)

7. SYMPHONY

License	Common Public License
Version	5.2
Website	`http://www.coin-or.org/SYMPHONY/index.htm`
Language	C

- The core solution methodology of SYMPHONY is a customizable branch, cut, and price algorithm that can be executed sequentially or in parallel.
- SYMPHONY has several unique features including the capability to warm start the branch-and-bound process from a previously calculated branch-and-bound tree, even after modifying the problem data.

MIP Software: further remarks/pointers

- Any review of software features is inherently limited by the temporal nature of software itself.

MIP Software: further remarks/pointers

- Any review of software features is inherently limited by the temporal nature of software itself.

- Many of the commercial packages have free or limited cost licensing options for academics.

MIP Software: further remarks/pointers

- Any review of software features is inherently limited by the temporal nature of software itself.
- Many of the commercial packages have free or limited cost licensing options for academics.
- Hans Mittelmann has for many years run independent benchmarks of MIP software, and been publishing the results. In general, the commercial software significantly outperforms noncommercial software, but no conclusion is possible on the relative performance of different commercial systems. See `http://plato.asu.edu/bench.html`.

MIP Software: further remarks/pointers

- Any review of software features is inherently limited by the temporal nature of software itself.
- Many of the commercial packages have free or limited cost licensing options for academics.
- Hans Mittelmann has for many years run independent benchmarks of MIP software, and been publishing the results. In general, the commercial software significantly outperforms noncommercial software, but no conclusion is possible on the relative performance of different commercial systems. See `http://plato.asu.edu/bench.html`.
- The most recent version of MIP library, MIPLIB 2010 `http://miplib.zib.de/`, not only provides problems and data, but it includes, for the first time, scripts to run automated tests in a predefined way, and a solution checker to test the accuracy of provided solutions using exact arithmetic [Koch et al. 2011].

MIP Software: further remarks/pointers

- Any review of software features is inherently limited by the temporal nature of software itself.
- Many of the commercial packages have free or limited cost licensing options for academics.
- Hans Mittelmann has for many years run independent benchmarks of MIP software, and been publishing the results. In general, the commercial software significantly outperforms noncommercial software, but no conclusion is possible on the relative performance of different commercial systems. See `http://plato.asu.edu/bench.html`.
- The most recent version of MIP library, MIPLIB 2010 `http://miplib.zib.de/`, not only provides problems and data, but it includes, for the first time, scripts to run automated tests in a predefined way, and a solution checker to test the accuracy of provided solutions using exact arithmetic [Koch et al. 2011].
- NEOS, server for optimization `www-neos.mcs.anl.gov/neos`: A user can submit an optimization problem to the server and obtain the solution and running time statistics using the preferred solver through different interfaces.

MIP Software: further remarks/pointers

- Any review of software features is inherently limited by the temporal nature of software itself.

- Many of the commercial packages have free or limited cost licensing options for academics.

- Hans Mittelmann has for many years run independent benchmarks of MIP software, and been publishing the results. In general, the commercial software significantly outperforms noncommercial software, but no conclusion is possible on the relative performance of different commercial systems. See `http://plato.asu.edu/bench.html`.

- The most recent version of MIP library, MIPLIB 2010 `http://miplib.zib.de/`, not only provides problems and data, but it includes, for the first time, scripts to run automated tests in a predefined way, and a solution checker to test the accuracy of provided solutions using exact arithmetic [Koch et al. 2011].

- NEOS, server for optimization `www-neos.mcs.anl.gov/neos`: A user can submit an optimization problem to the server and obtain the solution and running time statistics using the preferred solver through different interfaces.

- COIN-OR `http://www.coin-or.org`: COmputational INfrastructure for Operations Research.

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 $\mathcal{N}P$-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 $\mathcal{N}P$-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
   - . . .

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.

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.

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".

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".
   It is less strict for optimality. A popular default value for that is 0.01% that, 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 $\mathcal{N}P$-hardness.

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{N}P$-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 $\mathcal{N}P$-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

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{N}P$-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

Slightly-less Trivial Facts: 3.

3. MIP $\mathcal{N}P$-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

   – *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 $\mathcal{N}P$-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

   – *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".

   – ...

Performance Variability, Emilie Danna #1

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.

        Nodes                                  Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0      917.0000   140                   917.0000     1100
      0     0      924.0000   165                   Cuts: 50     1969
      0     0      924.0000   167                   Cuts: 17     2348
      0     0      924.0000   175                Cliques: 14     2731
*     0+    0                        924.0000       924.0000     2731    0.00%

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

Solution pool: 1 solution saved.

MIP - Integer optimal solution:  Objective =  9.2400000000e+02
Solution time =    0.41 sec.  Iterations = 2731  Nodes = 0
```

2

Example: 10 teams, CPLEX 11, AIX

ILOG Changing the rules of business

```
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/
   Node  Left     Objective  IInf  Best Integer    Best Node    ItCnt     Gap

      0     0     917.0000   151                    917.0000     1053
      0     0     924.0000   152                    Cuts: 53     1801
      0     0     924.0000   161                 Cliques: 14     2336
      0     0     924.0000   163                 Cliques: 12     2609
      0     2     924.0000   163                    924.0000     2609
*   100+   96                       952.0000      924.0000    12316    2.94%
   1000   520     926.7273    85      952.0000      924.0000    97832    2.94%
*  1425     0     integral     0      924.0000      924.0000   122948    0.00%

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
```

3

Slightly-less Trivial Facts: 4.

4. Benchmarking of (commercial) MIP solvers.

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.

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

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.

Benchmarking: Performance Variability

- The performance variability phenomenon has certainly been observed for decades, especially by the Artificial Intelligence and SATisfiability communities.
- However, in the MIP context it has only been reported explicitly rather recently [Danna 2008].

Benchmarking: Performance Variability

- The performance variability phenomenon has certainly been observed for decades, especially by the Artificial Intelligence and SATisfiability communities.
- However, in the MIP context it has only been reported explicitly rather recently [Danna 2008].

- Since then, some work has been devoted to gain a deeper understanding of performance variability and to point out its implications in the benchmarking of MIP solvers.
- Indeed, the most dangerous effect of performance variability is the misinterpretation of the computational results obtained by testing a scientific idea or even a change in the code that might appear harmless.

Benchmarking: Performance Variability

- The performance variability phenomenon has certainly been observed for decades, especially by the Artificial Intelligence and SATisfiability communities.
- However, in the MIP context it has only been reported explicitly rather recently [Danna 2008].

- Since then, some work has been devoted to gain a deeper understanding of performance variability and to point out its implications in the benchmarking of MIP solvers.
- Indeed, the most dangerous effect of performance variability is the misinterpretation of the computational results obtained by testing a scientific idea or even a change in the code that might appear harmless.

- Precisely in the attempt of showing potential mistakes associated with performance variability, yet another very instructive example is discussed in the provocative talk by Fischetti [Fischetti and Monaci 2012].

Performance Variability, Fischetti & Monaci #1

- The computational investigation calls for amending MIP (1) by one single (mysterious) cut obtained by a parametrized lifting procedure that has 9 different variants.
- In other words, 9 copies of any MIP instance are produced, each one differing from the original MIP by 1 valid inequality only.

Performance Variability, Fischetti & Monaci #1

- The computational investigation calls for amending MIP (1) by one single (mysterious) cut obtained by a parametrized lifting procedure that has 9 different variants.
- In other words, 9 copies of any MIP instance are produced, each one differing from the original MIP by 1 valid inequality only.

- The testbed is composed by 38 hard instances selected by

1. taking all the instances in the MIPLIB 2003 and COR@L libraries,

Performance Variability, Fischetti & Monaci #1

- The computational investigation calls for amending MIP (1) by one single (mysterious) cut obtained by a parametrized lifting procedure that has 9 different variants.
- In other words, 9 copies of any MIP instance are produced, each one differing from the original MIP by 1 valid inequality only.
- The testbed is composed by 38 hard instances selected by

1. taking all the instances in the MIPLIB 2003 and COR@L libraries,
2. solving them through IBM ILOG Cplex 12.2 (default setting, no upper cutoff, single-thread mode) on an Intel i5-750 CPU running at 2.67GHz, and

Performance Variability, Fischetti & Monaci #1

- The computational investigation calls for amending MIP (1) by one single (mysterious) cut obtained by a parametrized lifting procedure that has 9 different variants.
- In other words, 9 copies of any MIP instance are produced, each one differing from the original MIP by 1 valid inequality only.

- The testbed is composed by 38 hard instances selected by

  1. taking all the instances in the MIPLIB 2003 and COR@L libraries,
  2. solving them through IBM ILOG Cplex 12.2 (default setting, no upper cutoff, single-thread mode) on an Intel i5-750 CPU running at 2.67GHz, and
  3. picking those requiring more than 10,000 nodes and 100 CPU seconds on the original version of the MIP.

Performance Variability, Fischetti & Monaci #2

	Avg. sec.s	Avg. nodes	Time ratio	Node ratio
Default (no cut)	**533,00**	**64499,09**	**1,00**	**1,00**
Method #1	**397,50**	**37194,89**	**0,75**	**0,58**
Method #2	419,22	44399,47	0,79	0,69
Method #3	468,87	48971,72	0,88	0,76
Method #4	491,77	46348,39	0,92	0,72
Method #5	582,42	58223,10	**1,09**	0,90
Method #6	425,38	43492,35	0,80	0,67
Method #7	457,95	46067,74	0,86	0,71
Method #8	446,89	44481,75	0,84	0,69
Method #9	419,57	41549,07	0,79	0,64

Cases with large speedup

	NO CUT		METHOD #1		
	Time	Nodes	Time	Nodes	Time Speedup
glass4	43,08	118.151	12,95	17.725	3,33
neos-1451294	3.590,27	20.258	102,94	521	34,88
neos-1593097	149,94	10.879	16,12	508	9,30
neos-1595230	1.855,69	152.951	770,6	89.671	2,41
neos-603073	452,4	36.530	130,75	10.017	3,46
neos-911970	3.588,54	5.099.389	3,29	1.767	1.090,74
ran14x18_1	3.287,59	1.480.624	2.066,70	759.265	1,59

Performance Variability, Fischetti & Monaci #4

- When the cutting plane separation procedure is disclosed, we discover that it produces the trivially-valid and redundant linear inequality

$$\sum_{j=1}^{n} x_{\pi^k(j)} \geq -1, \tag{6}$$

where π^k is a random permutation of the nonnegative variables $x_1, \ldots, x_n$.

Performance Variability, Fischetti & Monaci #4

- When the cutting plane separation procedure is disclosed, we discover that it produces the trivially-valid and redundant linear inequality

$$\sum_{j=1}^{n} x_{\pi^k(j)} \geq -1, \tag{6}$$

where π^k is a random permutation of the nonnegative variables $x_1, \ldots, x_n$.

- Through the simple trick of loading the new constraint first, each cut simply changes the order in which the variables are stored by the MIP solver, thus determining a permutation of its columns.

Performance Variability, Fischetti & Monaci #4

- When the cutting plane separation procedure is disclosed, we discover that it produces the trivially-valid and redundant linear inequality

$$\sum_{j=1}^{n} x_{\pi^k(j)} \geq -1, \tag{6}$$

where π^k is a random permutation of the nonnegative variables $x_1, \ldots, x_n$.

- Through the simple trick of loading the new constraint first, each cut simply changes the order in which the variables are stored by the MIP solver, thus determining a permutation of its columns.

- However, there is more. The testbed selection is heavily biased to favor problems in which the reference solver does not perform well.

Performance Variability, Fischetti & Monaci #4

- When the cutting plane separation procedure is disclosed, we discover that it produces the trivially-valid and redundant linear inequality

$$\sum_{j=1}^{n} x_{\pi^k(j)} \geq -1, \tag{6}$$

where π^k is a random permutation of the nonnegative variables $x_1, \ldots, x_n$.

- Through the simple trick of loading the new constraint first, each cut simply changes the order in which the variables are stored by the MIP solver, thus determining a permutation of its columns.

- However, there is more. The testbed selection is heavily biased to favor problems in which the reference solver does not perform well.
- Thus, virtually any change in the solution process (due to the column permutation) is a winner.

Overall: does Cplex v.x dominate Cplex v.y ($x > y$, even $x >> y$)?

- Let us go back and look at the Cplex numbers.

Overall: does Cplex v.x dominate Cplex v.y ($x > y$, even $x >> y$)?

- Let us go back and look at the Cplex numbers.

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

Overall: does Cplex v.x dominate Cplex v.y ($x > y$, even $x >> y$)?

- Let us go back and look at the Cplex numbers.

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

- There are 17 instances on which Cplex 1.2 (1991) is at least 10% faster than Cplex 11.0 (2007)!

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

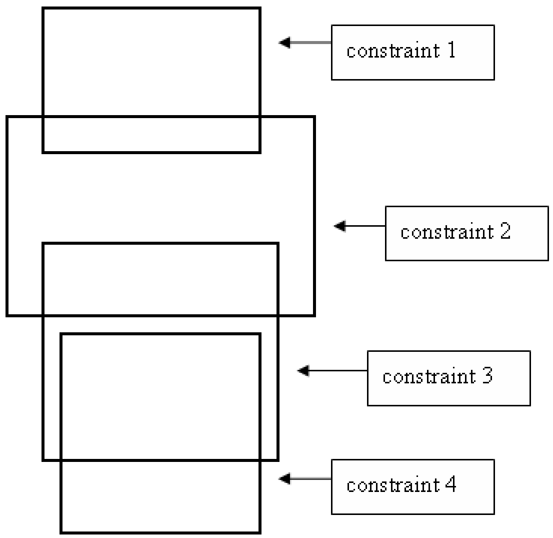
  D Primal heuristics:
  rounding heuristics (from easy to complex), local search, metaheuristics, . . .

5.A: Preprocessing

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

5.A: Preprocessing

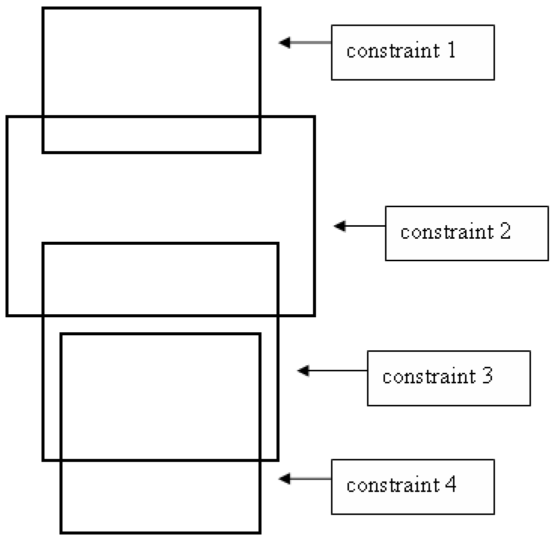
- Many of the techniques used in preprocessing are called in the *Constraint Programming* context propagation algorithms:



  – 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:

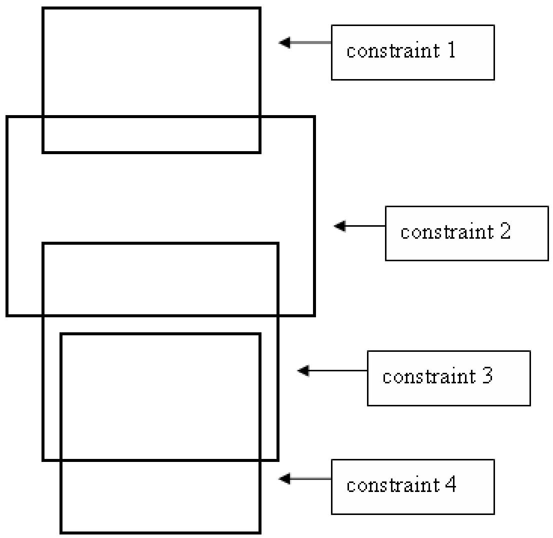


  – A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.

  – 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:

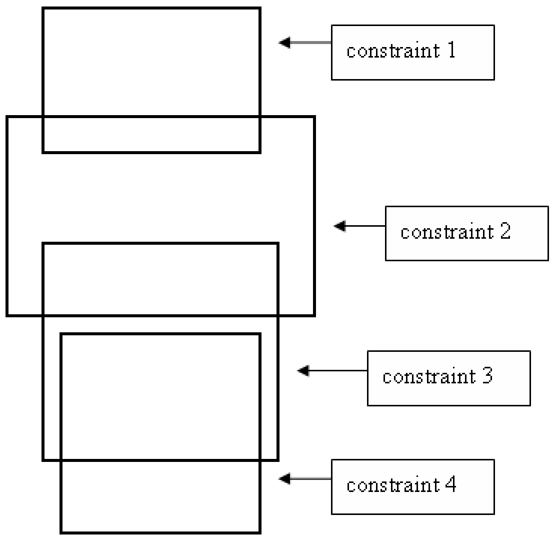


  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
  - Moreover, a global constraint contains an algorithm that prunes (filters) values from the variable domains so as to reduce as much as possible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

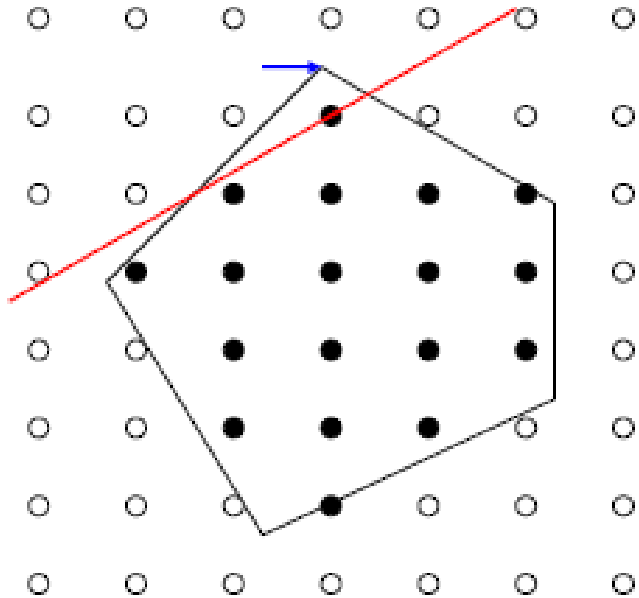


  - A global constraint defines combinatorially a portion of the feasible region, i.e., it is able to check feasibility of an assignment of values to variables.
  - Moreover, a global constraint contains an algorithm that prunes (filters) values from the variable domains so as to reduce as much as possible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

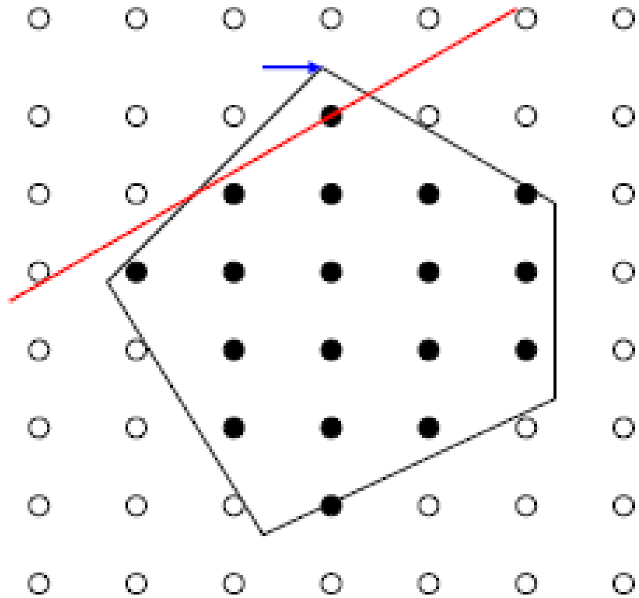
- MIPs do not explicitly contain global constraints, thus, the propagation is applied by LOCALLY comparing constraints/variables, a structurally heuristic process.

- 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

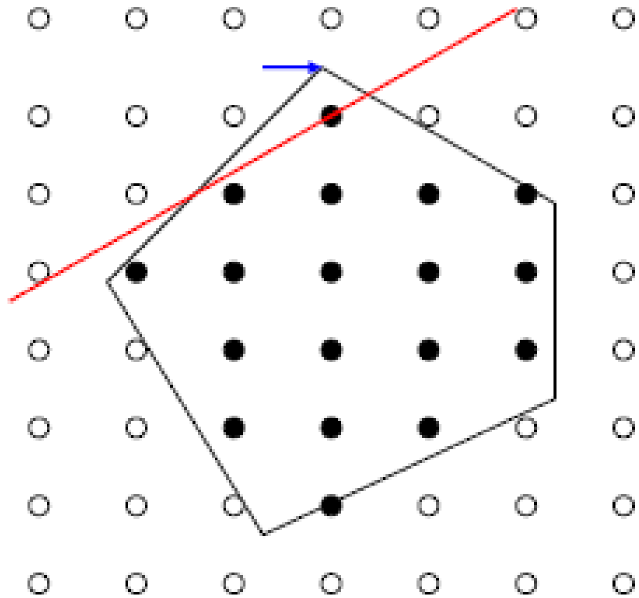


5.B: Cutting Planes



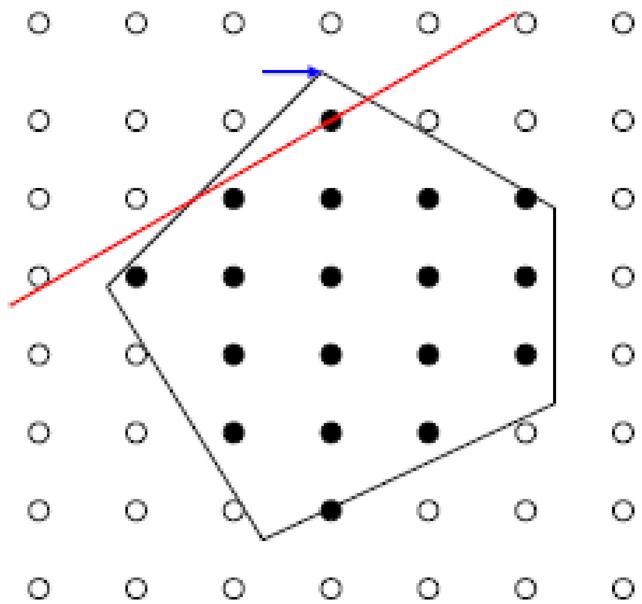
- Current MIP cutting plane technology is based on the following almost trivial two-step heuristic:

5.B: Cutting Planes



- 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, $\{0, \frac{1}{2}\}$, . . .)

5.B: Cutting Planes



- 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, $\{0, \frac{1}{2}\}$, . . .)
- 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.

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 in the size 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 in the size 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 in the size 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).

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 in the size 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 in the size 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.

5.D: Primal Heuristics

- 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!

5.D: Primal Heuristics (cont.d)

- 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.

5.D: Primal Heuristics (cont.d)

- 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 has been found, the challenge becomes getting better and better ones, and local search heuristics come into the play for that.

5.D: Primal Heuristics (cont.d)

- 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 has been found, 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.

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/`.

- 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.

Summary and Conclusions

- We have seen
  1. The building blocks of a MIP solver.

  2. The nature and capability of a MIP solver as a sophisticated (heuristic) framework (including some performance variability drawbacks).

Summary and Conclusions

- We have seen
  1. The building blocks of a MIP solver.

  2. The nature and capability of a MIP solver as a sophisticated (heuristic) framework (including some performance variability drawbacks).

- In summary, MIP technology provides, through its commercial and noncommercial solvers, a challenging, reliable, flexible and effective environment for application-oriented optimization.

  1. challenging: a lot of good theoretical, methodological and experimental work is needed;
  2. reliable: the software tools are stable;
  3. flexible: it is open to hybridization, cannibalization, extensions;
  4. effective: problems that were conceived as impossible only few years ago can routinely be solved nowadays.

Summary and Conclusions

- We have seen
  1. The building blocks of a MIP solver.

  2. The nature and capability of a MIP solver as a sophisticated (heuristic) framework (including some performance variability drawbacks).

- In summary, MIP technology provides, through its commercial and noncommercial solvers, a challenging, reliable, flexible and effective environment for application-oriented optimization.

  1. challenging: a lot of good theoretical, methodological and experimental work is needed;
  2. reliable: the software tools are stable;
  3. flexible: it is open to hybridization, cannibalization, extensions;
  4. effective: problems that were conceived as impossible only few years ago can routinely be solved nowadays.

- All of the above look like solid reasons for developing the skills for using (and, why not, improving on) the MIP technology.