The MIP-Solving-Framework SCIP

Timo Berthold

Zuse Institut Berlin

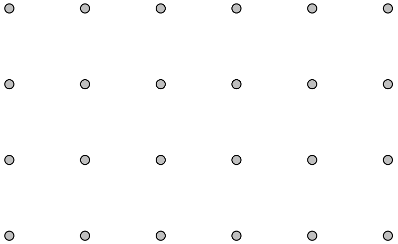
DFG Research Center MATHEON
Mathematics for key technologies

Berlin, 23.05.2007

Definition MIP

The optimization problem

is called a MIP
(mixed integer program).

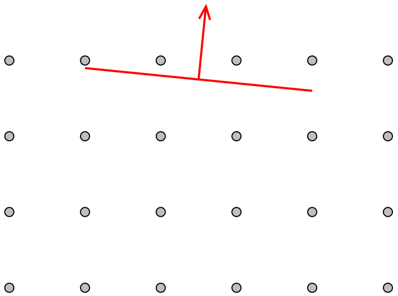


Definition MIP

The optimization problem

$$\min \quad c^T x$$

is called a MIP
(mixed integer program).

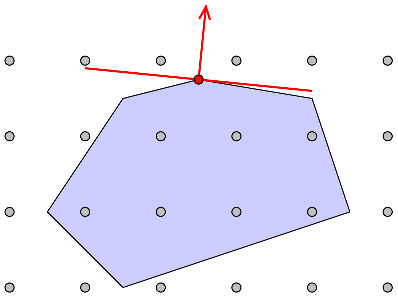


Definition MIP

The optimization problem

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \end{aligned}$$

is called a MIP
(mixed integer program).

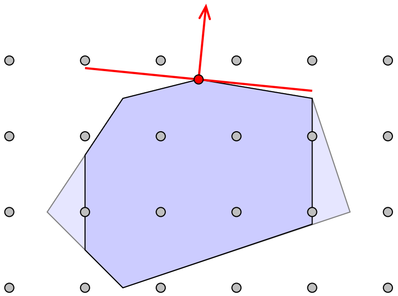


Definition MIP

The optimization problem

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & l \leq x \leq u \end{aligned}$$

is called a MIP
(mixed integer program).

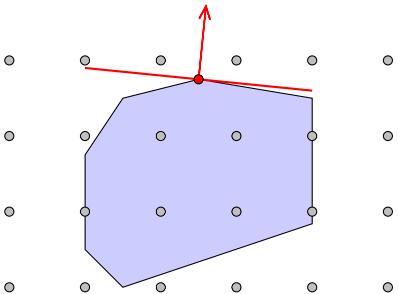


Definition MIP

The optimization problem

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & l \leq x \leq u \end{aligned}$$

is called a MIP
(mixed integer program).

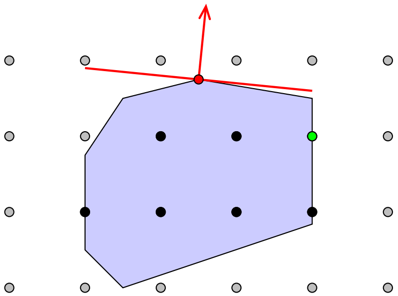


Definition MIP

The optimization problem

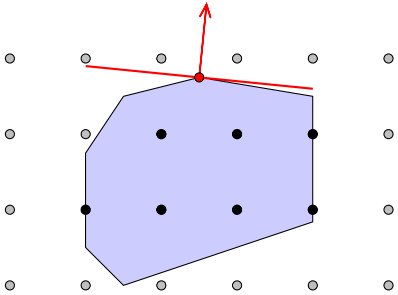
$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & l \le x \le u \\ & x_j \in \mathbb{Z} \qquad \forall j \in I \end{aligned}$$

is called a MIP
(mixed integer program).



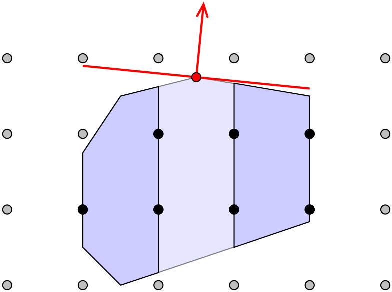
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



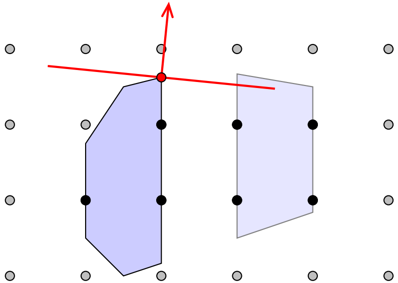
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



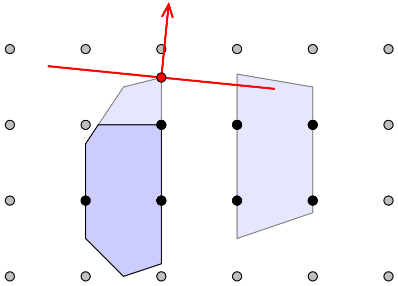
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



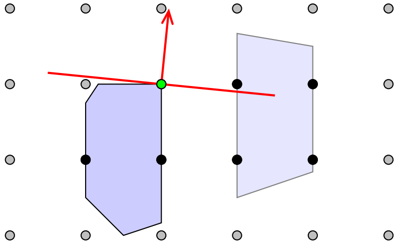
Exact methods

- ▷ Branch-And-Bound
- ▷ Cutting planes
- ▷ Combination: Branch-And-Cut



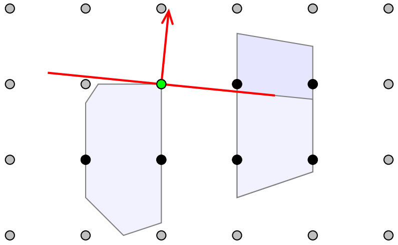
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



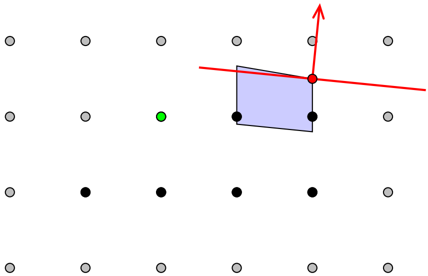
Exact methods

- ▷ Branch-And-Bound
- ▷ Cutting planes
- ▷ Combination: Branch-And-Cut



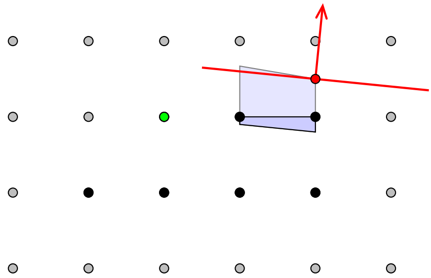
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



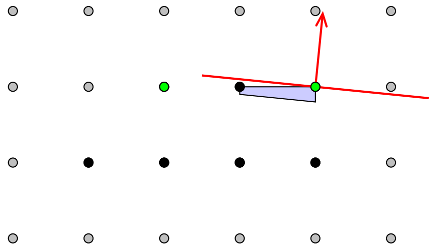
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



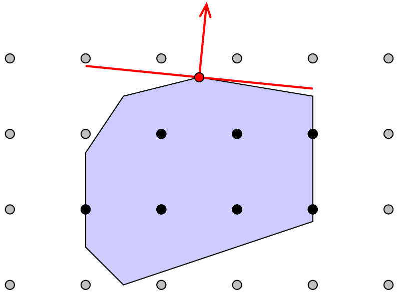
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



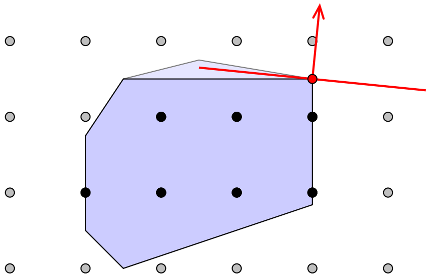
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



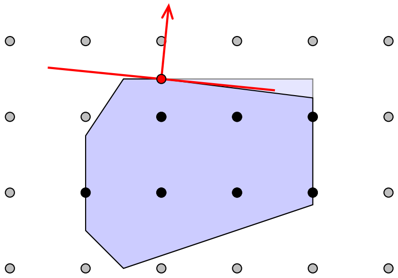
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



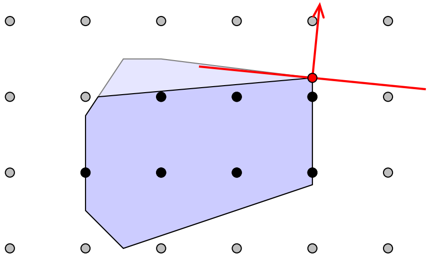
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



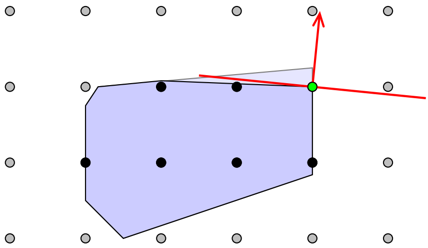
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



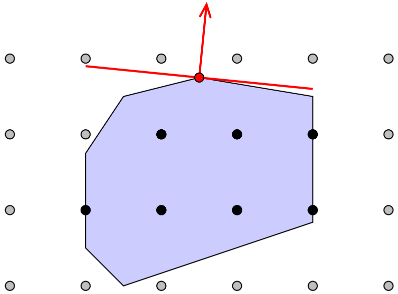
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



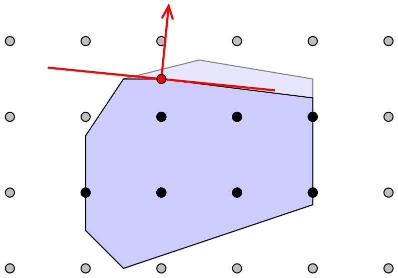
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



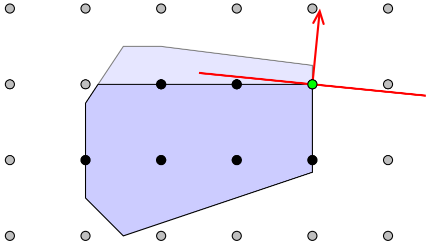
Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



Exact methods

- Branch-And-Bound
- Cutting planes
- Combination: Branch-And-Cut



Constraint Program (CP)

- General constraints
- Integer variables $\Rightarrow$ (CIP)

Examples

How do we solve CPs?

Constraint Program (CP)

- General constraints
- Integer variables $\Rightarrow$ (CIP)

Examples

- TSP as CP
- n-Queens

How do we solve CPs?

Constraint Program (CP)

- General constraints
- Integer variables $\Rightarrow$ (CIP)

Examples

- TSP as CP
- n-Queens

How do we solve CPs?

- Branching:
  - Divide into subproblems, solve recursively
- Domain Propagation:
  - Reductions in variables' domains "propagate"
  - E.g. $x_1 + 2x_2 \geq 5,\ x_1 \leq 2 \ \Rightarrow\ x_2 \geq 2$

SCIP ← CP+MIP

- SCIP combines technologies
- Standalone-solver for MIP
  - A bundle of MIP-solving-components as default plugins
  - MIP-solver as fast as CPlex 9.0
  - Underlying LP-solver: treated as blackbox
- Branch-Cut-And-Price-Framework for MIP and CIP
  - C++ wrapper classes for user plugins

SCIP ← CP+MIP

- SCIP combines technologies
- Standalone-solver for MIP
  - A bundle of MIP-solving-components as default plugins
  - MIP-solver as fast as CPlex 9.0
  - Underlying LP-solver: treated as blackbox
- Branch-Cut-And-Price-Framework for MIP and CIP
  - C++ wrapper classes for user plugins

Key data

- Since October 2002
- Achterberg, Wolter, B. et al.
- Approx. 220.000 lines of C code
- Free für academic use
- Available at `http://scip.zib.de`
- July 2007: version 1.0

Key data

- Since October 2002
- Achterberg, Wolter, B. et al.
- Approx. 220.000 lines of C code
- Free für academic use
- Available at `http://scip.zib.de`
- July 2007: version 1.0 . . . I hope so. . .

Types of plugins

Types of plugins

- Presolver: simplifies the problem in advance, strengthens structure

Types of plugins

- ▷ Presolver: simplifies the problem in advance, strengthens structure
- ▷ Node selection: which subproblem should be regarded next?

Types of plugins

- ▷ Presolver: simplifies the problem in advance, strengthens structure
- ▷ Node selection: which subproblem should be regarded next?
- ▷ Branching rule: how to divide the problem?

Types of plugins

- Presolver: simplifies the problem in advance, strengthens structure
- Node selection: which subproblem should be regarded next?
- Branching rule: how to divide the problem?
- Separator: adds cuts, improves dual bound

Types of plugins

- **Presolver:** simplifies the problem in advance, strengthens structure
- **Node selection:** which subproblem should be regarded next?
- **Branching rule:** how to divide the problem?
- **Separator:** adds cuts, improves dual bound
- **Constraint handler:** assures feasibility, strengthens formulation

Types of plugins

- Presolver: simplifies the problem in advance, strengthens structure
- Node selection: which subproblem should be regarded next?
- Branching rule: how to divide the problem?
- Separator: adds cuts, improves dual bound
- Constraint handler: assures feasibility, strengthens formulation
- Conflict handler: learns from infeasibility, improves dual bound

Types of plugins

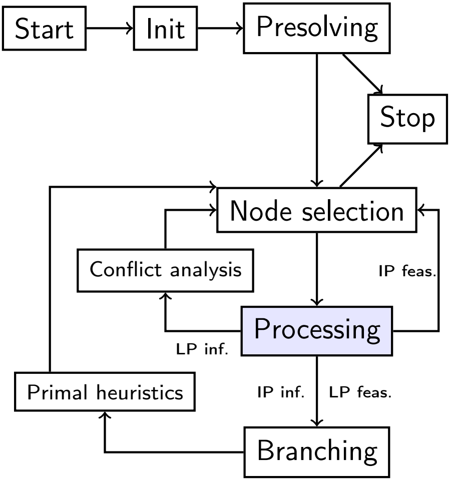
- Presolver: simplifies the problem in advance, strengthens structure
- Node selection: which subproblem should be regarded next?
- Branching rule: how to divide the problem?
- Separator: adds cuts, improves dual bound
- Constraint handler: assures feasibility, strengthens formulation
- Conflict handler: learns from infeasibility, improves dual bound
- Heuristic: searches solutions, improves primal bound

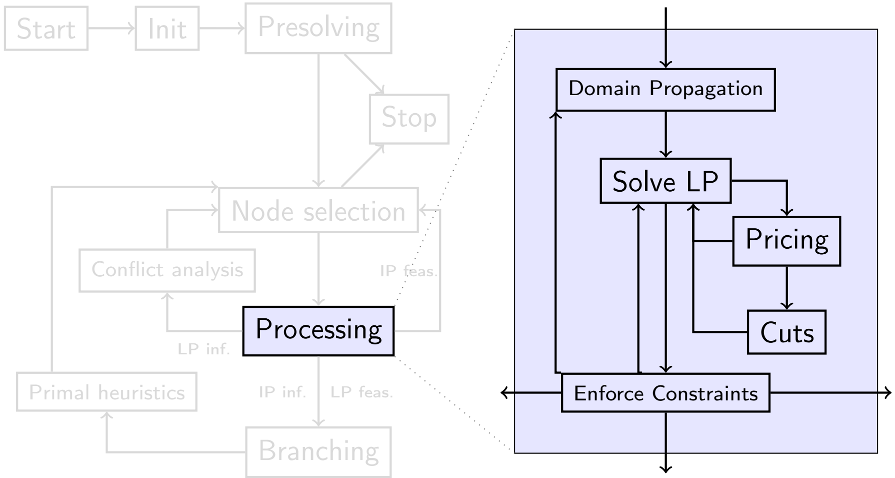
Types of plugins

- Presolver: simplifies the problem in advance, strengthens structure
- Node selection: which subproblem should be regarded next?
- Branching rule: how to divide the problem?
- Separator: adds cuts, improves dual bound
- Constraint handler: assures feasibility, strengthens formulation
- Conflict handler: learns from infeasibility, improves dual bound
- Heuristic: searches solutions, improves primal bound
- Propagator: simplifies problem, improves dual bound locally

Types of plugins

- Presolver: simplifies the problem in advance, strengthens structure
- Node selection: which subproblem should be regarded next?
- Branching rule: how to divide the problem?
- Separator: adds cuts, improves dual bound
- Constraint handler: assures feasibility, strengthens formulation
- Conflict handler: learns from infeasibility, improves dual bound
- Heuristic: searches solutions, improves primal bound
- Propagator: simplifies problem, improves dual bound locally
- Pricer: allows dynamic generation of variables


Start
Init
Presolving
Stop
Node selection
Conflict analysis
IP feas.
Processing
LP inf.
Primal heuristics
IP inf.
LP feas.
Branching


Start
Init
Presolving
Stop
Node selection
Conflict analysis
IP feas.
Processing
LP inf.
Primal heuristics
IP inf.
LP feas.
Branching
Domain Propagation
Solve LP
Pricing
Cuts
Enforce Constraints

Default plugins

- 5 presolvers
- 5 node selection rules
- 14 constraint handlers
- 8 separators
- 8 branching rules
- 4 conflict handlers
- 2 propagators
- 23 primal heuristics

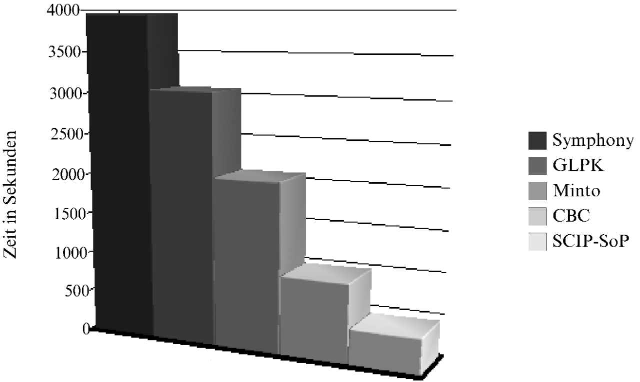
SCIP as a framework for a TSP-solver

Default plugins

SCIP as a framework for a TSP-solver

main program:	196 lines
TSP file reader:	407 lines
graph structure:	80 lines
subtour constraint:	793 lines
Gomory-Hu algo.:	658 lines
FarInsert heuristic:	354 lines
2-Opt heuristic:	304 lines
altogether:	2792 lines

Comparison With Other Free MIP-Solvers

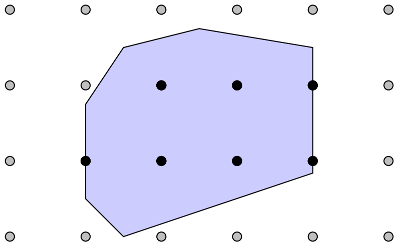


Characteristics

- Highest priority to feasibility
- Distinguish:
  - Start heuristics
  - Improvement heuristics
- Keep control of effort!
- Use available information

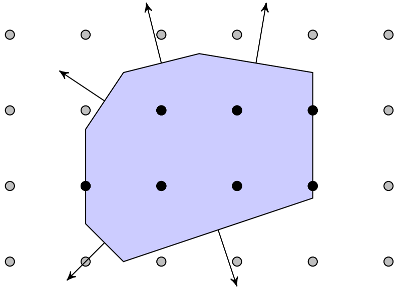
Used Information

- **Variables' locking numbers:**
  Potentially violated rows
- **Variables' pseudocosts:**
  Average objective change
- **Special points:**
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



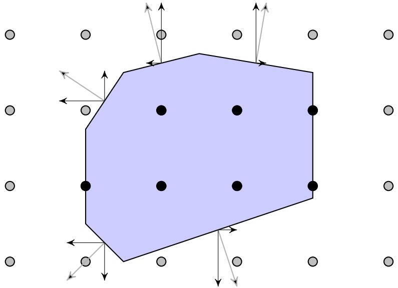
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



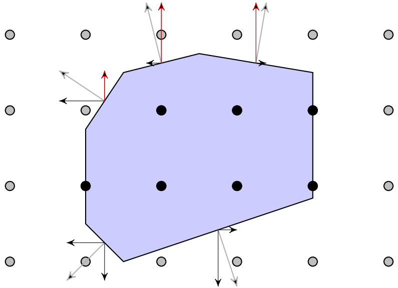
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



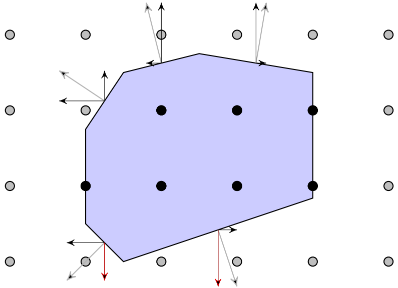
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



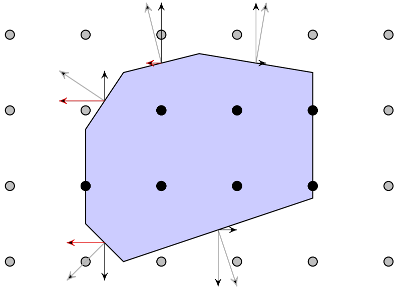
Used Information

- ▷ Variables' locking numbers:
  Potentially violated rows
- ▷ Variables' pseudocosts:
  Average objective change
- ▷ Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



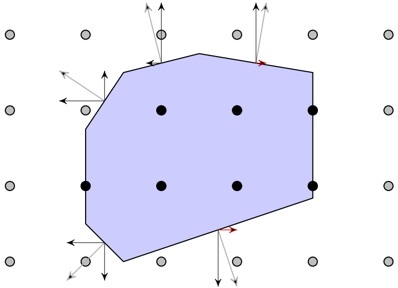
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



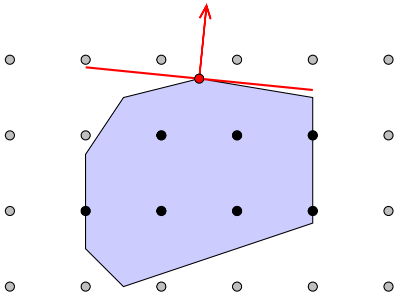
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



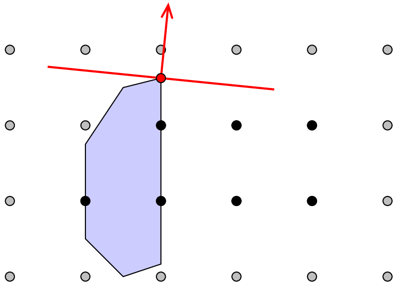
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



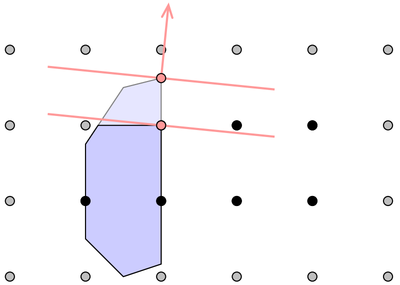
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



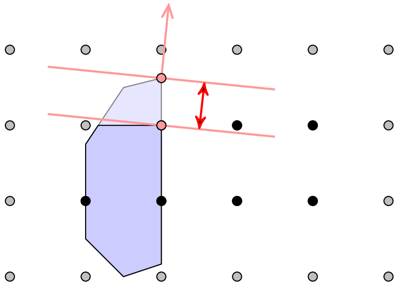
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



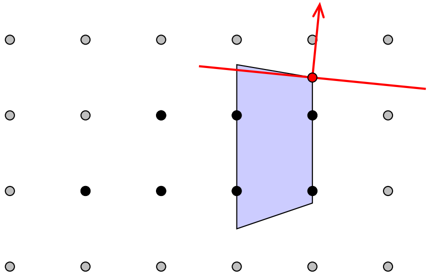
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



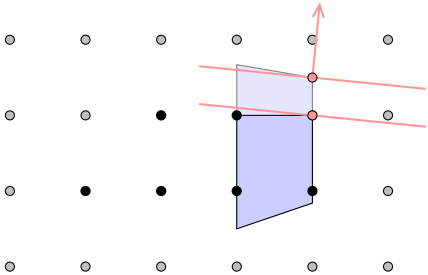
Used Information

- ▷ Variables' locking numbers:
  Potentially violated rows
- ▷ Variables' pseudocosts:
  Average objective change
- ▷ Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



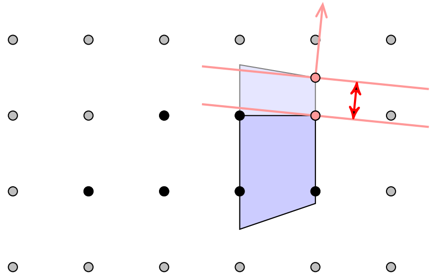
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



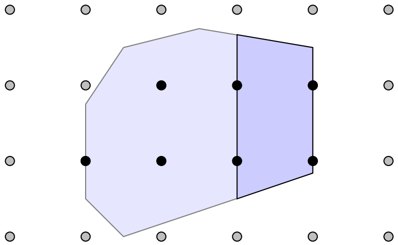
Used Information

- ▷ Variables' locking numbers:
  Potentially violated rows
- ▷ Variables' pseudocosts:
  Average objective change
- ▷ Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



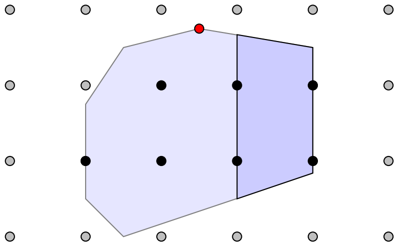
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



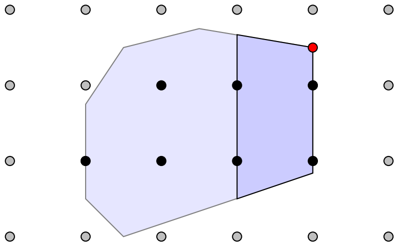
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



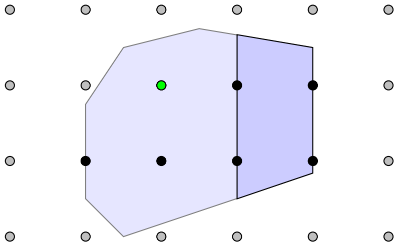
Used Information

- ▷ Variables' locking numbers:
  Potentially violated rows
- ▷ Variables' pseudocosts:
  Average objective change
- ▷ Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



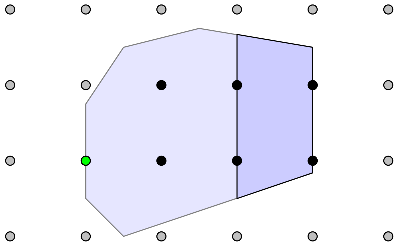
Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



Used Information

- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP optimum
  - Current best solution
  - Other known solutions



Approaches

- Rounding
- Diving: simulate DFS in the Branch-And-Bound-tree using some special branching rule
- Objective diving: manipulate objective function
- LNS: solve some sub-MIP
- Pivoting: manipulate simplex algorithm

Approaches

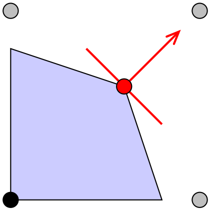
- **Rounding**
- **Diving:** simulate DFS in the Branch-And-Bound-tree using some special branching rule
- **Objective diving:** manipulate objective function
- **LNS:** solve some sub-MIP
- **Pivoting:** manipulate simplex algorithm

Implemented into SCIP

- 5 Rounding heuristics
- 8 Diving heuristics
- 3 Objective divers
- 4 LNS improvement heuristics

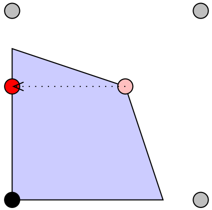
Ideas

- ▷ Simple Rounding always stays feasible,
- ▷ Rounding may violate constraints,
- ▷ Shifting may unfix integers.



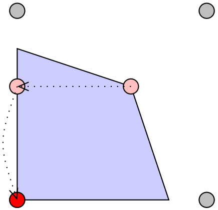
Ideas

- ▷ Simple Rounding always stays feasible,
- ▷ Rounding may violate constraints,
- ▷ Shifting may unfix integers.



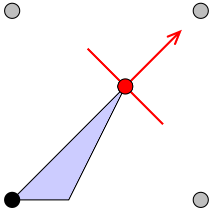
Ideas

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers.



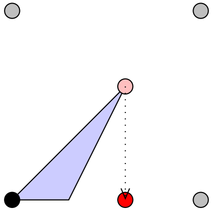
Ideas

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers.



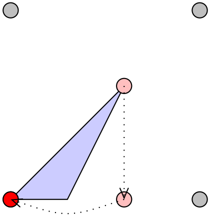
Ideas

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers.



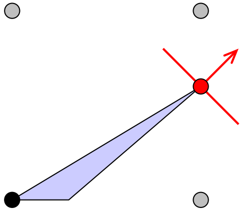
Ideas

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers.



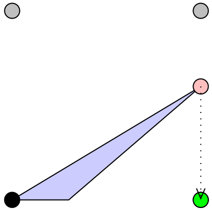
Ideas

- ▷ Simple Rounding always stays feasible,
- ▷ Rounding may violate constraints,
- ▷ Shifting may unfix integers.



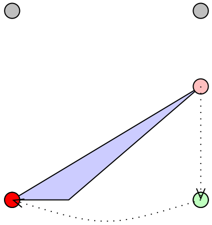
Ideas

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers.



Ideas

- ▷ Simple Rounding always stays feasible,
- ▷ Rounding may violate constraints,
- ▷ Shifting may unfix integers.



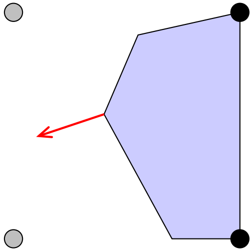
Idea: iteratively solve the LP and round a variable

Applied branching rules

- Fractional Diving: lowest fractionality
- Coefficient Diving: lowest locking number
- Linesearch Diving: highest increase since root
- Guided Diving: lowest difference to best known solution
- Pseudocost Diving: highest ratio of pseudocosts
- Vectorlength Diving: lowest ratio of objective change and number of rows containing the variable

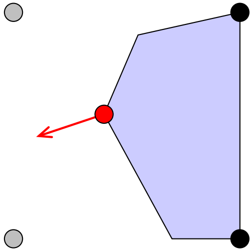
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



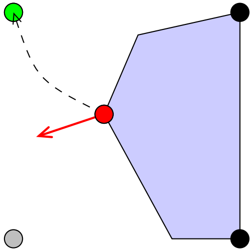
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



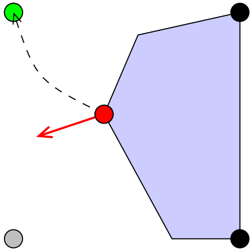
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



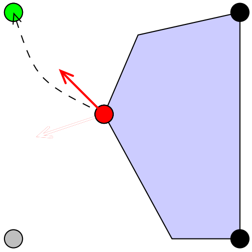
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



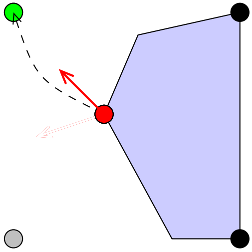
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



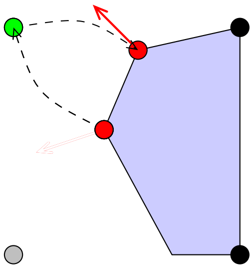
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



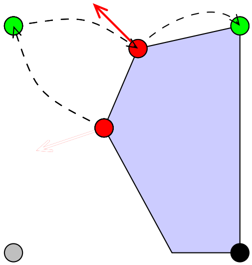
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



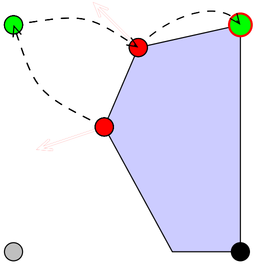
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



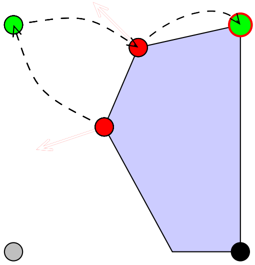
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change Objective;
7. Go to 1;



Improvements

- Objective $c^T x$ regarded at each step
- Algorithm able to resolve from cycling
- Quality of solutions much better

Improvements

- Objective $c^T x$ regarded at each step
- Algorithm able to resolve from cycling
- Quality of solutions much better

Results

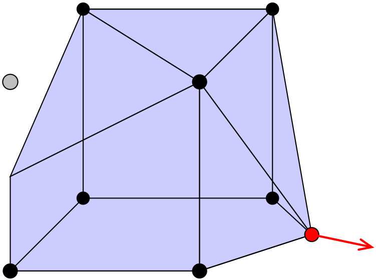
- Finds a solution for 74% of the test instances
- On average 5.5 seconds running time
- Optimality gap decreased from 107% to 38%

Algorithm

1. $\bar{x} \leftarrow$ LP optimum;
2. Fix all integral variables:
   $x_i := \bar{x}_i \ \ \forall i : \bar{x}_i \in \mathbb{Z}$;
3. Reduce domain of fractional variables:
   $x_i \in \{\lfloor \bar{x}_i \rfloor ; \lceil \bar{x}_i \rceil\}$;
4. Solve the resulting sub-MIP

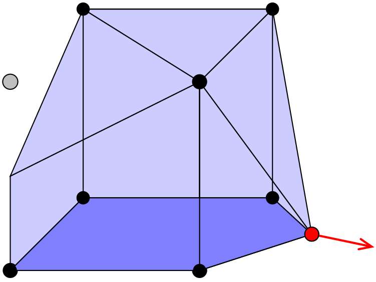
Algorithm

1. $\bar{x} \leftarrow$ LP optimum;
2. Fix all integral variables: $x_i := \bar{x}_i \;\; \forall i : \bar{x}_i \in \mathbb{Z}$;
3. Reduce domain of fractional variables: $x_i \in \{\lfloor \bar{x}_i \rfloor; \lceil \bar{x}_i \rceil\}$;
4. Solve the resulting sub-MIP



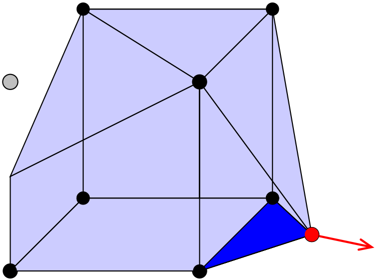
Algorithm

1. $\bar{x} \leftarrow$ LP optimum;
2. Fix all integral variables: $x_i := \bar{x}_i \;\; \forall i : \bar{x}_i \in \mathbb{Z}$;
3. Reduce domain of fractional variables: $x_i \in \{\lfloor \bar{x}_i \rfloor ; \lceil \bar{x}_i \rceil\}$;
4. Solve the resulting sub-MIP



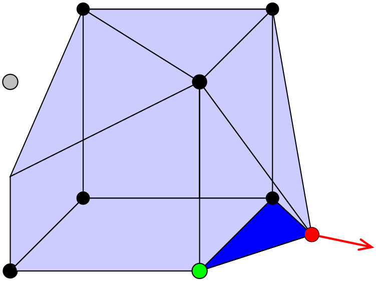
Algorithm

1. $\bar{x} \leftarrow$ LP optimum;
2. Fix all integral variables:
   $x_i := \bar{x}_i \;\; \forall i : \bar{x}_i \in \mathbb{Z}$;
3. Reduce domain of fractional variables:
   $x_i \in \{\lfloor \bar{x}_i \rfloor; \lceil \bar{x}_i \rceil\}$;
4. Solve the resulting sub-MIP



Algorithm

1. $\bar{x} \leftarrow$ LP optimum;
2. Fix all integral variables:
   $x_i := \bar{x}_i \;\; \forall i : \bar{x}_i \in \mathbb{Z}$;
3. Reduce domain of fractional variables:
   $x_i \in \{\lfloor \bar{x}_i \rfloor; \lceil \bar{x}_i \rceil\}$;
4. Solve the resulting sub-MIP



Observations

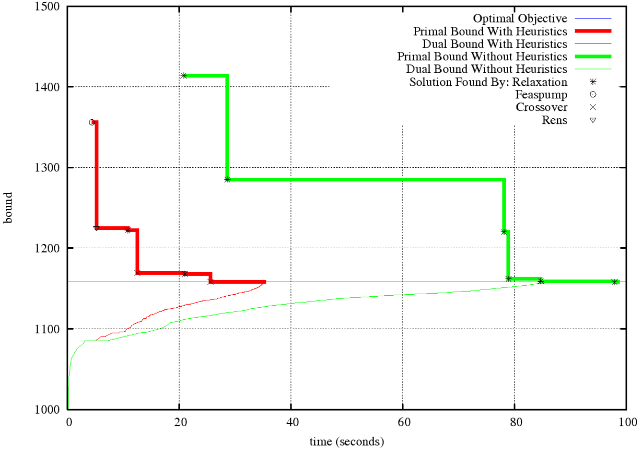
- Solutions found by Rens are roundings of $\bar{x}$
- Yields best possible rounding
- Yields certificate, if no rounding exists

Observations

- Solutions found by Rens are roundings of $\bar{x}$
- Yields best possible rounding
- Yields certificate, if no rounding exists

Results

- Approx. $\frac{2}{3}$ of the test instances are roundable
- Rens finds optimum for 20%!
- Dominates all other rounding heuristics



Instanz `aflow30a`: performance of SCIP with and without heuristics

Results

- Coordination important
- Positive side effects
- Improvement of overall performance
- SCIP with heuristics twice as fast

The MIP-Solving-Framework SCIP

Timo Berthold

Zuse Institut Berlin

DFG Research Center MATHEON

Mathematics for key technologies

Berlin, 23.05.2007