Primal Heuristics in SCIP

Timo Berthold

Zuse Institute Berlin

DFG Research Center MATHEON
Mathematics for key technologies

Berlin, 10/11/2007

Outline

1. Introduction
   - Basics
   - Integration Into SCIP
2. Available Heuristics
   - Rounding Heuristics
   - (Objective) Diving
   - LNS & Others
3. Remarks & Results

1. Introduction
   - Basics
   - Integration Into SCIP
2. Available Heuristics
   - Rounding Heuristics
   - (Objective) Diving
   - LNS & Others
3. Remarks & Results

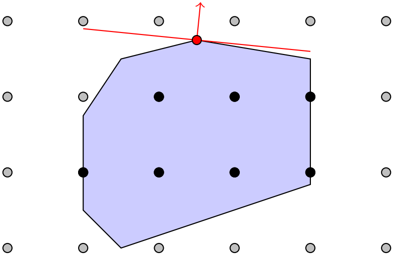
Outline

How Do We Solve MIPs?

Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

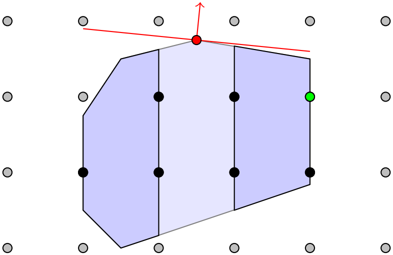
Heuristics



Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

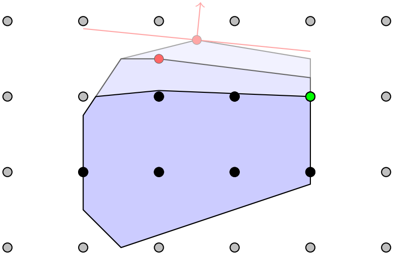
Heuristics



Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

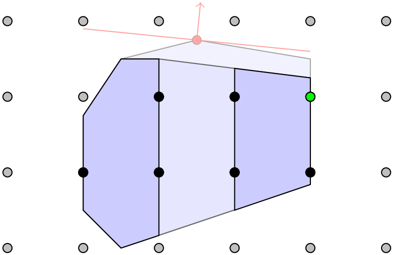
Heuristics



Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

Heuristics

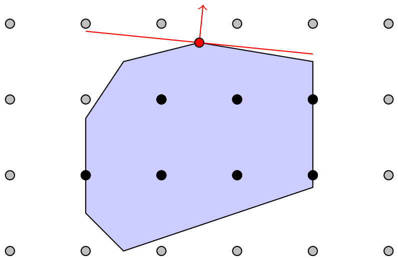


Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

Heuristics

- Often find good solutions

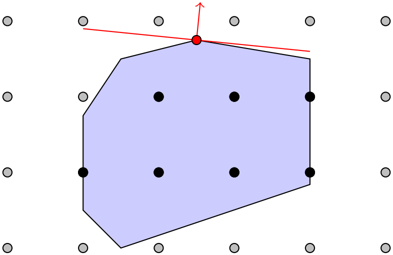


Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

Heuristics

- Often find good solutions
- in a reasonable time

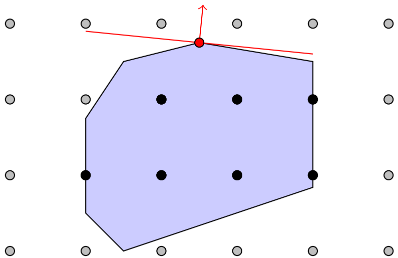


Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

Heuristics

- Often find good solutions
- in a reasonable time
- without any warranty!

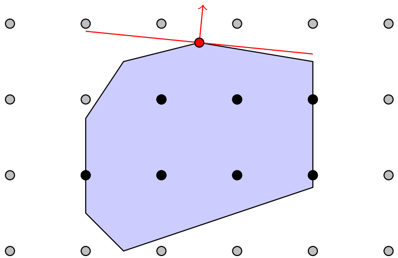


Exact methods

- Branch-And-Bound
- Cutting planes
- Branch-And-Cut

Heuristics

- Often find good solutions
- in a reasonable time
- without any warranty!
- $\rightsquigarrow$ Integrate into exact solver



Why use heuristics inside an exact solver?

- Able to prove feasibility of the model
- Often nearly optimal solution suffices in practice
- Feasible solutions guide remaining search process

Characteristics

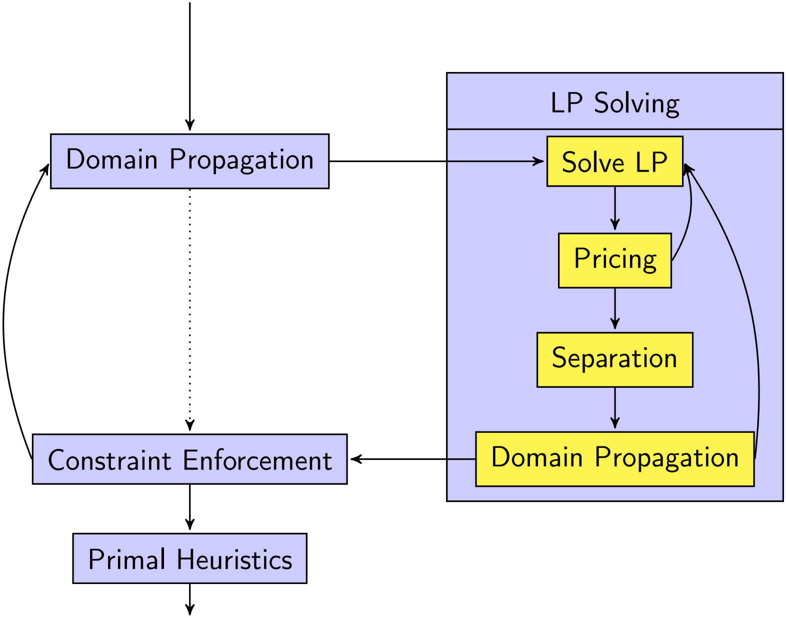
Why use heuristics inside an exact solver?

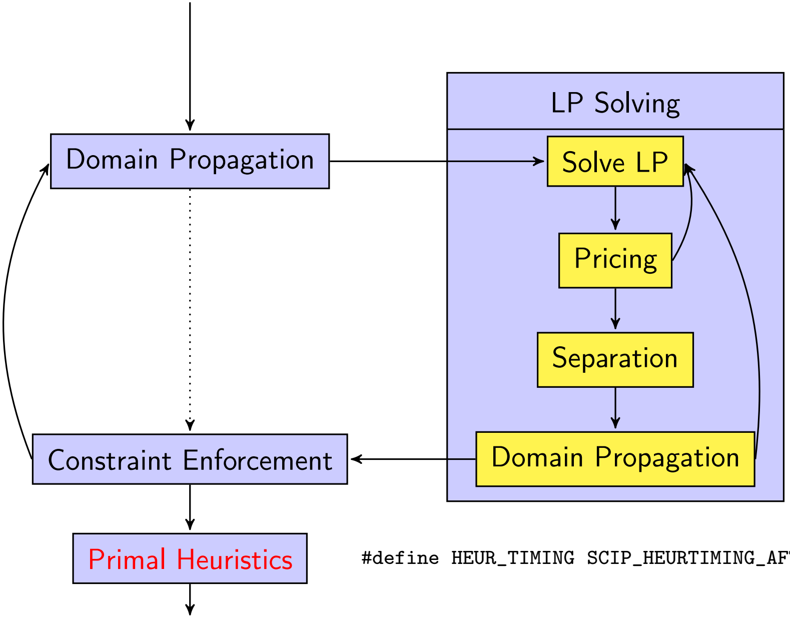
- ▷ Able to prove feasibility of the model
- ▷ Often nearly optimal solution suffices in practice
- ▷ Feasible solutions guide remaining search process

Characteristics

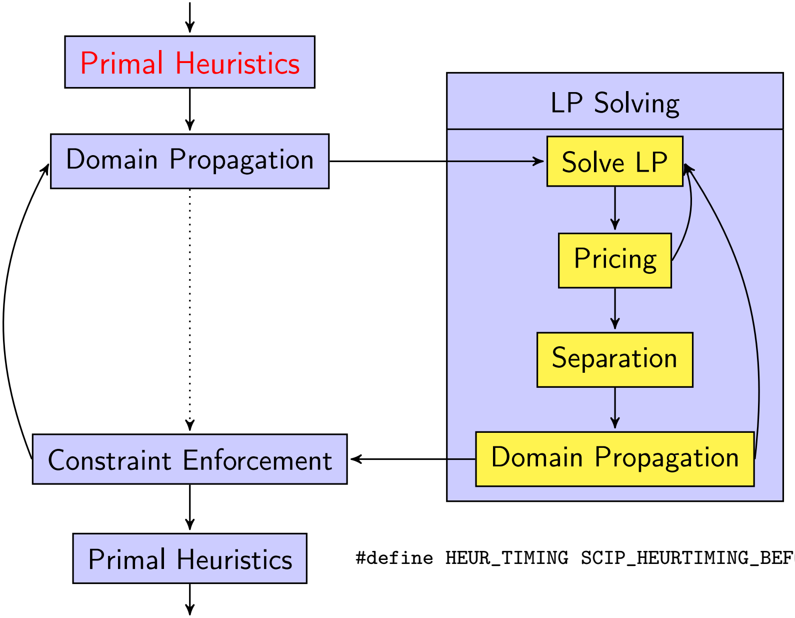
- ▷ Highest priority to feasibility
- ▷ Keep control of effort!
- ▷ Use as much information as you can get

```
SCIPincludeHeur(
 scip,                           // scip
 "Christofides",                 // HEUR_NAME
 "Start␣heuristic␣for␣TSP",      // HEUR_DESC
 'X',                            // HEUR_DISPCHAR
 -15000,                         // HEUR_PRIORITY
 0,                              // HEUR_FREQ
 0,                              // HEUR_FREQOFS
 0,                              // HEUR_MAXDEPTH
 SCIP_HEURTIMING_BEFORENODE      // HEUR_TIMING
);
```

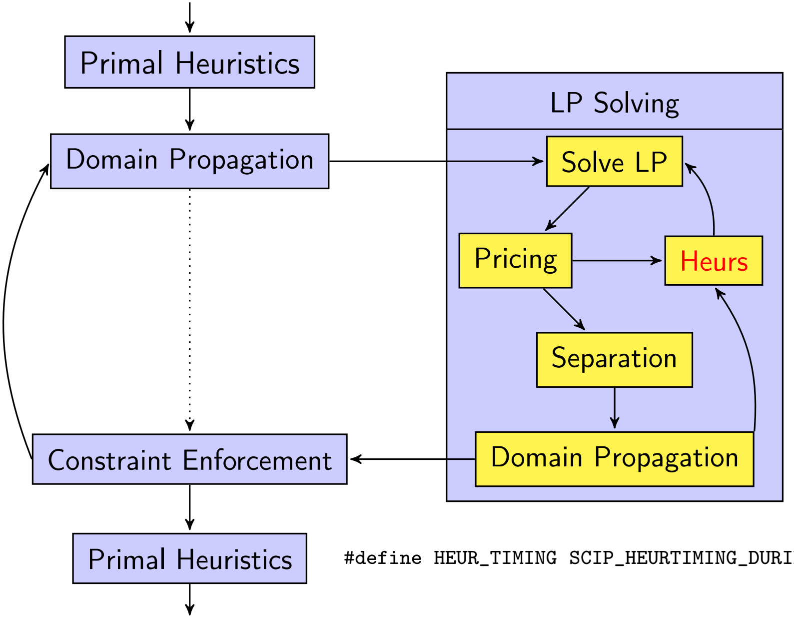

Domain Propagation
LP Solving
Solve LP
Pricing
Separation
Domain Propagation
Constraint Enforcement
Primal Heuristics




```
#define HEUR_TIMING SCIP_HEURTIMING_AFTERNODE
```




```
#define HEUR_TIMING SCIP_HEURTIMING_BEFORENODE
```




```
#define HEUR_TIMING SCIP_HEURTIMING_DURINGLPLOOP
```

Two main categories

- Start heuristics
  - Often already at root node
  - Mostly start from LP optimum
- Improvement heuristics
  - Require feasible solution
  - Normally at most once for each incumbent

Two main categories

- Start heuristics
  - Often already at root node
  - Mostly start from LP optimum
- Improvement heuristics
  - Require feasible solution
  - Normally at most once for each incumbent

```
#define HEUR_FREQOFS 0
```

Two main categories

- Start heuristics
  - Often already at root node
  - Mostly start from LP optimum
- Improvement heuristics
  - Require feasible solution
  - Normally at most once for each incumbent

```
if( SCIPgetLPSolstat(scip) != SCIP_LPSOLSTAT_OPTIMAL )
    return SCIP_OKAY;
```

Two main categories

- Start heuristics
  - Often already at root node
  - Mostly start from LP optimum
- Improvement heuristics
  - Require feasible solution
  - Normally at most once for each incumbent

```
if( SCIPgetNSols(scip) <= 0 )
   return SCIP_OKAY;
```

Two main categories

- Start heuristics
  - Often already at root node
  - Mostly start from LP optimum
- Improvement heuristics
  - Require feasible solution
  - Normally at most once for each incumbent

```
struct SCIP_HeurData
{
   SCIP_SOL* lastsol;
}
```

Five main approaches

- Rounding assign integer values to fractional variables
- Diving: DFS in the Branch-And-Bound-tree
- Objective diving: manipulate objective function
- LargeNeighborhoodSearch: solve some subMIP
- Pivoting: manipulate simplex algorithm

Five main approaches

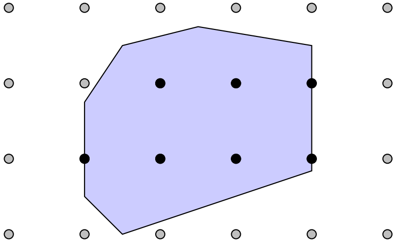
- Rounding assign integer values to fractional variables
- Diving: DFS in the Branch-And-Bound-tree
- Objective diving: manipulate objective function
- LargeNeighborhoodSearch: solve some subMIP
- Pivoting: manipulate simplex algorithm

Implemented into SCIP

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

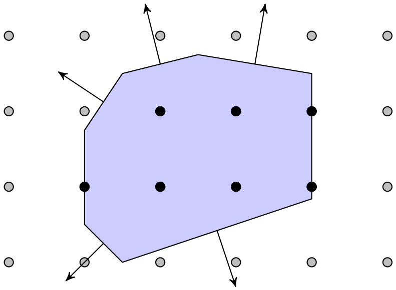
Statistics & points

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



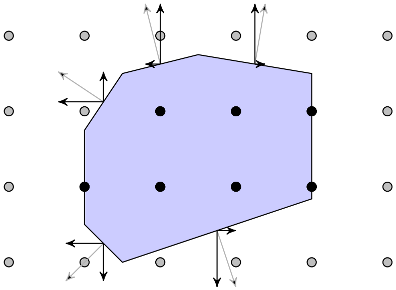
Statistics & points

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



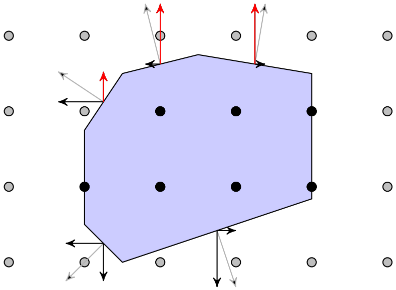
Statistics & points

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



Statistics & points

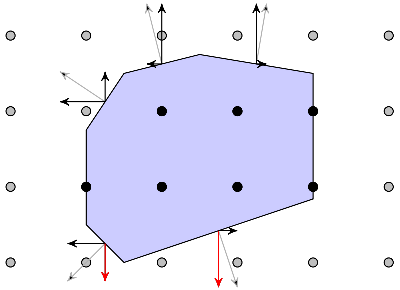
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
int nlocks = SCIPvarGetNLocksUp(var);
```

Statistics & points

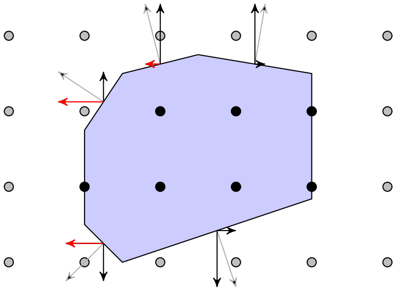
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
int nlocks = SCIPvarGetNLocksDown(var);
```

Statistics & points

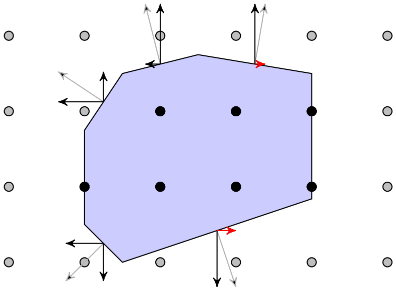
- Variables' locking numbers: Potentially violated rows
- Variables' pseudocosts: Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
int nlocks = SCIPvarGetNLocksDown(var);
```

Statistics & points

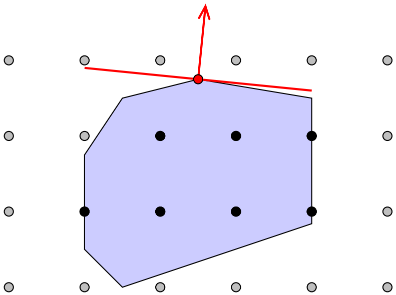
- Variables' locking numbers: Potentially violated rows
- Variables' pseudocosts: Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
int nlocks = SCIPvarGetNLocksUp(var);
```

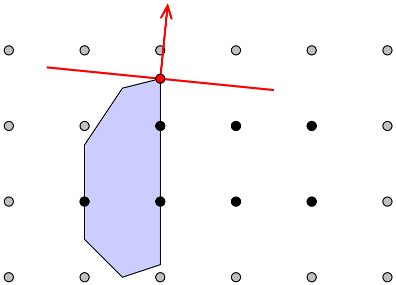
Statistics & points

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



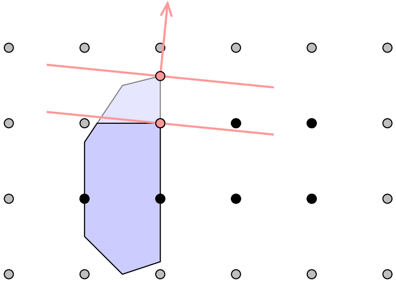
Statistics & points

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



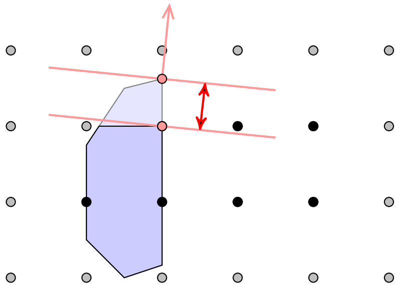
Statistics & points

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



Statistics & points

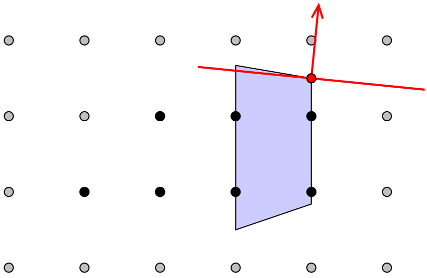
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Real pscost = SCIPgetVarPseudocost(scip, var, delta);
```

Statistics & points

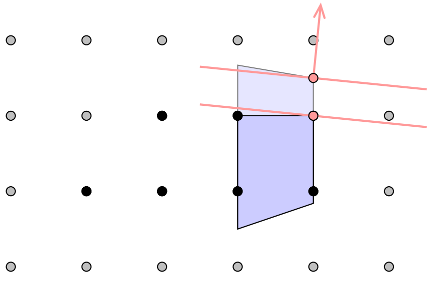
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Real pscost = SCIPgetVarPseudocost(scip, var, delta);
```

Statistics & points

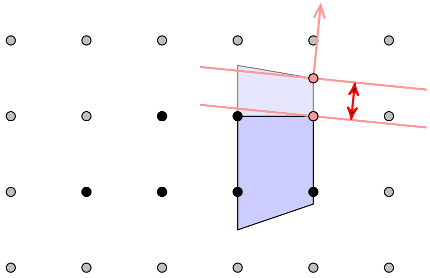
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Real pscost = SCIPgetVarPseudocost(scip, var, delta);
```

Statistics & points

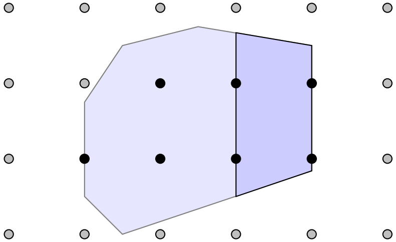
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Real pscost = SCIPgetVarPseudocost(scip, var, delta);
```

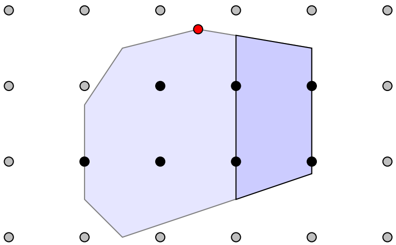
Statistics & points

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



Statistics & points

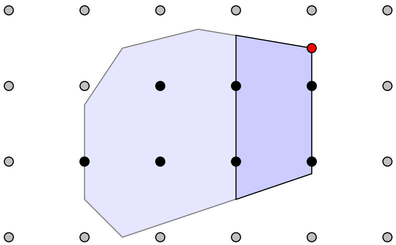
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Real rootsolval = SCIPvarGetRootSol(var);
```

Statistics & points

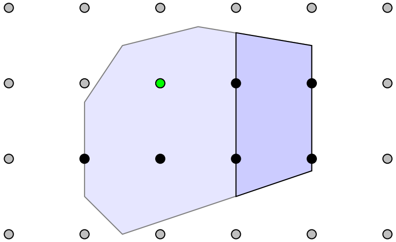
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Real solval = SCIPgetSolVal(scip, NULL, var);
```

Statistics & points

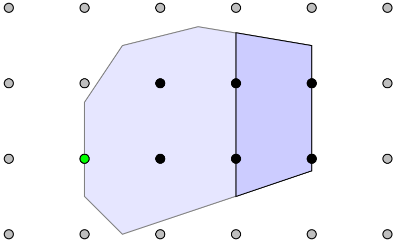
- Variables' locking numbers:
  Potentially violated rows
- Variables' pseudocosts:
  Average objective change
- Special points:
  - LP optimum at root node
  - Current LP solution
  - Current best solution
  - Other known solutions



```
SCIP_Sol* bestsol = SCIPgetBestSol(scip);
SCIP_Real solval = SCIPgetSolVal(scip, bestsol, var);
```

Statistics & points

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



```
SCIP_Sol** sols = SCIPgetSols(scip);
SCIP_Real solval = SCIPgetSolVal(scip, sols[i], var);
```

Guideline: Stay feasible!

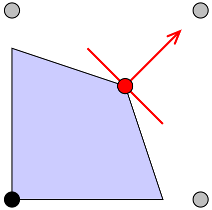
Features

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.

Guideline: Stay feasible!

Features

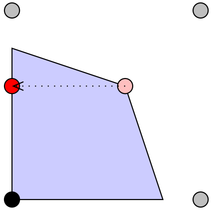
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

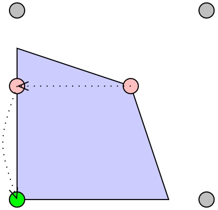
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

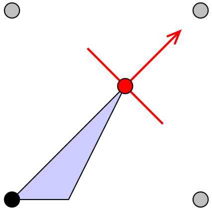
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

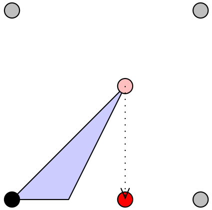
- ▷ Simple Rounding always stays feasible,
- ▷ Rounding may violate constraints,
- ▷ Shifting may unfix integers,
- ▷ Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

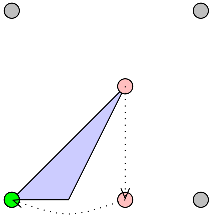
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

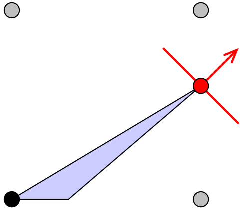
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

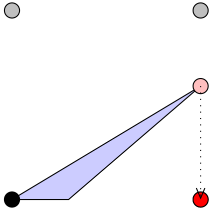
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

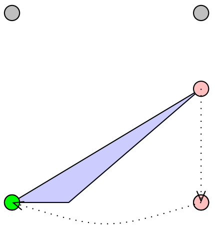
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

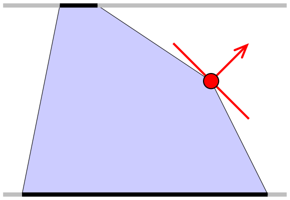
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

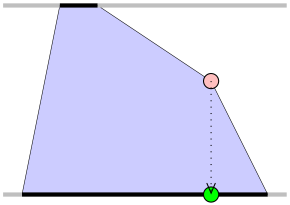
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

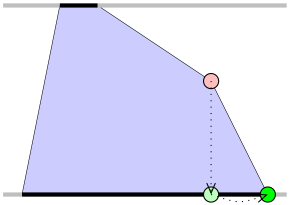
- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.



Guideline: Stay feasible!

Features

- Simple Rounding always stays feasible,
- Rounding may violate constraints,
- Shifting may unfix integers,
- Integer Shifting finally solves an LP.

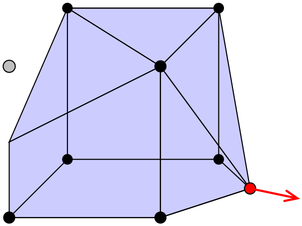


Algorithm

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 subMIP

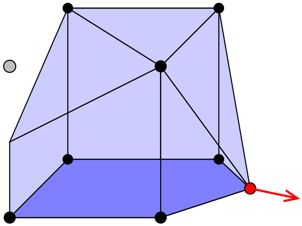
Algorithm

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 subMIP



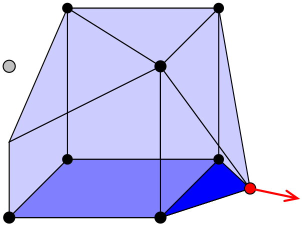
Algorithm

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 subMIP



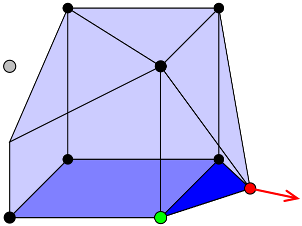
Algorithm

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 subMIP



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 subMIP



Observations

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

Results

Rens: An LNS Rounding Heuristic

Observations

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

Results

- 82 of 129 test instances are roundable
- Rens finds a global optimum for 23 instances!
- Dominates all other rounding heuristics

Idea

- Alternately solve the LP and round a variable
- Simulates DFS in Branch-And-Bound-tree
- Complementary target for branching
- Backtracking possible

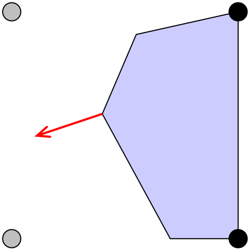
Applied branching rules

- 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

The Feasibility Pump (Fischetti, Lodi et al.)

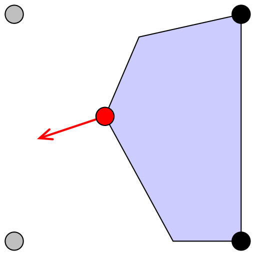
Algorithm

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



Algorithm

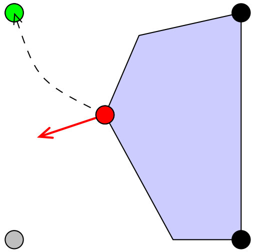
1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change objective;
7. Go to 1;



The Feasibility Pump (Fischetti, Lodi et al.)

Algorithm

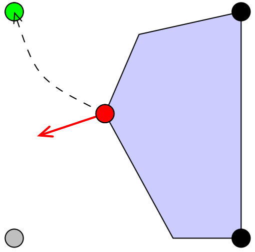
1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change objective;
7. Go to 1;



The Feasibility Pump (Fischetti, Lodi et al.)

Algorithm

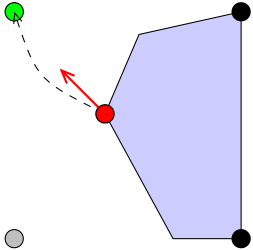
1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change objective;
7. Go to 1;



Algorithm

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

$$\Delta(x,\tilde{x}) = \sum |x_j - \tilde{x}_j|$$

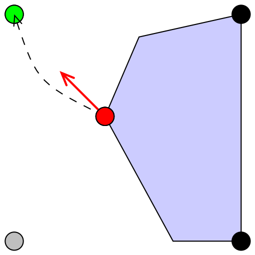


The Feasibility Pump (Fischetti, Lodi et al.)

Algorithm

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

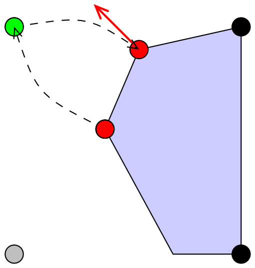
$\Delta(x,\tilde{x}) = \sum |x_j - \tilde{x}_j|$



Algorithm

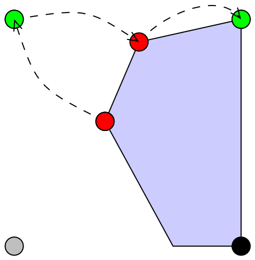
1. Solve LP;
2. Round LP optimum;
3. If feasible:
4.   Stop!
5. Else:
6.   Change objective;
7.   Go to 1;

$$\Delta(x,\tilde{x}) = \sum |x_j - \tilde{x}_j|$$



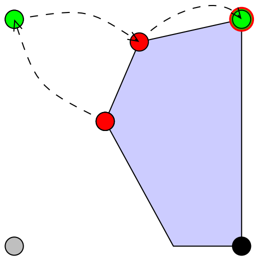
Algorithm

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



Algorithm

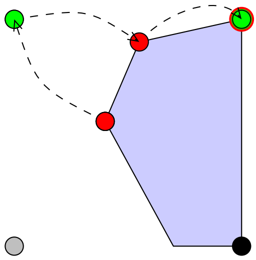
1. Solve LP;
2. Round LP optimum;
3. If feasible:
4.   Stop!
5. Else:
6.   Change objective;
7.   Go to 1;



The Feasibility Pump (Fischetti, Lodi et al.)

Algorithm

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



Objective Feasibility Pump (Achterberg & B.)

Improvements

- Objective $c^T x$ regarded at each step:
  $\tilde{\Delta} := (1-\alpha)\Delta(x) + \alpha c^T x$, with $\alpha \in [0,1]$
- Algorithm able to resolve from cycling
- Quality of solutions much better

Results

Objective Feasibility Pump (Achterberg & B.)

Improvements

- Objective $c^T x$ regarded at each step:
  $\tilde{\Delta} := (1-\alpha)\Delta(x) + \alpha c^T x$, with $\alpha \in [0, 1]$
- 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%

Other objective divers

- Objective Pseudocost Diving
  - analogon to Pseudocost Diving
  - Punishment by high objective coefficients
- Rootsolution Diving
  - analogon to Linesearch Diving
  - Objective function faded out

Idea: Create subMIP by fixing variables or adding constraints

Approaches

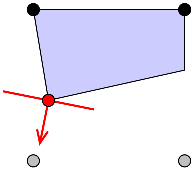
- Rins: fix variables equal in LP optimum and incumbent
- Crossover: fix variables equal in different feasible solutions
- Mutation: fix variables randomly
- Local Branching: add distance constraint wrt. incumbent

Combinatorial heuristics

- 1-Opt
  - Shifts value of integer variable
  - Solves LP afterwards

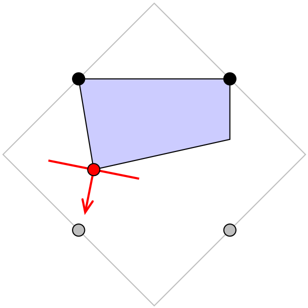
Combinatorial heuristics

- 1-Opt
  - Shifts value of integer variable
  - Solves LP afterwards
- Octane
  - Duality of cube and octahedron
  - Ray shooting algorithm



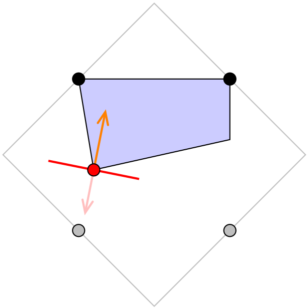
Combinatorial heuristics

- 1-Opt
  - Shifts value of integer variable
  - Solves LP afterwards
- Octane
  - Duality of cube and octahedron
  - Ray shooting algorithm



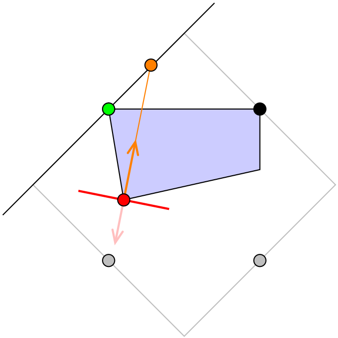
Combinatorial heuristics

- 1-Opt
  - Shifts value of integer variable
  - Solves LP afterwards
- Octane
  - Duality of cube and octahedron
  - Ray shooting algorithm



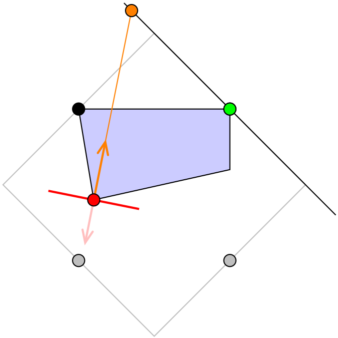
Combinatorial heuristics

- 1-Opt
  - Shifts value of integer variable
  - Solves LP afterwards
- Octane
  - Duality of cube and octahedron
  - Ray shooting algorithm



Combinatorial heuristics

- 1-Opt
  - Shifts value of integer variable
  - Solves LP afterwards
- Octane
  - Duality of cube and octahedron
  - Ray shooting algorithm



Some tips and tricks

- Use limits which respect the problem size
- LNS: stalling node limit
- Diving: try simple rounding on the fly
- Favor binaries over general integers
- Avoid cycling without randomness

Some tips and tricks

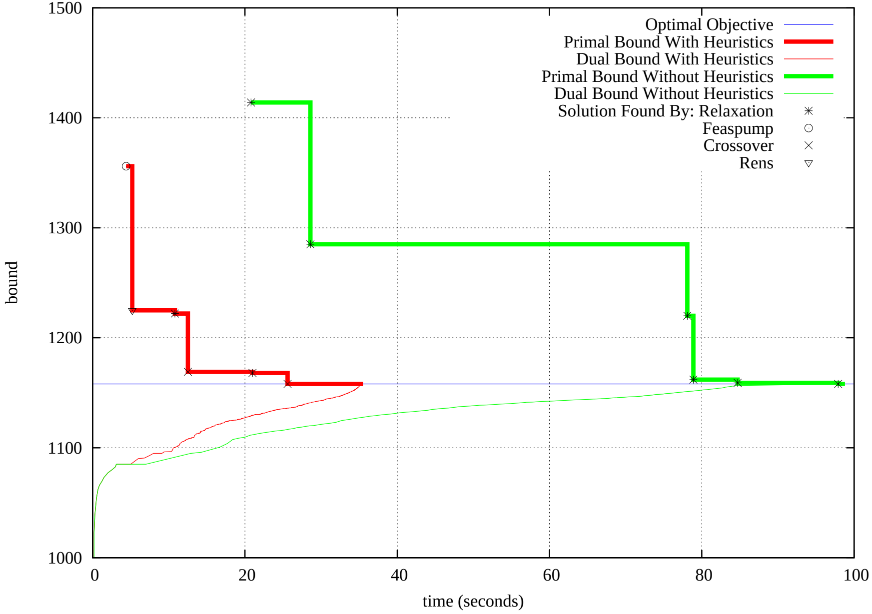
- Use limits which respect the problem size
- LNS: stalling node limit
- Diving: try simple rounding on the fly
- Favor binaries over general integers
- Avoid cycling without randomness

```
int nlpiterations = SCIPgetNNodeLPIterations(scip);
int maxlpiterations = heurdata→maxlpiterquot * nlpiterations;
```

Some tips and tricks

- Use limits which respect the problem size
- LNS: stalling node limit
- Diving: try simple rounding on the fly
- Favor binaries over general integers
- Avoid cycling without randomness

```
SCIP_CALL(
   SCIPsetLongintParam(subscip,"limits/stallnodes",nstallnodes)
);
```


Optimal Objective
Primal Bound With Heuristics
Dual Bound With Heuristics
Primal Bound Without Heuristics
Dual Bound Without Heuristics
Solution Found By: Relaxation
Feaspump
Crossover
Rens
bound
time (seconds)
1000
1100
1200
1300
1400
1500
0
20
40
60
80
100

Single heuristics

- Deactivating a single heuristic yields 1%-6% degradation
- No heuristic dominates the others
- Coordination important

Overall effect (SCIP 0.82b)

Single heuristics

- ▷ Deactivating a single heuristic yields 1%-6% degradation
- ▷ No heuristic dominates the others
- ▷ Coordination important

Overall effect (SCIP 0.82b)

- ▷ Better pruning, earlier fixing
- ▷ 7% less instances without any solution
- ▷ 5% more instances solved within one hour
- ▷ only half of the branch-and-bound-nodes
- ▷ only half of the solving time

Single heuristics

- ▷ Deactivating a single heuristic yields 1%-6% degradation
- ▷ No heuristic dominates the others
- ▷ Coordination important

Overall effect (SCIP 0.82b)

- ▷ Better pruning, earlier fixing
- ▷ 7% less instances without any solution
- ▷ 5% more instances solved within one hour
- ▷ only half of the branch-and-bound-nodes $\rightsquigarrow$ Not this much
- ▷ only half of the solving time $\rightsquigarrow$ in SCIP 1.00

Perspective & Further Research

To be implemented

- DINS heuristic
- k-opt for k > 1
- probing heuristics

Known problems

- Coordination could be strengthened
- Often poor for pure combinatorial problems

Primal Heuristics in SCIP

Timo Berthold

Zuse Institute Berlin

DFG Research Center MATHEON
Mathematics for key technologies

Berlin, 10/11/2007