Using SCIP to Solve Your Favorite Integer Optimization Problem

Gregor Hendel, hendel@zib.de, www.zib.de/hendel
Monash University, Melbourne
March 12, 2019
A research institute and computing center of the State of Berlin with research units:

- Numerical Analysis and Modeling
- Visualization and Data Analysis
- Optimization:
  - Energy – Transportation – Health – Mathematical Optimization Methods
- Scientific Information Systems
- Computer Science and High Performance Computing
What is SCIP?

SCIP (Solving Constraint Integer Programs) . . .

- provides a full-scale MIP and MINLP solver,
- is constraint based,
- incorporates
  - MIP features (cutting planes, LP relaxation), and
  - MINLP features (spatial branch-and-bound, OBBT)
  - CP features (domain propagation),
  - SAT-solving features (conflict analysis, restarts),
- is a branch-cut-and-price framework,
- has a modular structure via plugins,
- is free for academic purposes,
- and is available in source-code under http://scip.zib.de!
Meet the SCIP Team

31 active developers

- 7 running Bachelor and Master projects
- 16 running PhD projects
- 8 postdocs and professors

4 development centers in Germany

- ZIB: SCIP, SoPlex, UG, ZIMPL
- TU Darmstadt: SCIP and SCIP-SDP
- FAU Erlangen-Nürnberg: SCIP
- RWTH Aachen: GCG

Many international contributors and users

- more than 10,000 downloads per year from 100+ countries

Careers

- 10 awards for Masters and PhD theses: MOS, EURO, GOR, DMV
- 7 former developers are now building commercial optimization software at CPLEX, FICO Xpress, Gurobi, MOSEK, and GAMS
SCIP – Solving Constraint Integer Programs

The SCIP Optimization Suite

(Mixed-) Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

http://scip.zib.de
SCIP – Solving Constraint Integer Programs

The SCIP Optimization Suite

(Mixed-) Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

http://scip.zib.de
Toolbox for generating and solving constraint integer programs, in particular Mixed Integer (Non-)Linear Programs.

free for academic use, available in source code
Toolbox for generating and solving constraint integer programs, in particular Mixed Integer (Non-)Linear Programs.

• free for academic use, available in source code

ZIMPL
• model and generate LPs, MIPs, and MINLPs

SCIP
• MIP, MINLP and CIP solver, branch-cut-and-price framework

SoPlex
• revised primal and dual simplex algorithm

GCG
• generic branch-cut-and-price solver

UG
• framework for parallelization of MIP and MINLP solvers
The SCIP C API

- **C code and documentation**
  - more than 800,000 lines of C code, 20% documentation
  - over 50,000 assertions and 5,000 debug messages
  - HowTos: plugins types, debugging, automatic testing
  - 10 examples illustrating the use of SCIP
  - 5 problem specific SCIP applications to solve Coloring, Steiner Tree, or Multiobjective problems.

- **Interface and usability**
  - Cross-platform availability due to CMake
  - user-friendly interactive shell
  - C++ wrapper classes
  - LP solvers: SoPlex, CPLEX, Gurobi, Xpress, CLP, MOSEK, QSopt
  - NLP solvers: IPOPT, FilterSQP, WORHP
  - over 2,300 parameters and 15 emphasis settings
• interactive shell supports 11 different input formats
  → cip, cnf, flatzinc, rlp, lp, mps, opb, pip, wbo, zimpl, smps

• C API/callable library

• C++ wrapper classes

• Python interface

• Java JNI interface

• AMPL

• GAMS

• Matlab (see also OPTI toolbox,
  http://www.i2c2.aut.ac.nz/Wiki/OPTI/)
Getting help

If you should ever get stuck, you can ...

1. type `help` in the interactive shell
2. read the documentation [http://scip.zib.de/doc/html](http://scip.zib.de/doc/html) → FAQ, HowTos for each plugin type, debugging, automatic testing, ...
3. search or post on Stack Overflow using the tag `scip` (more than 100 questions already answered)
4. active mailing list `scip@zib.de` (350+ members)
   - search the mailing list archive (append site:listserv/pipermail/scip)
SCIP – Solving Constraint Integer Programs

The SCIP Optimization Suite

(Mixed-) Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

http://scip.zib.de
What is a good diet?

<table>
<thead>
<tr>
<th>Product</th>
<th>Servings</th>
<th>Energy (kj)</th>
<th>Protein (g)</th>
<th>Calcium (g)</th>
<th>Price (ct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oatmeal</td>
<td>4</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Chicken</td>
<td>3</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>Eggs</td>
<td>2</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>Milk</td>
<td>8</td>
<td>160</td>
<td>8</td>
<td>284</td>
<td>9</td>
</tr>
<tr>
<td>Pie</td>
<td>2</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>Pork</td>
<td>2</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
</tr>
</tbody>
</table>

Needed per day:

- Energy: 2000 kj
- Protein: 55g
- Calcium: 800 mg
What is a good diet?

<table>
<thead>
<tr>
<th>Product</th>
<th>Servings</th>
<th>Energy(kj)</th>
<th>Protein(g)</th>
<th>Calcium(g)</th>
<th>Price(ct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oatmeal</td>
<td>4</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Chicken</td>
<td>3</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>Eggs</td>
<td>2</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>Milk</td>
<td>8</td>
<td>160</td>
<td>8</td>
<td>284</td>
<td>9</td>
</tr>
<tr>
<td>Pie</td>
<td>2</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>Pork</td>
<td>2</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
</tr>
</tbody>
</table>

Needed per day:

- Energy: 2000 kj
- Protein: 55g
- Calcium: 800 mg

Task: Find a diet satisfying all nutritional requirements at minimum cost!
Find a diet satisfying all nutritional requirements at minimum cost!

Our nutrition plan should be expressed as $x \in \mathbb{R}_{\geq 0}^6$.

- Energy: 2000 kj
- Protein: 55g
- Calcium: 800 mg
Find a diet satisfying all nutritional requirements at minimum cost!

Our nutrition plan should be expressed as $x \in \mathbb{R}^6_{\geq 0}$.

\[
\min 3x_{\text{oat}} + 24x_{\text{Chi}} + 13x_{\text{Egg}} + 9x_{\text{Mil}} + 20x_{\text{Pie}} + 19x_{\text{Por}}
\]

- Energy: $2000$ kj
- Protein: $55g$
- Calcium: $800$ mg
The Diet LP

Find a diet satisfying all nutritional requirements at minimum cost!

Our nutrition plan should be expressed as $x \in \mathbb{R}_0^6$.

$$\min 3x_{\text{oat}} + 24x_{\text{Chi}} + 13x_{\text{Egg}} + 9x_{\text{Mil}} + 20x_{\text{Pie}} + 19x_{\text{Por}}$$

- Energy: 2000 kj
  $$110x_{\text{oat}} + 205x_{\text{Chi}} + 160x_{\text{Egg}} + 160x_{\text{Mil}} + 420x_{\text{Pie}} + 260x_{\text{Por}} \geq 2000$$

- Protein: 55g

- Calcium: 800 mg
Find a diet satisfying all nutritional requirements at minimum cost!

Our nutrition plan should be expressed as $x \in \mathbb{R}^6_{\geq 0}$.

$$\min 3x_{\text{oat}} + 24x_{\text{Chi}} + 13x_{\text{Egg}} + 9x_{\text{Mil}} + 20x_{\text{Pie}} + 19x_{\text{Por}}$$

- Energy: 2000 kj
  $$110x_{\text{oat}} + 205x_{\text{Chi}} + 160x_{\text{Egg}} + 160x_{\text{Mil}} + 420x_{\text{Pie}} + 260x_{\text{Por}} \geq 2000$$

- Protein: 55g
  $$4x_{\text{oat}} + 32x_{\text{Chi}} + 13x_{\text{Egg}} + 8x_{\text{Mil}} + 4x_{\text{Pie}} + 14x_{\text{Por}} \geq 55$$

- Calcium: 800 mg
  $$2x_{\text{oat}} + 12x_{\text{Chi}} + 54x_{\text{Egg}} + 284x_{\text{Mil}} + 22x_{\text{Pie}} + 80x_{\text{Por}} \geq 800$$
Diet Problem
Only an integer vector $x \in \mathbb{Z}^6_{\geq 0}$ is truly feasible, because food such as eggs are unsplittable.

$$\begin{align*}
\text{min} & \quad (3 \quad 24 \quad 13 \quad 9 \quad 20 \quad 19) \quad x \\
\text{s.t.} & \quad \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 2 & 12 & 54 & 284 & 22 & 80 \end{pmatrix} x \geq \begin{pmatrix} 2000 \\ 55 \\ 800 \end{pmatrix} \\
& \quad x \geq 0 \\
& \quad x_i \in \mathbb{Z} \quad \forall i \in \{1, \ldots, 6\}
\end{align*}$$
Diet Problem
Only an integer vector $x \in \mathbb{Z}^6_{\geq 0}$ is truly feasible, because food such as eggs are unsplittable.

\[
\begin{align*}
\text{min} & \quad (3 \ 24 \ 13 \ 9 \ 20 \ 19) \ x \\
\text{s.t.} & \quad \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\
4 & 32 & 13 & 8 & 4 & 14 \\
2 & 12 & 54 & 284 & 22 & 80 \end{pmatrix} x \geq \begin{pmatrix} 2000 \\
55 \\
800 \end{pmatrix} \\
& \quad x \geq 0 \\
& \quad x_i \in \mathbb{Z} \quad \forall i \in \{1, \ldots, 6\}
\end{align*}
\]

General MIP
\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax \geq b \\
& \quad x \geq 0 \\
& \quad x_i \in \mathbb{Z} \quad \forall i \in I \subseteq \{1, \ldots, n\}
\end{align*}
\]
A simple graph $G$ is a tuple $(V, E)$ of
- $|V| = n$ nodes and
- $m = |E|$ edges connecting pairs of nodes, i.e. $E \subseteq \binom{V}{2}$.
- nodes may have weights/demands $w : V \to \mathbb{R}$
- edges may have associated costs $c : E \to \mathbb{R}$
Most of the classical optimization problems on graphs can be formulated as Mixed Integer Programs.

- Max. stable set
- Min. vertex cover
- Max. matching
- Graph coloring
- Traveling salesman problem
- Steiner tree problem
- ...
Most of the classical optimization problems on graphs can be formulated as Mixed Integer Programs.

- **Max. stable set**
- **Min. vertex cover**
- **Max. matching**
- **Graph coloring**
- **Traveling salesman problem**
- **Steiner tree problem**
- ...
Most of the classical optimization problems on graphs can be formulated as Mixed Integer Programs.

- Max. stable set
- Min. vertex cover
- Max. matching
- Graph coloring
- Traveling salesman problem
- Steiner tree problem
- ...
SCIP – Solving Constraint Integer Programs

The SCIP Optimization Suite

(Mixed-) Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

http://scip.zib.de
**Definition (TSP)**

Given a complete graph $G = (V, E)$ and distances $d_e$ for all $e \in E$:

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.
Definition (TSP)
Given a complete graph \( G = (V, E) \) and distances \( d_e \) for all \( e \in E \):

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.
Definition (TSP)
Given a complete graph
\( G = (V, E) \) and distances \( d_e \) for all \( e \in E \):

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.
### Mixed Integer Program

Objective function:

- linear function

Feasible set:

- described by linear constraints

Variable domains:

- real or integer values

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax \leq b \\
& \quad (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C
\end{align*}
\]

### Constraint Program

Objective function:

- arbitrary function

Feasible set:

- given by arbitrary constraints

Variable domains:

- arbitrary (usually finite)

\[
\begin{align*}
\text{min} & \quad c(x) \\
\text{s.t.} & \quad x \in F \\
& \quad (x_I, x_N) \in \mathbb{Z}^I \times \mathcal{X}
\end{align*}
\]
Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables

- $x_e = 1$ if edge $e$ is used
TSP – Integer Programming Formulation

Given

- complete graph \( G = (V, E) \)
- distances \( d_e > 0 \) for all \( e \in E \)

Binary variables

- \( x_e = 1 \) if edge \( e \) is used

\[
\begin{align*}
\text{min} & \quad \sum_{e \in E} d_e x_e \\
\text{subject to} & \quad \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in \{0, 1\} & \forall e \in E
\end{align*}
\]
TSP – Integer Programming Formulation

Given
- complete graph \( G = (V, E) \)
- distances \( d_e > 0 \) for all \( e \in E \)

Binary variables
- \( x_e = 1 \) if edge \( e \) is used

\[
\begin{align*}
\text{min} \quad & \sum_{e \in E} d_e x_e \\
\text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset \\
& x_e \in \{0, 1\} \quad \forall e \in E
\end{align*}
\]
TSP – Integer Programming Formulation

Given

\begin{itemize}
  \item complete graph \( G = (V, E) \)
  \item distances \( d_e > 0 \) for all \( e \in E \)
\end{itemize}

Binary variables

\begin{itemize}
  \item \( x_e = 1 \) if edge \( e \) is used
\end{itemize}

\[
\begin{align*}
\min & \quad \sum_{e \in E} d_e x_e \\
\text{subject to } & \quad \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in \{0, 1\} \quad \forall e \in E
\end{align*}
\]

subtour elimination
TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables

- $x_e = 1$ if edge $e$ is used

\[
\text{min} \quad \sum_{e \in E} d_e x_e
\]

subject to

- $\sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V$
- $\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset$
- $x_e \in \{0, 1\} \quad \forall e \in E$
Given

- complete graph $G = (V, E)$
- for each $e \in E$ a distance $d_e > 0$

Integer variables

- $x_v$ position of $v \in V$ in tour
TSP – Constraint Programming Formulation

Given
- complete graph \( G = (V, E) \)
- for each \( e \in E \) a distance \( d_e > 0 \)

Integer variables
- \( x_v \) position of \( v \in V \) in tour

\[
\begin{align*}
\min \ & \text{length}(x_1, \ldots, x_n) \\
\text{subject to} \ & \text{alldifferent}(x_1, \ldots, x_n) \\
x_v \in \{1, \ldots, n\} & \quad \forall v \in V
\end{align*}
\]
What is a Constraint Integer Program?

**Constraint Integer Program**

Objective function:

- linear function

Feasible set:

- described by arbitrary constraints

Variable domains:

- real or integer values

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad x \in F \\
& \quad (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C
\end{align*}
\]

Remark:

- arbitrary objective or variables modeled by constraints

Gregor Hendel – Using SCIP
**Constraint Integer Program**

**Objective function:**

- **linear function**

**Feasible set:**

- described by arbitrary constraints

**Variable domains:**

- real or integer values

\[
\begin{align*}
\text{min} & \quad \sum_{e \in E} d_e x_e \\
\text{s.t.} & \quad \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
& \quad \text{nosubtour}(x) \\
& \quad x_e \in \{0, 1\} \quad \forall e \in E
\end{align*}
\]

(CIP formulation of TSP)

Single nosubtour constraint rules out subtours (e.g. by domain propagation). It may also separate subtour elimination inequalities.
Mixed-Integer Nonlinear Programs (MINLPs)

\[ \min \ c^T x \]

s.t. \[ g_k(x) \leq 0 \quad \forall k \in [m] \]

\[ x_i \in \mathbb{Z} \quad \forall i \in \mathcal{I} \subseteq [n] \]

\[ x_i \in [\ell_i, u_i] \quad \forall i \in [n] \]

The functions \( g_k \in C^1([\ell, u], \mathbb{R}) \) can be \textit{convex} or \textit{nonconvex}.

![Convex Function](image1)

![Nonconvex Function](image2)
Support Vector Machine, e.g., with ramp loss.

\[
\begin{align*}
\text{min} & \quad \frac{1}{2} w^T w + \frac{C}{n} \sum_{i=1}^{n} (\xi_i + 2(1 - z_i)) \\
\text{s.t.} & \quad z_i (y_i (w^T x_i + b) - 1 + \xi_i) \geq 0 \quad \forall i \\
& \quad \xi_i \in [0, 2], \quad z_i \in \{0, 1\} \quad \forall i \\
& \quad w \in \mathbb{R}^d, \quad b \in \mathbb{R}
\end{align*}
\]
• Mixed Integer Programs

- MIP

- Constraint Integer Programming

- Constraint Programming

- Constraint Integer Programming

- SAT

- Pseudo-Boolean Optimization

- MINLP

- CP

- CIP

- MIP ⊊ CIP

- Every CP over a finite domain space is a CIP.

- Every MIP is a CIP.
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems

Relation to CP and MIP
- Every MIP is a CIP.
- Every CP over a finite domain space is a CIP.
- **Mixed Integer Programs**
- **SAT** satisfiability problems
- **Pseudo-Boolean Optimization**

Relation to CP and MIP
- Every MIP is a CIP.
- "MIP $\subseteq$ CIP"
- Every CP over a finite domain space is a CIP.
- "FD $\subseteq$ CIP"
• Mixed Integer Programs
• SATisfiability problems
• Pseudo-Boolean Optimization
• Mixed Integer Nonlinear Programs
Mixed Integer Programs

SAT isfiability problems

Pseudo-Boolean Optimization

Mixed Integer Nonlinear Programs

Constraint Programming

Every MIP is a CIP.

"MIP ⊊ CIP"

Every CP over a finite domain space is a CIP.

"FD ⊊ CIP"
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Mixed Integer Nonlinear Programs
- Constraint Programming
- Constraint Integer Programming

Relation to CP and MIP:
- Every MIP is a CIP.
- Every CP over a finite domain space is a CIP.
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Mixed Integer Nonlinear Programs
- Constraint Programming
- Constraint Integer Programming

Relation to CP and MIP

- Every MIP is a CIP. "MIP ⊆ CIP"
- Every CP over a finite domain space is a CIP. "FD ⊆ CIP"
SCIP – Solving Constraint Integer Programs

The SCIP Optimization Suite

(Mixed-) Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

http://scip.zib.de
Branch-and-bound

\[
\begin{align*}
\text{min } & \quad c^T x \\
\text{s.t. } & \quad Ax \geq b \\
& \quad x \geq 0 \\
& \quad x_i \in \mathbb{Z} \quad \forall i \in I \subseteq \{1, \ldots, n\}
\end{align*}
\]

(Mixed Integer Program)
Basic Workflow

read .. / check / instances / MIP / bell5 . mps
optimize
write solution mysolution . sol
quit

Displaying information
Use the display ... command to enter the menu and

- obtain solution information
- print the current transproblem to the console
- display plugin information, e.g., list all available branching rules

Changing Settings
Use the set ... command to list the settings menu.
Important Parameters

Numerical parameters
These must be set before reading a problem.

- `numerics/feastol`, default $10^{-6}$
- `numerics/epsilon`, default $10^{-9}$
- `numerics/infinity`, default $10^{20}$

Limits

- `limits/time`
- `limits/nodes`
- `limits/gap`
- ...

Randomization

- `randomization/randomseedshift`
- `randomization/lpseed`
- `randomization/permutationseed`
Operational Stages

- **Init**: Basic data structures are allocated and initialized.
- **Problem**: User includes required plugins (or just takes default plugins).
- **Transforming**
- **Presolving**
- **Init Solve**: Solving
- **Free Transform**
- **Free Solve**
Basic data structures are allocated and initialized.
User includes required plugins (or just takes default plugins).
- User creates and modifies the original problem instance.
- Problem creation is usually done in file readers.
SCIP_VAR* var;
SCIP_CALL(  // return value macro
    SCIPcreateVar(
        scip,   // SCIP pointer
        &var,   // save in variable
        "x_1",  // pass variable name
        0.0,    // lower bound
        1.0,    // upper bound
        100.0,  // obj. value
        SCIP_VARTYPE_BINARY,  // type
        TRUE,   // initial
        FALSE,  // removable
        NULL, NULL, NULL, // no callback functions
        NULL    // no variable data
    )
);
SCIP_CALL( SCIPaddVar(scip, var) );  // add var.
Diet Problem Constraint

SCIP_CALL( SCIPcreateConsLinear(
    scip,  // SCIP pointer
    &cons, // save in cons
    "energy", // name
    6, // number of variables
    vars, // array of variables
    vals, // array of values (coefficients)
    2000, // left hand side
    SCIPinfinity(scip), // no right hand side (use for equation)
    TRUE, // initial?
    FALSE, // separate?
    TRUE, // enforce?
    TRUE, // check?
    TRUE, // propagate?
    FALSE, // local?
    FALSE, // modifiable?
    FALSE, // dynamic?
    FALSE, // removable?
    FALSE // stick at node?
));

SCIP_CALL( SCIPaddCons(scip, cons) ); // add constraint
SCIP_CALL( SCIPreleaseCons(scip, &cons) ); // free cons. space

MIPs are specified using linear constraints only (may be “upgraded”).
Transforming Free Transform

- Creates a working copy of the original problem.
• data is copied into separate memory area
• presolving and solving operate on transformed problem
• original data can only be modified in problem modification stage
Presolving

**Task**
- reduce size of model by removing irrelevant information
- strengthen LP relaxation by exploiting integrality information
- make the LP relaxation numerically more stable
- extract useful information

**Primal Reductions:**
- based on feasibility reasoning
- no feasible solution is cut off

**Dual Reductions:**
- consider objective function
- at least one optimal solution remains
Use display presolvers to list all presolvers of SCIP.

**Disable Presolving**
Disable all presolving for a model

```
set presolving emphasis off
```

Deactivate single techniques

```
set presolving tworowbnd maxrounds 0
set propagating probing maxprerounds 0
set constraints components advanced maxprerounds 0
```

**Aggressive Presolving**

```
set presolving emphasis aggressive
```

**General Rule of Thumb**
Only deactivate single presolving techniques if you encounter performance problems.
Solving

- Init
- Problem
- Transforming
- Presolving
- Solving
- Free Transform
- Free Solve

Init Solve
Flow Chart SCIP

Presolving

Stop

Node selection

Conflict analysis

Primal heuristics

Processing

Branching

Solve LP

Domain propagation

Pricing

Cuts

Enforce constraints
Node Selection

Task
- improve primal bound
- keep comp. effort small
- improve global dual bound

Techniques
- basic rules
  - depth first search (DFS) → early feasible solutions
  - best bound search (BBS) → improve dual bound
  - best estimate search (BES) → improve primal bound
- combinations
  - BBS or BES with plunging
  - hybrid BES/BBS
  - interleaved BES/BBS
Available Node Selectors

```
display nodeselectors
```

<table>
<thead>
<tr>
<th>node selector</th>
<th>std priority</th>
<th>memsave priority</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate</td>
<td>200000</td>
<td>100</td>
<td>best estimate search</td>
</tr>
<tr>
<td>bfs</td>
<td>100000</td>
<td>0</td>
<td>best first search</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dfs</td>
<td>0</td>
<td>100000</td>
<td>depth first search</td>
</tr>
</tbody>
</table>

Switching Node Selectors

Only the node selector with highest standard priority is active. Use

```
set nodeselection dfs stdprio 1000000
```

to activate depth first search also in non-memsave mode.
Domain Propagation

Task

- simplify model locally
- improve local dual bound
- detect infeasibility

Techniques

- constraint specific
  - each cons handler may provide a propagation routine
  - reduced presolving (usually)
- dual propagation
  - root reduced cost strengthening
  - objective function
- special structures
  - variable bounds
Flow Chart SCIP

Presolving

Stop

Node selection

Conflict analysis

Processing

Primal heuristics

Branching

Solve LP

Domain propagation

Pricing

Cuts

Enforce constraints
LP Solving

- LP solver is a black box
- interface to different LP solvers: SoPlex, CPLEX, XPress, Gurobi, CLP, ...
- primal/dual simplex
- barrier with/without crossover
- feasibility double-checked by SCIP
- condition number check
- resolution by changing parameters: scaling, tolerances, solving from scratch, other simplex
Most Important LP Parameters

- `lp/initalgorithm`, `lp/resolvealgorithm`
  - Primal/Dual Simplex Algorithm
  - Barrier w and w/o crossover
- `lp/pricing`
  - normally LP solver specific default
  - Devex
  - Steepest edge
  - Quick start steepest edge
- `lp/threads`

Slow LP performance is a blocker for the solving process and can sometimes be manually tuned significantly.
Flow Chart SCIP

Presolving

Stop

Node selection

Conflict analysis

Primal heuristics

Processing

Branching

Solve LP

Domain propagation

Pricing

Cuts

Enforce constraints
Cutting Plane Separation

Task
- strengthen relaxation
- add valid constraints
- generate on demand

Techniques
- general cuts
  - complemented MIR cuts
  - Gomory mixed integer cuts
  - strong Chvátal-Gomory cuts
  - implied bound cuts
  - reduced cost strengthening
- problem specific cuts
  - 0-1 knapsack problem
  - stable set problem
  - 0-1 single node flow problem
Cuts for the 0-1 Knapsack Problem

Feasible region: \((b \in \mathbb{Z}_+, a_j \in \mathbb{Z}_+ \ \forall \ j \in N)\)

\[
X^{BK} := \{ \ x \in \{0,1\}^n : \sum_{j \in N} a_j x_j \leq b \ \}
\]

Minimal Cover: \(C \subseteq N\)

- \(\sum_{j \in C} a_j > b\)
- \(\sum_{j \in C \setminus \{i\}} a_j \leq b \ \forall \ i \in C\)

Minimal Cover Inequality

\[
\sum_{j \in C} x_j \leq |C| - 1
\]

Minimal cover:

\(C = \{2,3,4\}\)

Minimal cover inequality:

\(x_2 + x_3 + x_4 \leq 2\)
Separation Tips and Parameters

Disable/Speed up/Emphasize All Separation

```
set separating emphasis off/fast/aggressive
```

Disable Single Separation Techniques

```
set separating clique freq −1
set constraints cardinality sepafreq −1
```

Some Important Parameters

- `separating/maxcuts, separating/maxcutsroot`
- `separating/maxrounds, separating/maxroundsroot`
- `separating/maxstallrounds, separating/maxstallroundsroot`
LP solution may violate a constraint not contained in the relaxation.

Enforcing is necessary for a correct implementation!

**Constraint handler** resolves the infeasibility by ...

- Reducing a variable’s domain,
- Separating a cutting plane (may use integrality),
- Adding a (local) constraint,
- Creating a branching,
- Concluding that the subproblem is infeasible and can be cut off, or
- Just saying “solution infeasible”.

Constraint Enforcement

- Presolving
  - Stop
  - Node selection
    - Conflict analysis
      - Primal heuristics
    - Processing
      - Branching
        - Enforce constraints
          - Solve LP
            - Pricing
              - Cuts
                - Domains propagation
  - Enforce constraints
    - Cutoff
      - Infeasible
    - Feasible
      - Branched
      - Added cut
      - Reduced domain
      - Added constraint
Constraint Enforcement

- Presolving
  - Stop
  - Node selection
    - Conflict analysis
    - Primal heuristics
    - Branching
  - Processing
    - Enforce constraints
    - Domain propagation
      - Solve LP
        - Pricing
        - Cuts
        - Enforce constraints
          - Reduced domain
            - Added cut
            - Cutoff
            - Infeasible
            - Feasible
          - Added constraint
            - Branched
Constraint Enforcement

- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible
Constraint Enforcement

**Presolving**
- Stop

**Node selection**

- Conflict analysis
- Primal heuristics
- Branching

**Processing**

**Presolving**

**Solve LP**
- Domain propagation
- Pricing
- Cuts

**Enforce constraints**

- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Gregor Hendel – Using SCIP
Constraint Enforcement

- Presolving
  - Stop
  - Node selection
    - Conflict analysis
    - Processing
      - Primal heuristics
      - Branching
      - Enforce constraints

- Solve LP
  - Domain propagation
  - Pricing
  - Cuts
  - Enforce constraints

- Reduced domain
- Added cut
- Cutoff
- Infeasible
- Added constraint
- Branched
- Feasible
### Constraint Enforcement

- **Presolving**
- **Stop**
- **Node selection**
- **Processing**
- **Branching**
- **Conflict analysis**
- **Solve LP**
- **Domain propagation**
- **Pricing**
- **Cuts**
- **Enforce constraints**

- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Gregor Hendel — Using SCIP
Constraint Enforcement

- Presolving
- Node selection
- Processing
- Branching
- Conflict analysis
- Primal heuristics
- Solve LP
- Domain propagation
- Pricing
- Cuts
- Enforce constraints
- Stop
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible
Constraint Enforcement

- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
  - Feasible

Gregor Hendel – Using SCIP
Flow Chart SCIP

- **Presolving**
  - Stop
  - Node selection
    - Conflict analysis
      - Primal heuristics
    - Processing
      - Branching
      - Solve LP
        - Domain propagation
        - Pricing
        - Cuts
        - Enforce constraints
  - Branching
Branching Rules

Task

- divide into (disjoint) subproblems
- improve local bounds

Techniques

- **branching on variables**
  - most infeasible
  - least infeasible
  - random branching
  - strong branching
  - pseudocost
  - reliability
  - VSIDS
  - hybrid reliability/inference

- **branching on constraints**
  - SOS1
  - SOS2
Branching Rule Tips and Parameters

**Branching Rule Selection**
Branching rules are applied in decreasing order of priority.

```
SCIP> display branching
```

<table>
<thead>
<tr>
<th>Branching Rule</th>
<th>Priority</th>
<th>Maxdepth</th>
<th>MaxBDDDist</th>
</tr>
</thead>
<tbody>
<tr>
<td>relpscost</td>
<td>10000</td>
<td>-1</td>
<td>100.0%</td>
</tr>
<tr>
<td>pscost</td>
<td>2000</td>
<td>-1</td>
<td>100.0%</td>
</tr>
<tr>
<td>inference</td>
<td>1000</td>
<td>-1</td>
<td>100.0%</td>
</tr>
<tr>
<td>mostinf</td>
<td>100</td>
<td>-1</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

**Reliability Branching Parameters**
All parameters prefixed with `branching/relpscost/`

- `sbiterquot`, `sbiterofs` to increase the budget for strong branching
- `minreliable (= 1), maxreliable (= 5)` to increase threshold to consider pseudo costs as reliable
Flow Chart SCIP

- **Presolving**
  - Node selection
    - Conflict analysis
      - Primal heuristics
    - Branching
  - Processing
  - Stop

- **Solve LP**
  - Pricing
  - Cuts
  - Enforce constraints
  - Domain propagation
Primal Heuristics

Task
- improve primal bound
- effective on average
- guide remaining search

Techniques
- structure-based
  - clique
  - variable bounds
- rounding
  - possibly solve final LP
- diving
  - least infeasible
  - guided
- objective diving
  - objective feasibility pump
- Large Neighborhood Search
  - RINS, local branching
  - RENS
  - Adaptive LNS
  - Completion of partial solutions
Primal Heuristics Tips and Parameters

Disable/Speed Up/Emphasize Heuristics

\texttt{set heuristics emphasis off/fast/aggressive}

Disable an individual heuristic via

\texttt{set heuristics feaspump freq \textasciitilde 1}

Important Parameters

- \texttt{heuristics/alns/nodesofs, heuristics/alns/nodesquot} to increase the computational budget of this LNS technique
- \texttt{heuristics/guideddiving/... lpsolvefreq, maxlpiterofs maxlpiterquot} to control the LP solving during this diving technique

Advice

Use emphasis settings. \textbf{Do not} attempt to individually tune heuristics by hand.
Task

- Analyze infeasibility
- Derive valid constraints
- Help to prune other nodes

Techniques

- Analyze:
  - Propagation conflicts
  - Infeasible LPs
  - Bound-exceeding LPs
  - Strong branching conflicts

- Detection:
  - Cut in conflict graph
  - LP: Dual ray heuristic

- Use conflicts:
  - Only for propagation
  - As cutting planes
Operational Stages

- Init
- Problem
- Transforming
- Presolving
- Free Transform
- Free Solve
- Solving
- Init Solve
SCIP – Solving Constraint Integer Programs

The SCIP Optimization Suite

(Mixed-) Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

http://scip.zib.de
Different plugin classes are responsible of the following tasks.

1. Presolving and node propagation
   - Constraint handlers
   - Presolvers
   - Propagators

2. Separation
   - Constraint handlers
   - Separators

3. Improving solutions
   - Primal heuristics

4. Branching
   - Constraint handlers
   - Branching rules

5. Node selection
   - Node selectors
Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

⇒ plugins are black boxes for the SCIP core

⇒ Very flexible branch-and-bound based search algorithm
Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugins

- external callback objects
- interact with the framework through a very detailed interface
Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugins

- external callback objects
- interact with the framework through a very detailed interface
- SCIP knows for each plugin type:
  - the number of available plugins
  - priority defining the calling order (usually)
- SCIP does not know any structure behind a plugin

⇒ plugins are black boxes for the SCIP core
Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugins

- external callback objects
- interact with the framework through a very detailed interface
- SCIP knows for each plugin type:
  - the number of available plugins
  - priority defining the calling order (usually)
- SCIP does not know any structure behind a plugin
  ⇒ plugins are black boxes for the SCIP core

⇒ Very flexible branch-and-bound based search algorithm
Types of Plugins

- Constraint handler: assures feasibility, strengthens formulation
- Separator: adds cuts, improves dual bound
- Pricer: allows dynamic generation of variables
- Heuristic: searches solutions, improves primal bound
- Branching rule: how to divide the problem?
- Node selection: which subproblem should be regarded next?
- Presolver: simplifies the problem in advance, strengthens structure
- Propagator: simplifies problem, improves dual bound locally
- Reader: reads problems from different formats
- Event handler: catches events (e.g., bound changes, new solutions)
- Display: allows modification of output
- ...
A closer look: branching rules
A closer look: branching rules

Branch

- allfull
- strong
- full
- strong
- inference
- leastinf
- mostinf
- relps
- cost
- random
- pscost
- Branch

Gregor Hendel – Using SCIP
What does SCIP know about branching rules?

- SCIP knows the number of available branching rules
- each branching rule has a priority
- SCIP calls the branching rule in decreasing order of priority
- the interface defines the possible results of a call:
  - branched
  - reduced domains
  - added constraints
  - detected cutoff
  - did not run
How does SCIP call a branching rule?

```c
/* start timing */
SCIPclockStart(branchrule->branchclock, set);

/* call external method */
SCIP_CALL( branchrule->branchexeclp(set->scip, branchrule,
    allowaddcons, result) );

/* stop timing */
SCIPclockStop(branchrule->branchclock, set);

/* evaluate result */
if( *result != SCIP_CUTOFF
    && *result != SCIP_CONSADDED
    && *result != SCIP_REDUCEDDOM
    && *result != SCIP_SEPARATED
    && *result != SCIP_BRANCHED
    && *result != SCIP_DIDNOTRUN )
{
    SCIPerrorMessage(
        "branching rule <%s> returned invalid result code <%d> from LP \n solution branching\n", branchrule->name, *result);
    return SCIP_INVALIDRESULT;
}
```
What can a plugin access?

Plugins are allowed to access all global (core) information

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...
What can a plugin access?

Plugins are allowed to access all global (core) information

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Ideally, plugins should not access data of other plugins!!!
What can a plugin access?

Plugins are allowed to access all global (core) information

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Ideally, plugins should not access data of other plugins!!!

Branching Rules

- LP solution
- variables
- statistics
Constraint Handlers

Constraint handlers

- most powerful plugins in SCIP
- define the feasible region
- a single constraint may represent a whole set of inequalities

Functions

- check and enforce feasibility of solutions
- can add linear representation to LP relaxation
- constraint-specific presolving, domain propagation, separation

Result

- SCIP is constraint based
  - Advantage: flexibility
  - Disadvantage: limited global view
Extending SCIP: TSP
Extending SCIP: Coloring
Important SCIP topics not covered in this workshop:

- branch-and-price: SCIP can be extended to a problem-specific branch-cut-and-price solver
  - see The Bin Packing Example in C
  - see also GCG
- allows for Benders decomposition since version 6.0, see Technical Report
- browse older technical reports for details on recently added cutting plane selection, primal heuristics, symmetry breaking, and much more
How to solve your MINLP optimization problem:

- write down the mathematical description
- modeling language, e.g., ZIMPL, generates input for MINLP solver
- SCIP can even read ZIMPL files directly

MIP and MINLP solving

- ... is a bag of tricks
- new tricks are introduced almost every day
- advantages of SCIP
  - open source, you can see everything that happens
  - hundreds of parameters to play with
  - broad scope
  - easily extendable