SCIP Introduction Sep. 26, 2018

Time	Schedule
9:30–11:00	Introduction & Overview (1)
11:00–11:30	Break
11:30–12:30	Introduction & Overview (2)
12:30–14:00	Lunch Break
14:00–15:30	Programming Exercises with PySCIPOpt (first steps)
15:30–16:00	Break
16:00–17:30	Programming Exercises With PySCIPOpt (more realistic examples)

SCIP Introduction at the 3rd IMI-ISM-ZIB Workshop, Tokio, Japan

Introduction to SCIP

Gregor Hendel, hendel@zib.de

SCIP Introduction Day
September 26, 2018
3rd IMI-ISM-ZIB MODAL Workshop

What is SCIP?

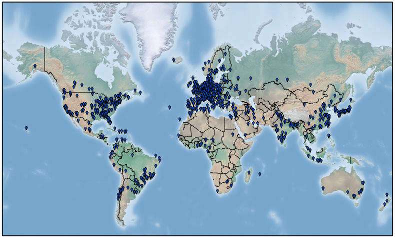
SCIP (Solving Constraint Integer Programs) . . .

- 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

Outline

SCIP – Solving Constraint Integer Programs

http://scip.zib.de

SCIP – Solving Constraint Integer Programs

Constraint Integer Programming

The Solving Process of SCIP

Extending SCIP by Plugins

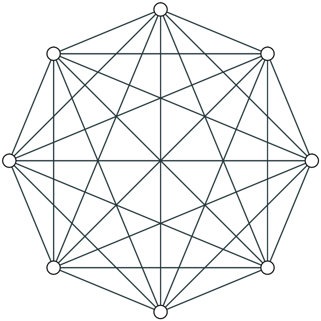
The SCIP Optimization Suite

http://scip.zib.de

An example: the Traveling Salesman Problem

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

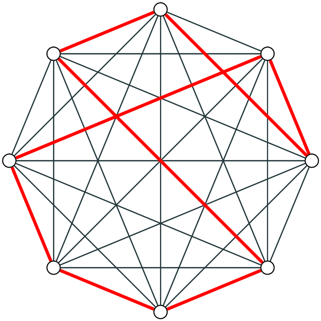
Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.



K_8

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.



K_8

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.

What is a Constraint Integer Program?

Mixed Integer Program

Objective function:

- linear function

Feasible set:

- described by linear constraints

Variable domains:

- real or integer values

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

Constraint Program

Objective function:

- arbitrary function

Feasible set:

- given by arbitrary constraints

Variable domains:

- arbitrary (usually finite)

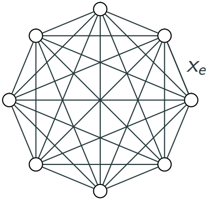
$$\begin{aligned} \min \quad & c(x) \\ s.t. \quad & x \in F \\ & (x_I, x_N) \in \mathbb{Z}^I \times X \end{aligned}$$

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



K_8

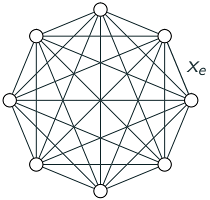
TSP – Integer Programming Formulation

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



K_8

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

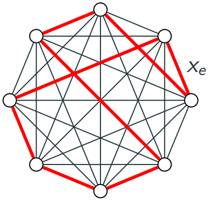
TSP – Integer Programming Formulation

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



K_8

$$
\begin{array}{llll}
\min & \sum_{e \in E} d_e \, x_e & & \\
\text{subject to} & \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V & \text{node degree} \\
& \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \emptyset & \\
& x_e \in \{0, 1\} & \forall e \in E &
\end{array}
$$

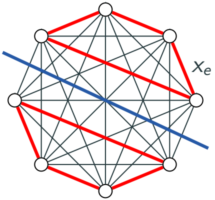
TSP – Integer Programming Formulation

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



K_8

$$
\begin{array}{lll}
\min & \sum_{e \in E} d_e \, x_e & \\
\text{subject to} & \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\
 & \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \varnothing \quad \text{subtour elimination} \\
 & x_e \in \{0, 1\} & \forall e \in E
\end{array}
$$

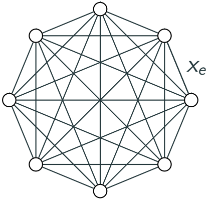
TSP – Integer Programming Formulation

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



K_8

$$\min \sum_{e \in E} d_e x_e \qquad \text{distance}$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 && \forall S \subset V, S \neq \emptyset \\
& x_e \in \{0, 1\} && \forall e \in E
\end{aligned}$$

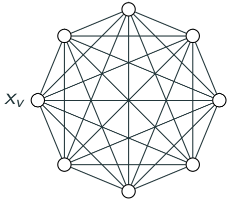
TSP – Constraint Programming Formulation

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



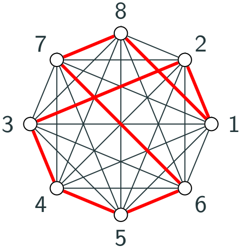
K_8

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



$$
\begin{aligned}
\min \quad & \text{length}(x_1, \ldots, x_n) \\
\text{subject to} \quad & \text{alldifferent}(x_1, \ldots, x_n) \\
& x_v \in \{1, \ldots, n\} \qquad \forall v \in V
\end{aligned}
$$

Constraint Integer Program

Objective function:

- ▷ linear function

Feasible set:

- ▷ described by arbitrary constraints

Variable domains:

- ▷ real or integer values

$$\begin{array}{ll} \min & c^T x \\ s.t. & x \in F \\ & (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C \end{array}$$

Remark:

- arbitrary objective or variables modeled by constraints

Constraint Integer Program

Objective function:

▷ linear function

Feasible set:

▷ described by arbitrary constraints

Variable domains:

▷ real or integer values

$$\begin{array}{llll} \min & \sum_{e \in E} d_e\, x_e & & \\ s.t. & \sum_{e \in \delta(v)} x_e & = 2 & \forall\, v \in V \\ & \text{nosubtour}(x) & & \\ & x_e & \in \{0,1\} & \forall\, e \in E \end{array}$$

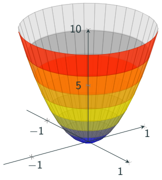
(CIP formulation of TSP)

Single nosubtour constraint rules out subtours (e.g. by domain propagation). It may also separate subtour elimination inequalities.

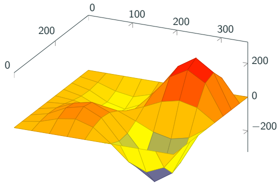
Mixed-Integer Nonlinear Programs (MINLPs)

$$\begin{aligned} \min\ & c^{\mathsf{T}} x & \\ \text{s.t. } & g_k(x) \leq 0 & \forall k \in [m] \\ & x_i \in \mathbb{Z} & \forall i \in \mathcal{I} \subseteq [n] \\ & x_i \in [\ell_i, u_i] & \forall i \in [n] \end{aligned}$$

The functions $g_k \in C^1([\ell, u], \mathbb{R})$ can be



convex or



nonconvex

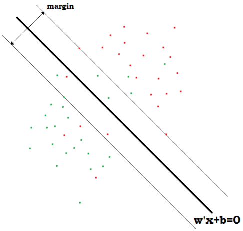
Support Vector Machine, e.g., with ramp loss.

$$\min \frac{w^{\mathsf{T}}w}{2} + \frac{C}{n}\sum_{i=1}^{n}(\xi_i + 2(1 - z_i))$$

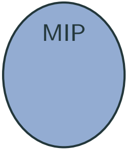
$$\text{s.t. } z_i(y_i(w^{\mathsf{T}}x_i + b) - 1 + \xi_i) \geq 0 \qquad \forall i$$

$$\xi_i \in [0,2], \quad z_i \in \{0,1\} \qquad \forall i$$

$$w \in \mathbb{R}^d, \quad b \in \mathbb{R}$$

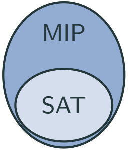


- Mixed Integer Programs



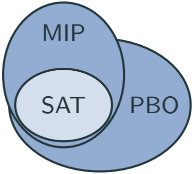
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems



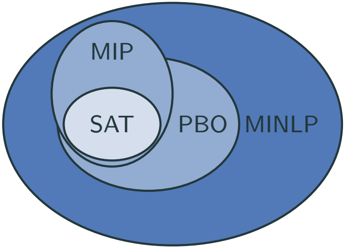
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization



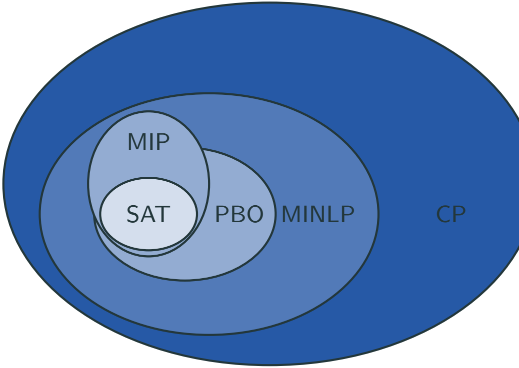
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Mixed Integer NonIinear Programs



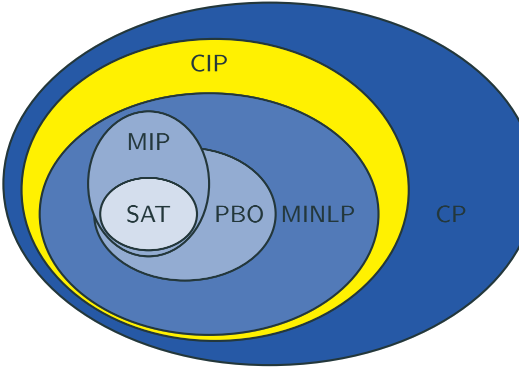
Constraint Integer Programming

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



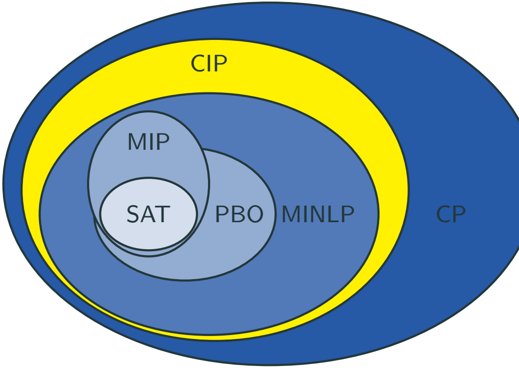
Constraint Integer Programming

- **M**ixed **I**nteger **P**rograms
- **SAT**isfiability problems
- **P**seudo-**B**oolean **O**ptimization
- **M**ixed **I**nteger **N**on**l**inear **P**rograms
- **C**onstraint **P**rogramming
- **C**onstraint **I**nteger **P**rogramming



Constraint Integer Programming

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



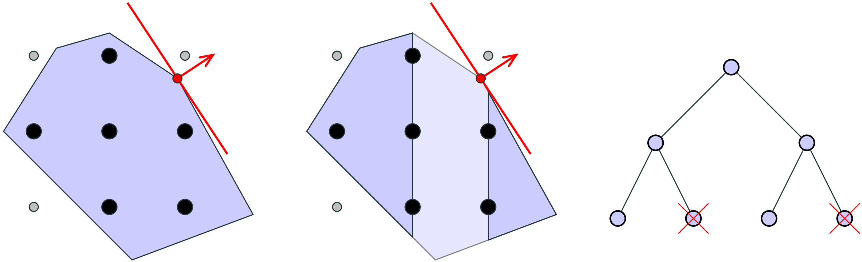
Relation to CP and MIP

- Every MIP is a CIP. *"MIP $\subsetneq$ CIP"*
- Every CP over a finite domain space is a CIP. *"FD $\subsetneq$ CIP"*

SCIP – Solving Constraint Integer Programs

http://scip.zib.de

Branch-and-bound



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

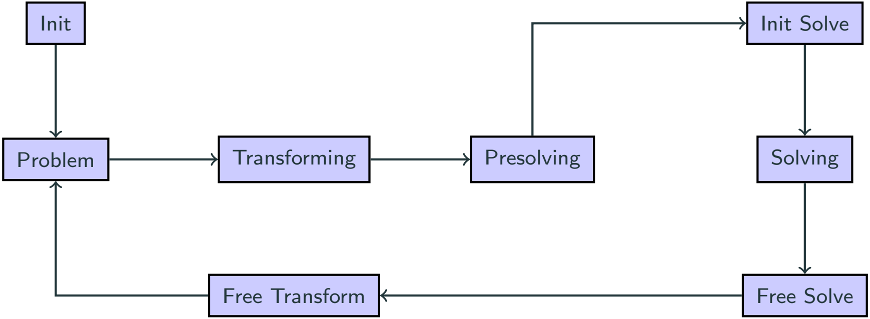
- `randomization/randomseedshift`
- `randomization/lpseed`
- `randomization/permutationseed`

Different Tasks – Different Plugins

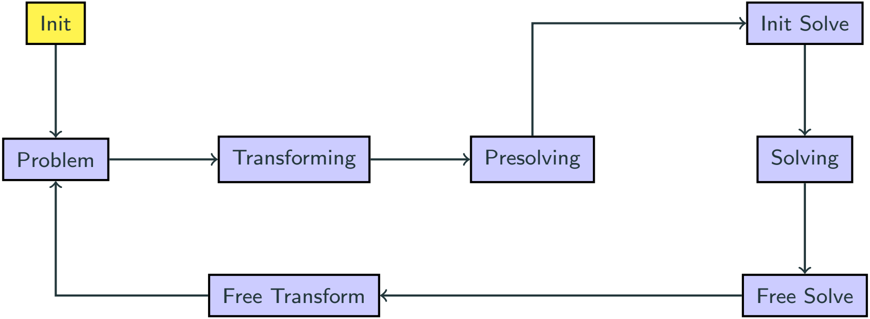
Different plugin classes are responsible of the following tasks.

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

Operational Stages

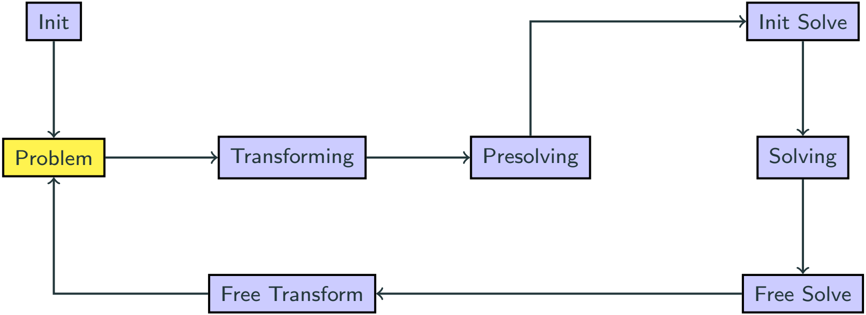


Operational Stages



- Basic data structures are allocated and initialized.
- User includes required plugins (or just takes default plugins).

Problem Specification



- User creates and modifies the original problem instance.
- Problem creation is usually done in file readers.

Define Variables (LOP Example)

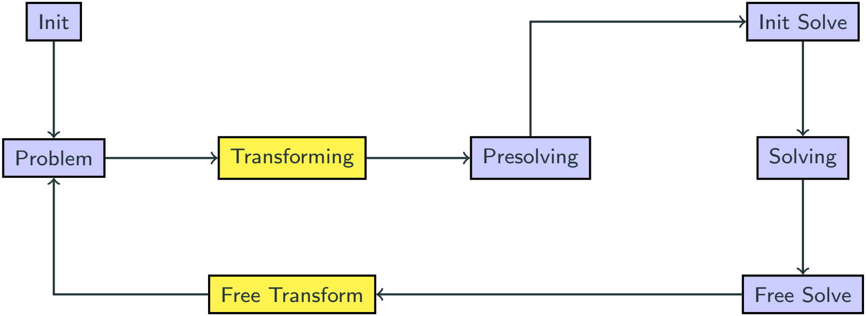
```
SCIP_VAR* var;
SCIP_CALL(                    // return value macro
  SCIPcreateVar(
    scip,                     // SCIP pointer
    &var,                     // save in variable
    "varname",                // pass variable name
    0.0,                      // lower bound
    1.0,                      // upper bound
    length,                   // 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.
```

TSP: Define Degree Constraints

```
SCIP_CALL( SCIPcreateConsLinear(
    scip,          // SCIP pointer
    &cons,         // save in cons
    "consname",    // name
    nvar,          // number of variables
    vars,          // array of variables
    vals,          // array of values
    2.0,           // left hand side
    2.0,           // right hand side (equation)
    TRUE,          // initial?
    FALSE,         // separate?
    TRUE,          // enforce?
    TRUE,          // check?
    TRUE,          // propagate?
    FALSE,         // local?
    FALSE,         // modifable?
    FALSE,         // dynamic?
    FALSE,         // removable?
    FALSE          // stick at node?
  ));
SCIP_CALL( SCIPaddCons(scip, cons) );      // add constraint
SCIP_CALL( SCIPreleaseCons(scip, &cons) ); // free cons. space
```

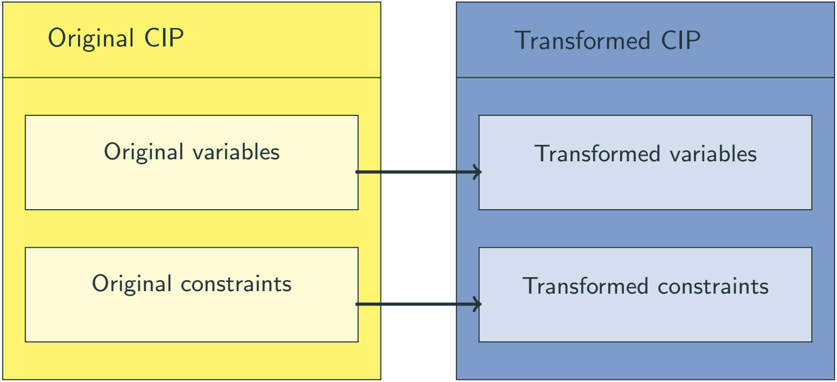
MIPs are specified using linear constraints only (may be "upgraded").

Transformation



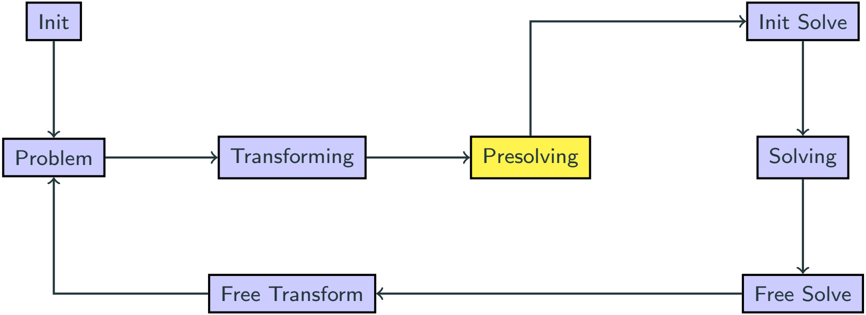
- Creates a working copy of the original problem.

Original and Transformed Problem

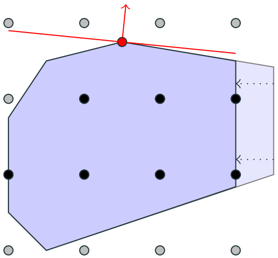


- data is copied into separate memory area
- presolving and solving operate on transformed problem
- original data can only be modified in problem modification stage

Presolving


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

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

Presolving Tips and Parameters

Use `display presolvers` to list all presolvers of SCIP.

Disable Presolving
Disable all presolving for a model

```
set presolving emphasis off
```

Deactivate single techniques

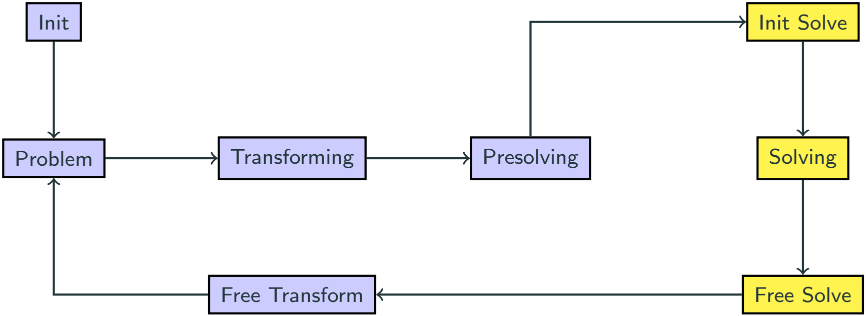
```
set presolving tworowbnd maxrounds 0
set propagating probing maxprerounds 0
set constraints components advanced maxprerounds 0
```

Aggressive Presolving

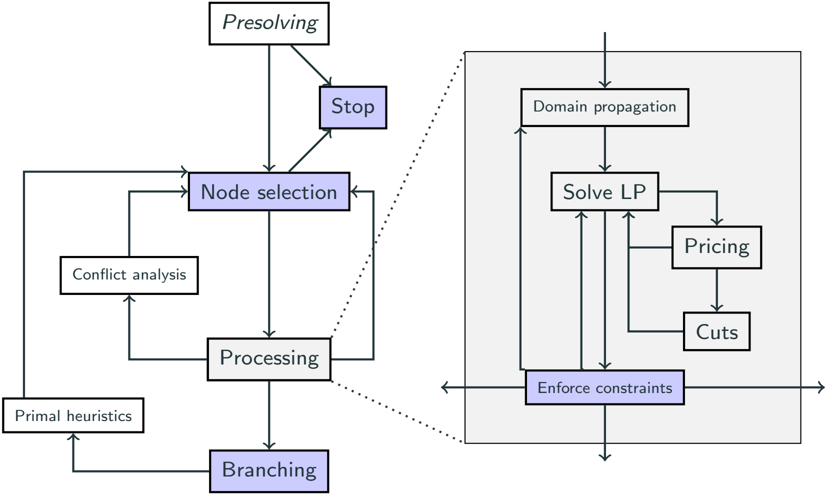
```
set presolving emphasis aggressive
```

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

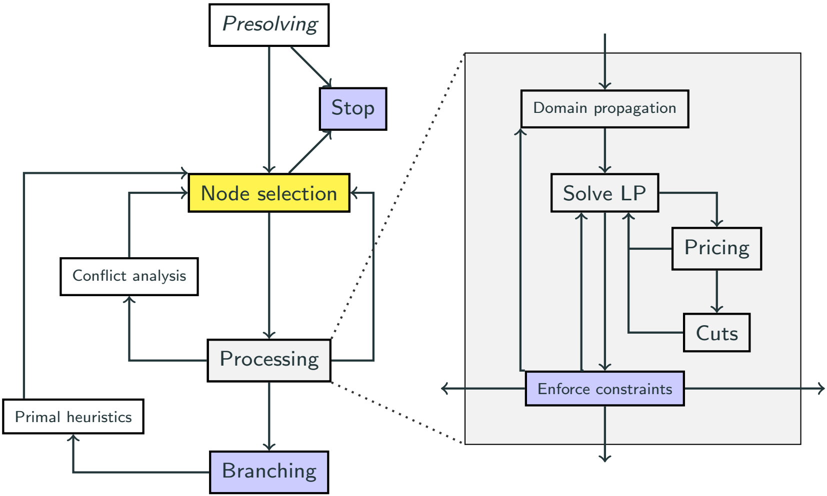
Solving

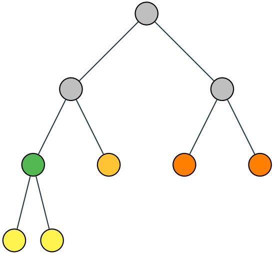

Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

Flow Chart SCIP



Flow Chart SCIP





Techniques

- basic rules
  - depth first search (DFS)
    $\rightarrow$ early feasible solutions
  - best bound search (BBS)
    $\rightarrow$ improve dual bound
  - best estimate search (BES)
    $\rightarrow$ improve primal bound

- combinations
  - BBS or BES with plunging
  - hybrid BES/BBS
  - interleaved BES/BBS

Task

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

Node Selection Tips and Parameters

Available Node Selectors

```
display nodeselectors

node selector std priority memsave prio  description
------------- ------------ ------------  -----------
estimate            200000          100  best estimate search
bfs                 100000            0  best first search
...
dfs                      0       100000  depth first search
```

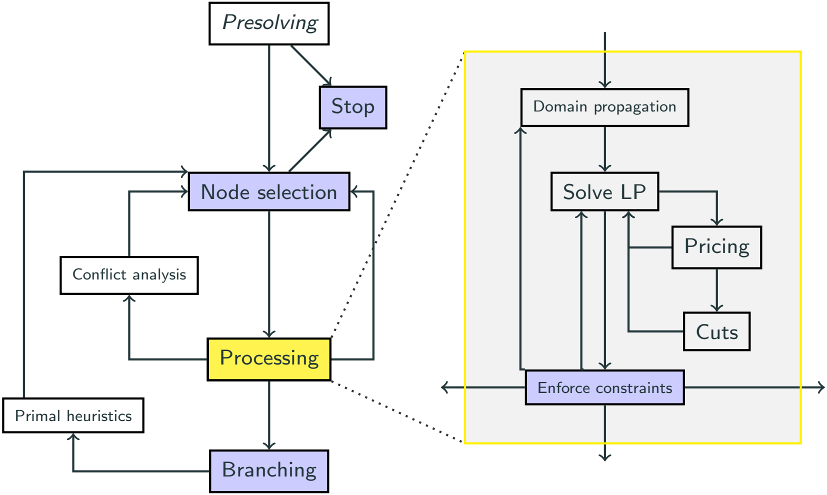
Switching Node Selectors

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

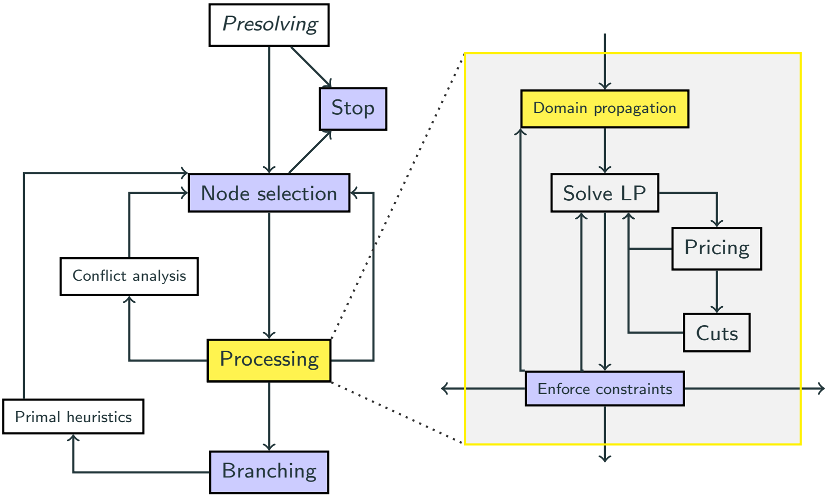
```
set nodeselection dfs stdpriority 1000000
```

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

Flow Chart SCIP



Flow Chart SCIP



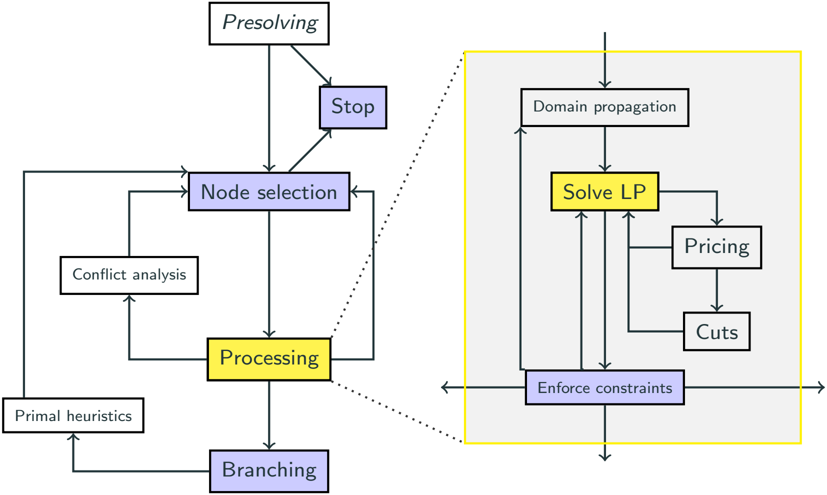
Domain Propagation

Task

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

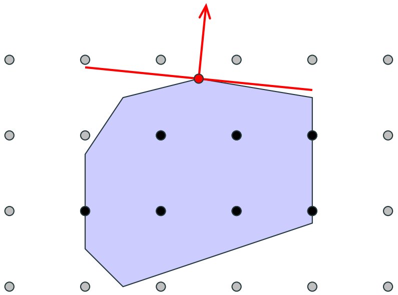
Techniques

- 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



Presolving
Stop
Node selection
Conflict analysis
Processing
Primal heuristics
Branching
Domain propagation
Solve LP
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

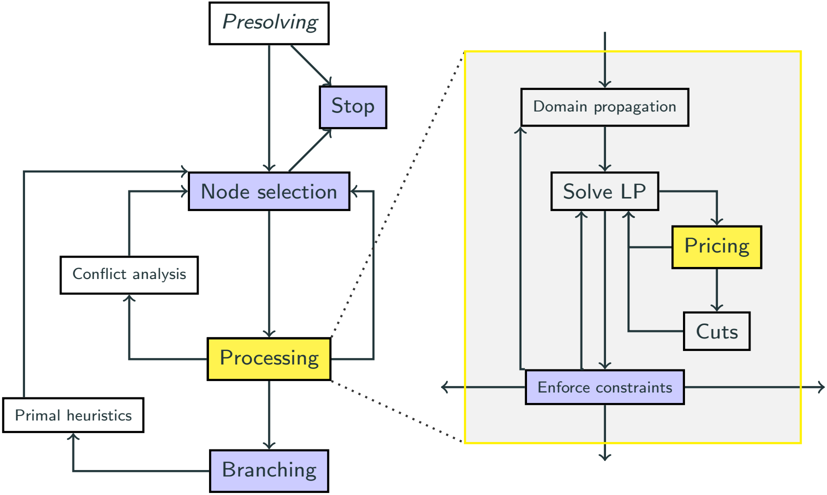
LP Solving Tips and Parameters

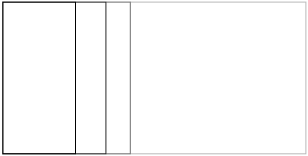
Most Important LP Parameters

- `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





Branch-and-Price

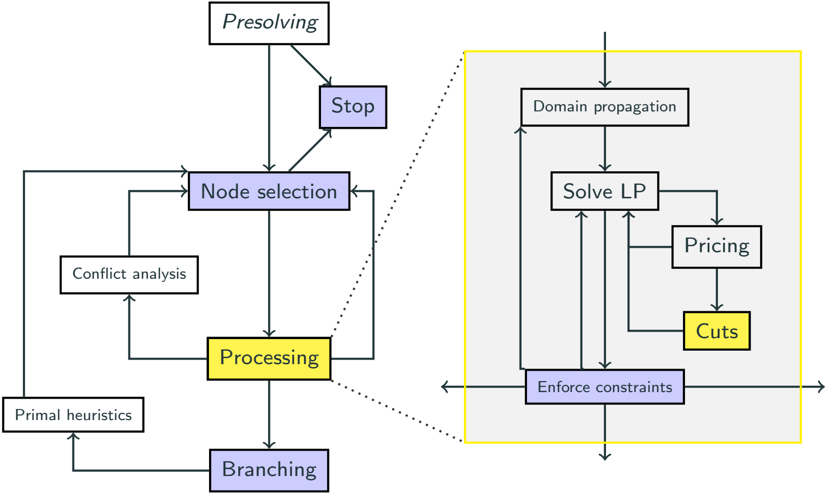
- huge number of variables
- start with subset
- add others, when needed

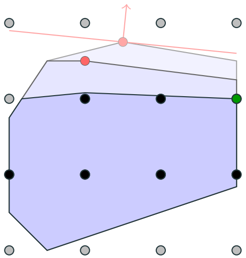
Pricing

- find variable with negative reduced costs
- or prove that there exists none
- typically problem specific

- dynamic aging of variables
- problem variable pricer to add them again
- early branching possible
- lazy variable bounds

Flow Chart SCIP





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

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

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

Minimal Cover: $C \subseteq N$

- $\sum_{j \in C} \quad a_j > b$
- $\sum_{j \in C \setminus \{i\}} a_j \leq b \quad \forall\, i \in C$

Minimal Cover Inequality

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

$5x_1 + 6x_2 + 2x_3 + 2x_4 \leq 8$

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
```

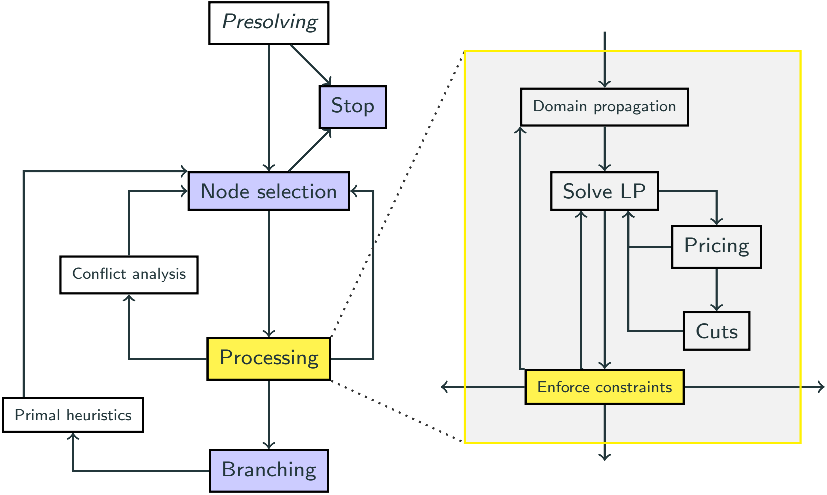
Disable Single Separation Techniques

```
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`

Flow Chart SCIP



Constraint Enforcement

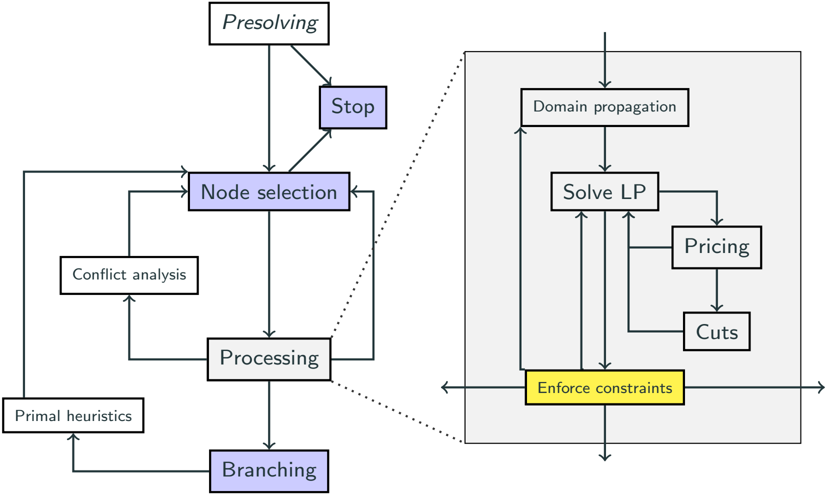
LP solution may violate a constraint not contained in the relaxation.

Enforcing is necessary for a correct implementation!

Constraint handler resolves the infeasibility by ...

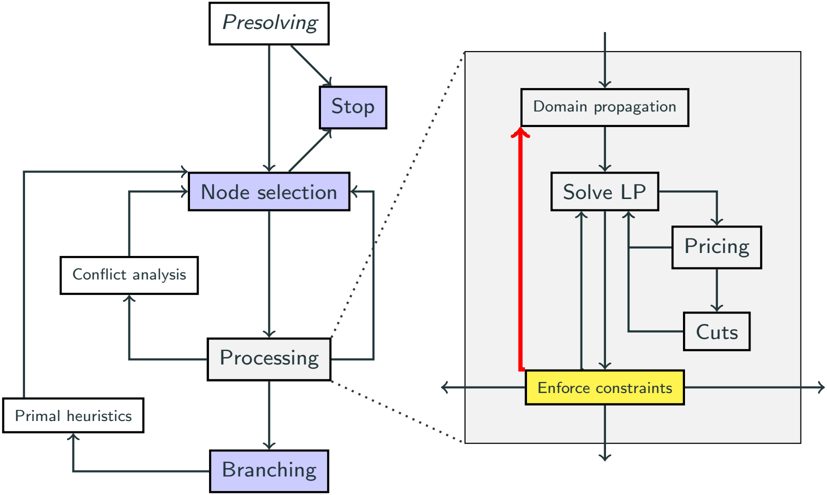
- Reducing a variable's domain,
- Separating a cutting plane (may use integrality),
- Adding a (local) constraint,
- Creating a branching,
- Concluding that the subproblem is infeasible and can be cut off, or
- Just saying "solution infeasible".

Constraint Enforcement



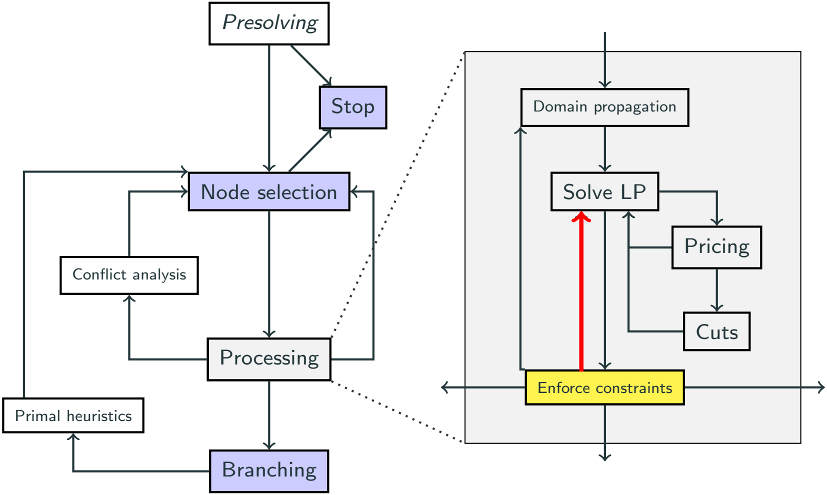
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement

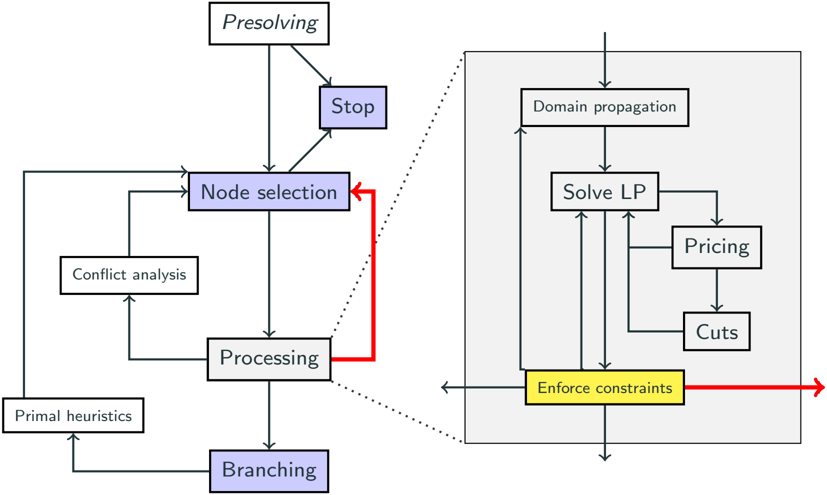


- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement

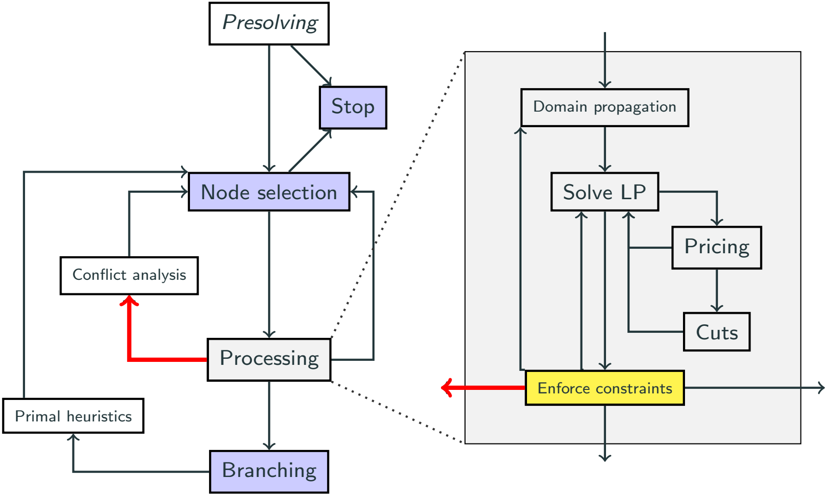


- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible



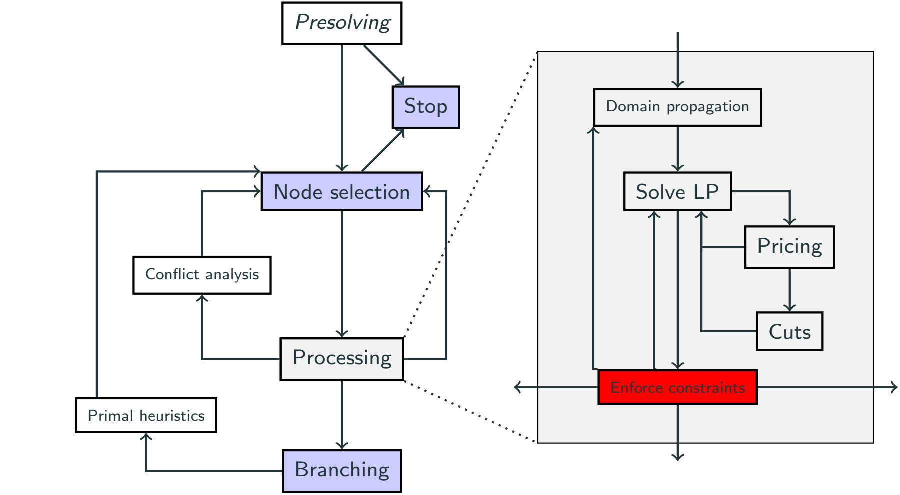
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



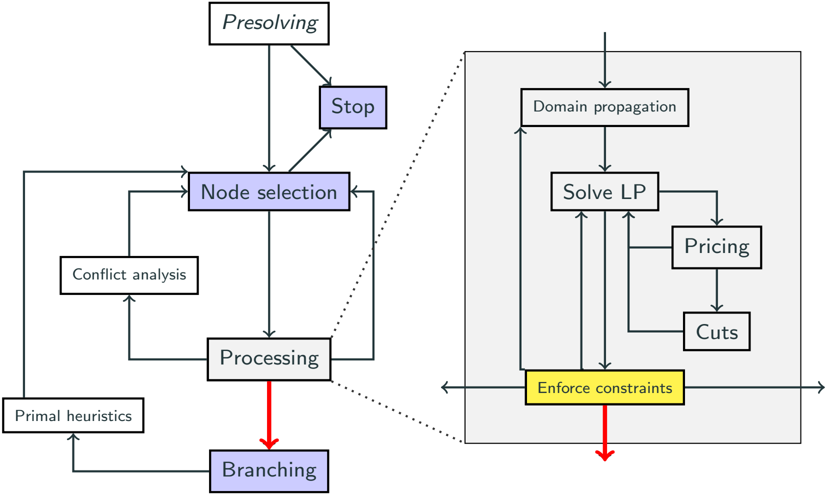
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



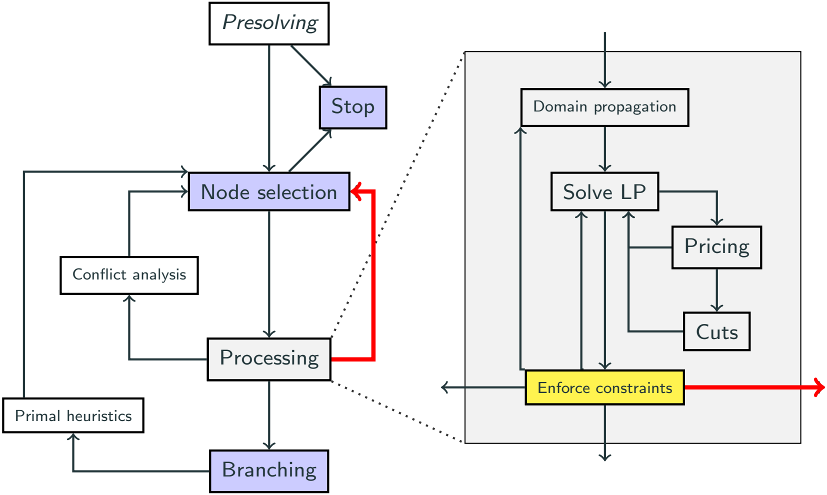
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



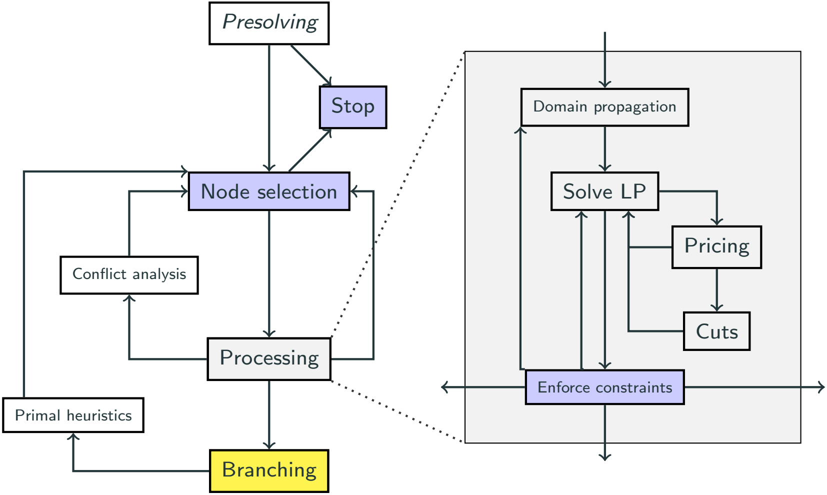
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement

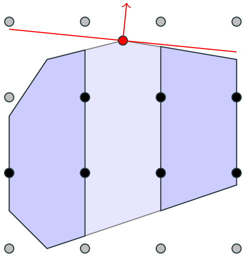


- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Flow Chart SCIP



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.

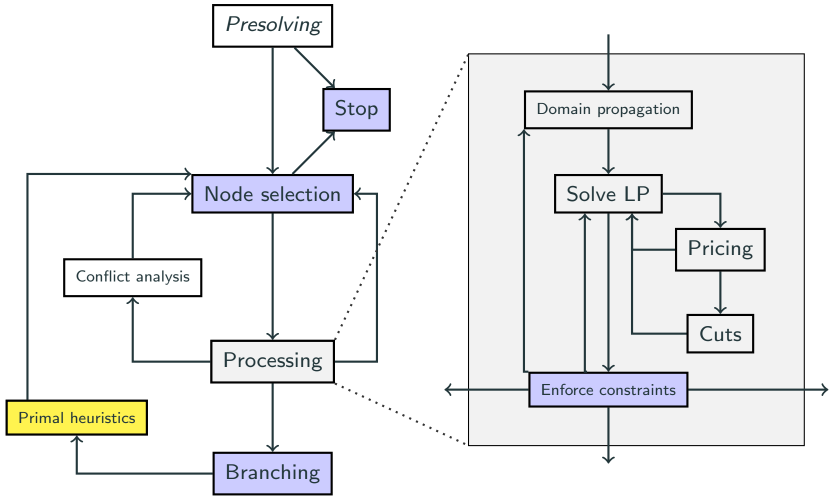
```
SCIP> display branching

 branching rule         priority  maxdepth  maxbddist
 ---------------        --------  --------  ---------
 relpscost                 10000        -1     100.0%
 pscost                     2000        -1     100.0%
 inference                  1000        -1     100.0%
 mostinf                     100        -1     100.0%
```

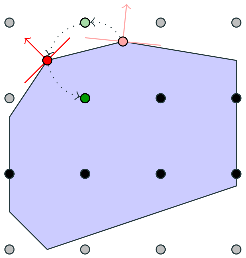
Reliability Branching Parameters
All parameters prefixed with `branching/relpscost/`

- `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



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

```
set heuristics emphasis off/fast/aggressive
```

Disable an individual heuristic via

```
set heuristics feaspump freq -1
```

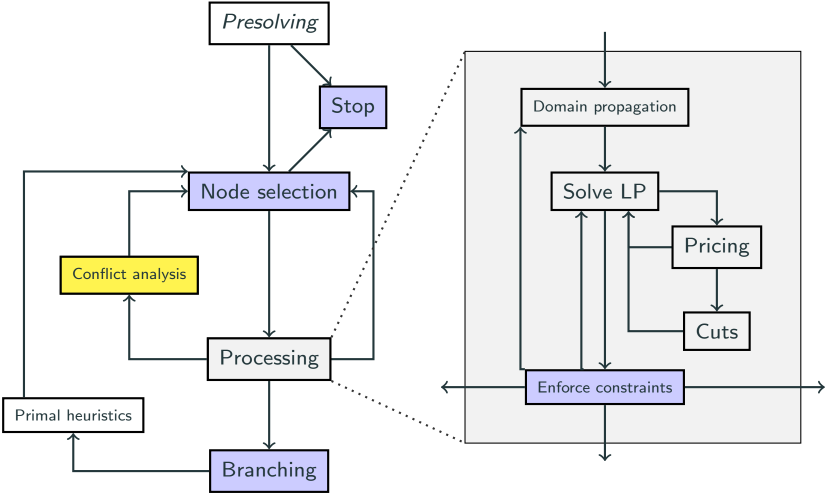
Important Parameters

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

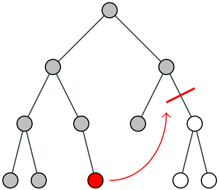
Advice

Use emphasis settings. Do not attempt to individually tune heuristics by hand.

Flow Chart SCIP



Conflict Analysis



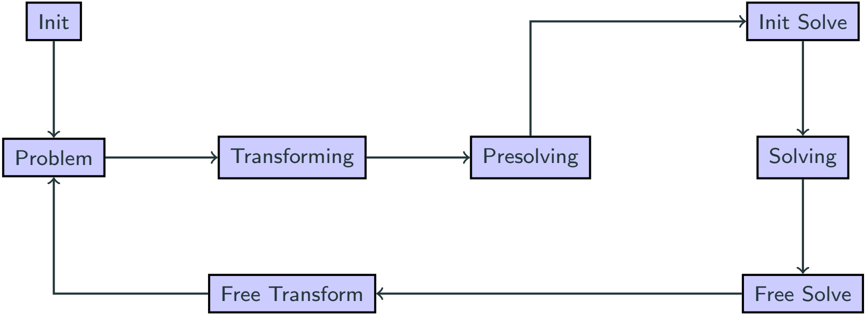
Task

- 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



SCIP – Solving Constraint Integer Programs

http://scip.zib.de

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

Interfaces to SCIP

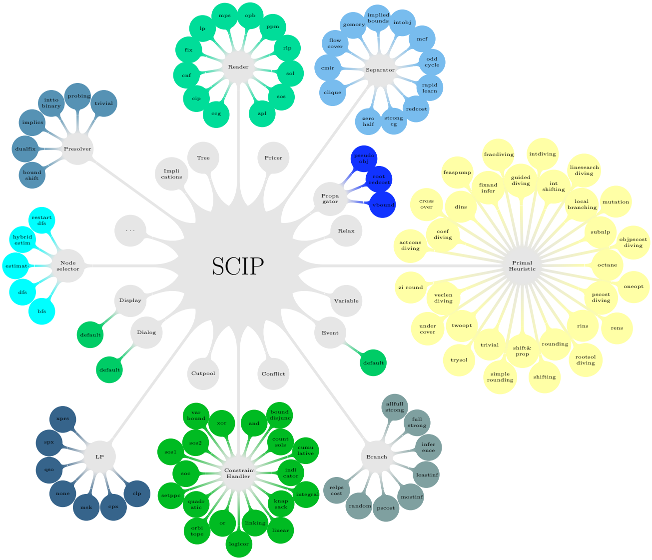
- interactive shell supports 10 different input formats
  → cip, cnf, flatzinc, rlp, lp, mps, opb, pip, wbo, zimpl
- C API/callable library
- C++ wrapper classes
- Python interface
- Java JNI interface
- AMPL
- GAMS
- Matlab (see also OPTI toolbox, `http://www.i2c2.aut.ac.nz/Wiki/OPTI/`)

Getting help

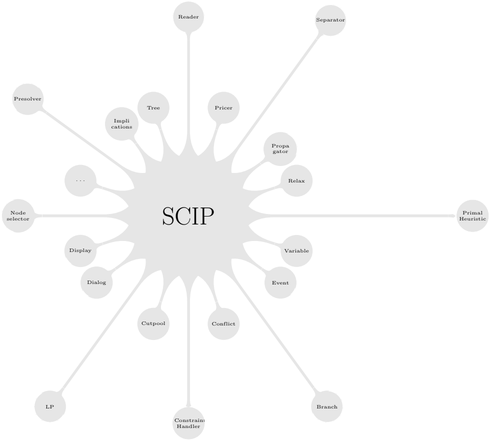
If you should ever get stuck, you can ...

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

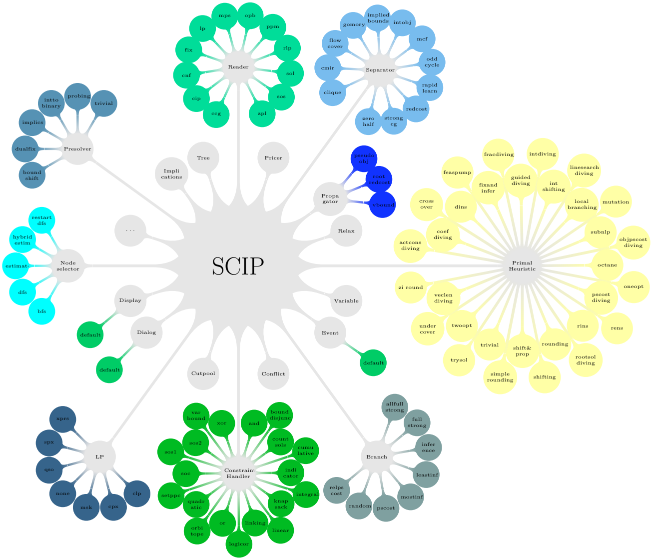
Structure of SCIP



Structure of SCIP



Structure of SCIP



Plugin based design

SCIP core

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

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

$\Rightarrow$ 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

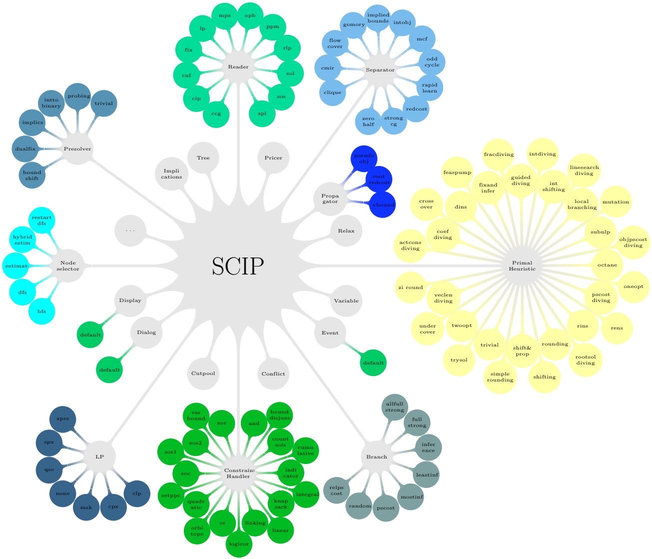
$\Rightarrow$ plugins are black boxes for the SCIP core

$\Rightarrow$ Very flexible branch-and-bound based search algorithm

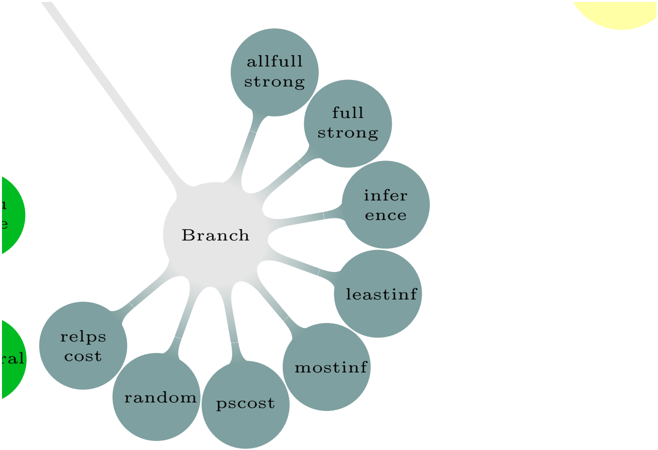
Types of Plugins

- 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



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?

```
/* 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␣\
␣␣␣␣␣␣␣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

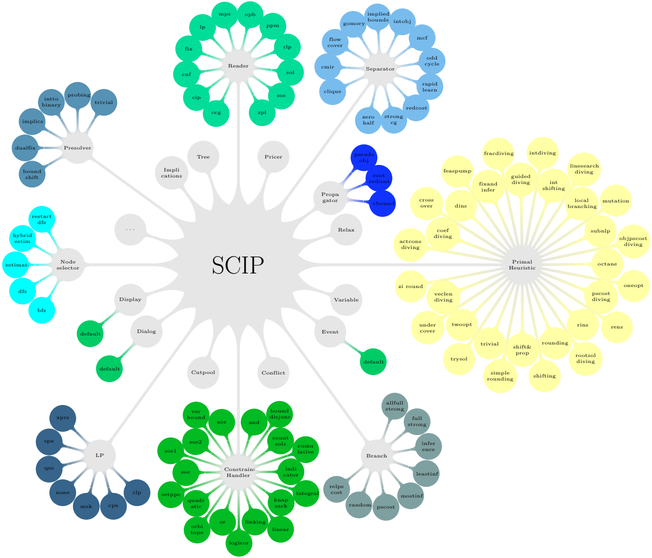
Functions

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

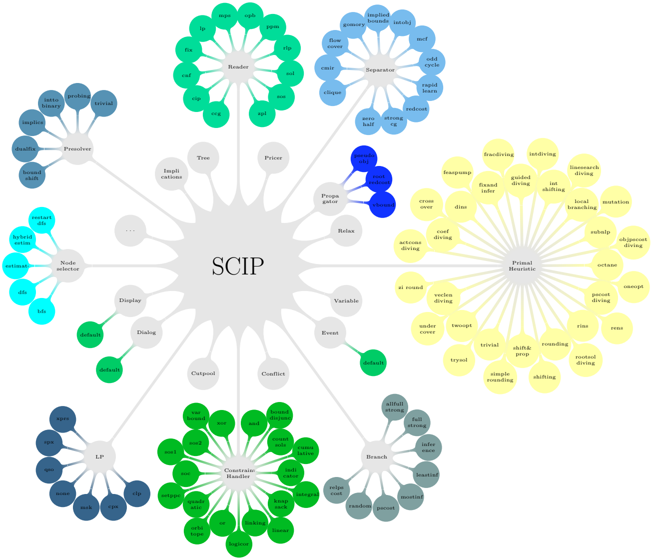
Result

- SCIP is constraint based
    - Advantage: flexibility
    - Disadvantage: limited global view

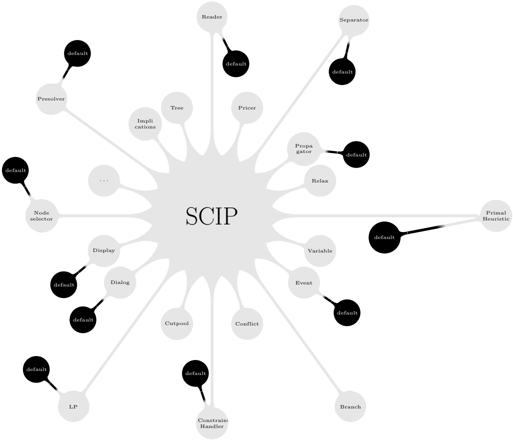
Default Plugins



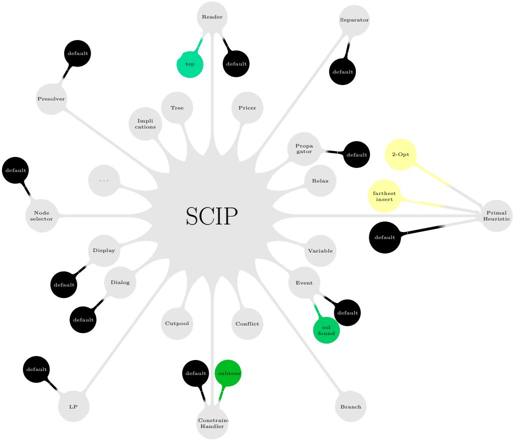
Extending SCIP



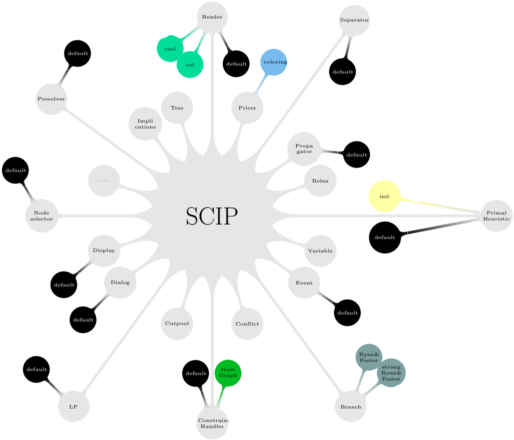
Extending SCIP



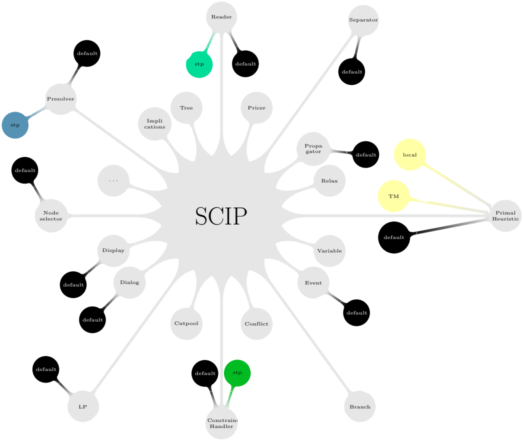
Extending SCIP: TSP



Extending SCIP: Coloring



Extending SCIP: STP



SCIP – Solving Constraint Integer Programs

http://scip.zib.de

- Toolbox for generating and solving constraint integer programs
- free for academic use, available in source code

SCIP Optimization Suite

- Toolbox for generating and solving constraint integer 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

Wrap-up

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