Constraint Integer Programming

Leon Eifler, eifler@zib.de

CO@Work, 2020

Outline

Constraint Integer Programming

SCIP's Design

The Solving Process of SCIP

http://scipopt.org

Outline

Constraint Integer Programming

SCIP's Design

The Solving Process of SCIP

http://scipopt.org

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

Restriction:

- When all integer variables are fixed, remaining subproblem is LP or NLP

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

Remark:

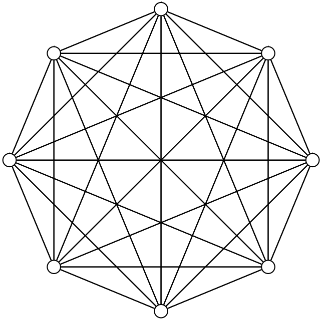
- arbitrary objective or variables modeled by constraints

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.



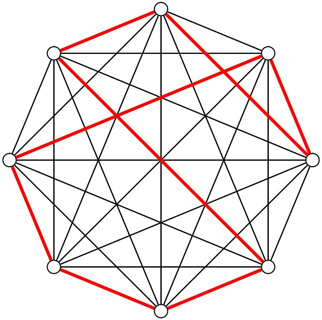
K_8

An Example: the Traveling Salesman Problem

Definition (TSP)

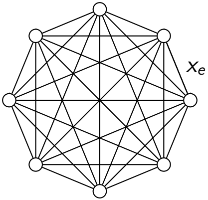
Given a complete graph $G = (V, E)$ and distances d_e for all $e \in E$:

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



K_8

TSP – Integer Programming Formulation



K_8

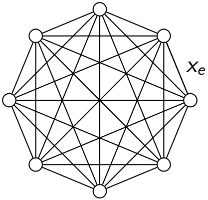
Given

- 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



K_8

Given

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

Binary variables

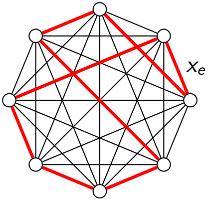
- $x_e = 1$ if edge e is used

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

TSP – Integer Programming Formulation

Given

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



Binary variables

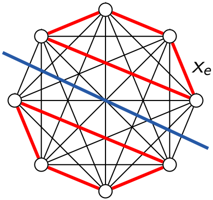
- $x_e = 1$ if edge e is used

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

TSP – Integer Programming Formulation

Given

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



Binary variables

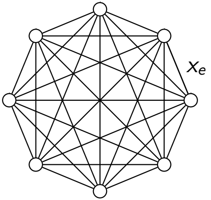
- $x_e = 1$ if edge e is used

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

TSP – Integer Programming Formulation

Given

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



K_8

Binary variables

- $x_e = 1$ if edge e is used

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

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

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

Restriction:

- When all integer variables are fixed, remaining subproblem is LP or NLP

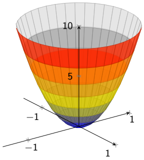
$$
\begin{array}{llll}
\min & \sum_{e \in E} d_e \, x_e & & \\
s.t. & \sum_{e \in \delta(v)} x_e & = 2 & \forall \, v \in V \\
 & \text{nosubtour}(x) & & \\
 & x_e & \in \{0, 1\} & \forall \, e \in E
\end{array}
$$

(CIP formulation of TSP)

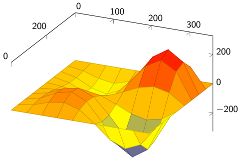
Single nosubtour constraint rules out subtours (e.g. by domain propagation). It may also separate subtour elimination inequalities.

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

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

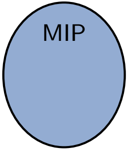


convex or



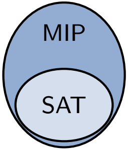
nonconvex

- Mixed Integer Programs



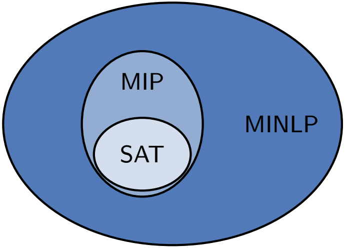
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems



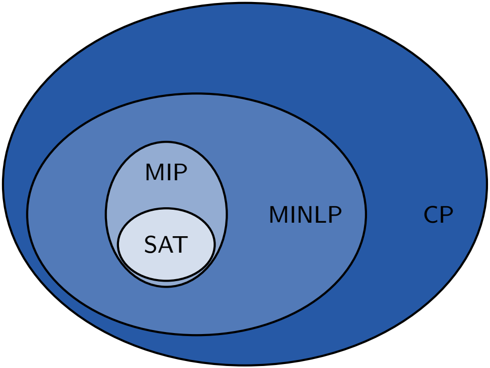
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Mixed Integer Nonlinear Programs



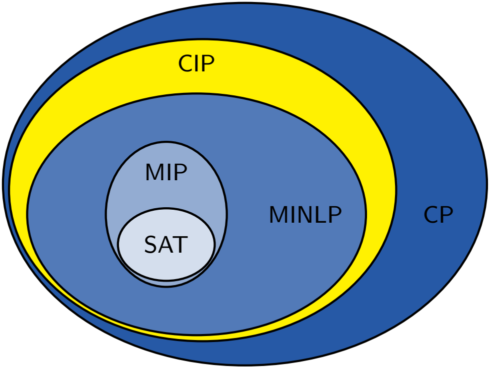
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Mixed Integer Nonlinear Programs
- Constraint Programming



Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Mixed Integer Nonlinear Programs
- Constraint Programming
- Constraint Integer Programming



Quiz - Part I

- What is the key restriction that defines a CIP and why is it necessary?

- Think of a different problem that could benefit from a formulation as a CIP

Quiz - Part I

- What is the key restriction that defines a CIP and why is it necessary?

  When all integer variables are fixed, remaining problem is LP or NLP

- Think of a different problem that could benefit from a formulation as a CIP

Quiz - Part I

- What is the key restriction that defines a CIP and why is it necessary?

  When all integer variables are fixed, remaining problem is LP or NLP

- Think of a different problem that could benefit from a formulation as a CIP

  Possibilities: Linear ordering problem, Steiner tree problem, scheduling, ...

Outline

Constraint Integer Programming

SCIP's Design

The Solving Process of SCIP

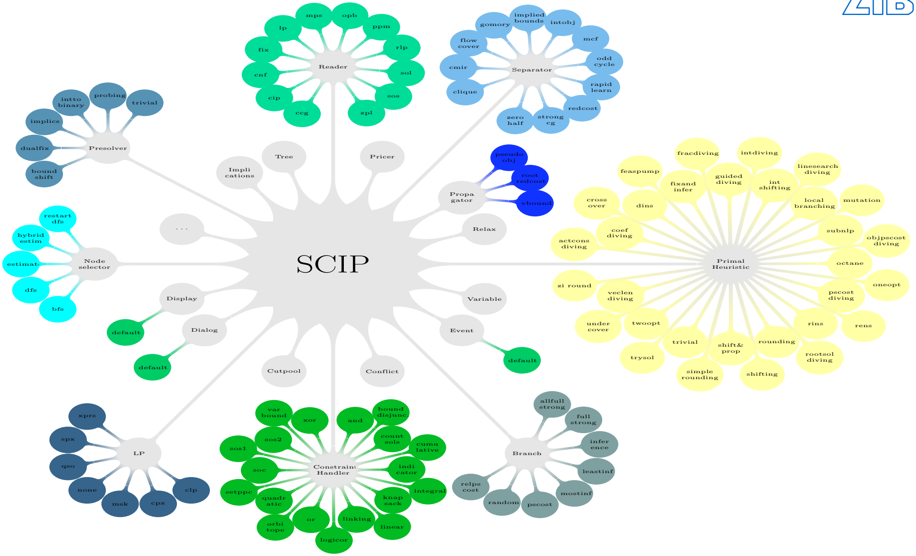
http://scipopt.org

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

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**

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: A constraint knows its variables but a variable does not know the constraints it appears in.
  - LP relaxation is an add-in and handled by external software via LP interface.

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
- Relaxation handlers: custom relaxations
- ...

What does SCIP know about plugins?

- interactive shell shows the information SCIP has for a plugin type

```
SCIP> display {branching | conshdlrs | heuristics | ...}
```

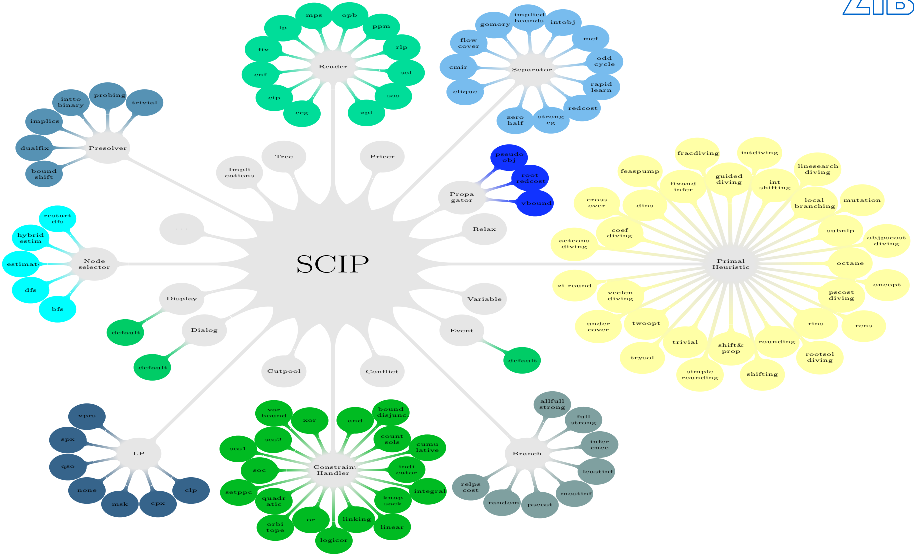
What does SCIP know about plugins?

- interactive shell shows the information SCIP has for a plugin type

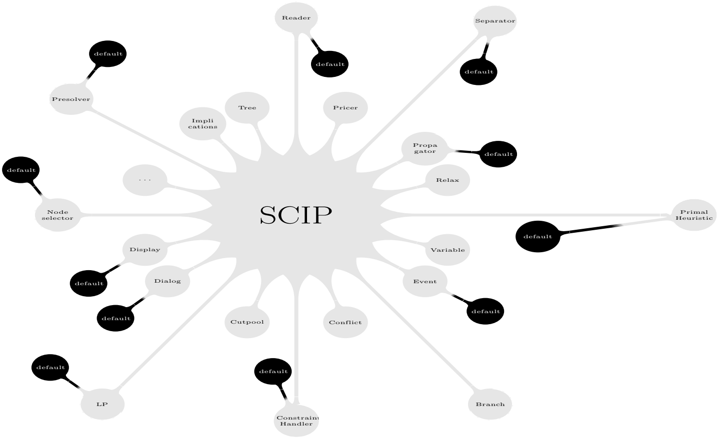
```
SCIP> display {branching | conshdlrs | heuristics | ...}
```

```
SCIP> display branching

branching rule priority maxdepth maxbddist description
-------------- -------- -------- --------- -----------
relpscost         10000       -1    100.0% reliability branching on pseudo cost values
pscost             2000       -1    100.0% branching on pseudo cost values
inference          1000       -1    100.0% inference history branching
mostinf             100       -1    100.0% most infeasible branching
leastinf             50       -1    100.0% least infeasible branching
distribution          0       -1    100.0% branching rule based on variable influence on cumu
fullstrong            0       -1    100.0% full strong branching
cloud                 0       -1    100.0% branching rule that considers several alternative
allfullstrong     -1000       -1    100.0% all variables full strong branching
random          -100000       -1    100.0% random variable branching
```

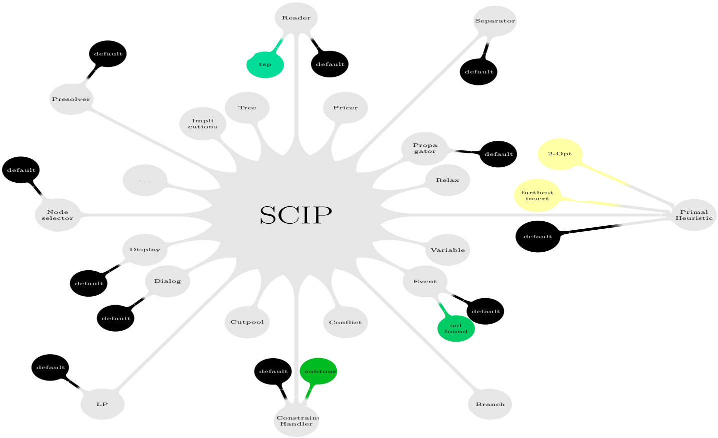
Extending SCIP



Extending SCIP



Extending SCIP: TSP



Summary

Plugins for the SCIP core

- are black boxes for SCIP
- perform all problem specific actions
- interact through a **very detailed interface**

Plugins

- can have private data
- can access global information of the SCIP core
- should not access data of other plugins

Result

- SCIP is **constraint based**
- new types of constraints can easily be defined
- easy to add problem specific algorithms

Advantages and Disadvantages of SCIP

Advantages

- Can be used as a black box MIP solver

Advantages and Disadvantages of SCIP

Advantages

- Can be used as a black box MIP solver
- Hundreds of parameters to play with

Advantages and Disadvantages of SCIP

Advantages

- Can be used as a black box MIP solver
- Hundreds of parameters to play with
- Robust, fast, and well documented

Advantages and Disadvantages of SCIP

Advantages

- Can be used as a black box MIP solver
- Hundreds of parameters to play with
- Robust, fast, and well documented
- **Extendable in any direction since constraint based**
  - Adding global constraints
  - Adding problem specific plugins

Advantages and Disadvantages of SCIP

Advantages

- Can be used as a black box MIP solver
- Hundreds of parameters to play with
- Robust, fast, and well documented
- **Extendable in any direction since constraint based**
  - Adding global constraints
  - Adding problem specific plugins

Disadvantages

- Hundreds of parameters to play with
- In general variables do not know in which constraint they appear since constraint based

Quiz - Part II

- Think of 3 different types of plugins within SCIP
- Why can't plugins directly communicate with each other?
- Recall your own CIP formulation from Quiz - Part I.
  What plugin types would you need for it?

Quiz - Part II

- Think of 3 different types of plugins within SCIP

  Constraint handler, event handler, separator, reader, ...

- Why can't plugins directly communicate with each other?

- Recall your own CIP formulation from Quiz - Part I.
  What plugin types would you need for it?

Quiz - Part II

- Think of 3 different types of plugins within SCIP

  Constraint handler, event handler, separator, reader, ...

- Why can't plugins directly communicate with each other?

  Keep the framework as flexible as possible, no interdependencies

- Recall your own CIP formulation from Quiz - Part I.
  What plugin types would you need for it?

Quiz - Part II

- Think of 3 different types of plugins within SCIP

  Constraint handler, event handler, separator, reader, ...

- Why can't plugins directly communicate with each other?

  Keep the framework as flexible as possible, no interdependencies

- Recall your own CIP formulation from Quiz - Part I.
  What plugin types would you need for it?

  Very likely: Constraint handler, reader, heuristic

Outline

Constraint Integer Programming

SCIP's Design

The Solving Process of SCIP

http://scipopt.org

SCIP Interactive Shell Basics

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`

Interfaces to SCIP

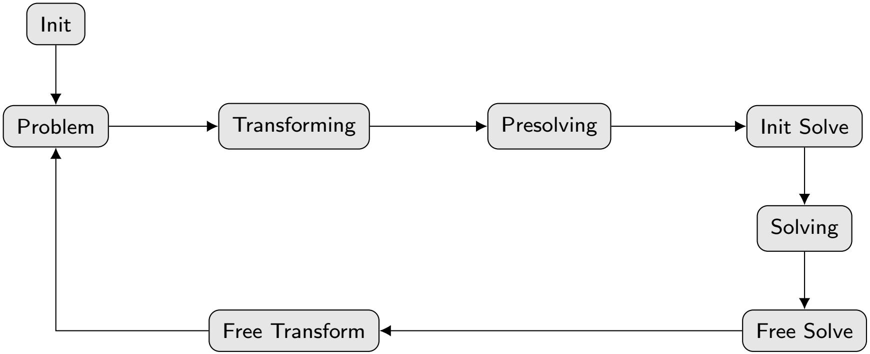
- interactive shell supports 11 different input formats
  → cip, cnf, flatzinc, rlp, lp, mps, opb, pip, wbo, zimpl, smps
- C API/callable library
- C++ wrapper classes
- Python interface
  - New Dockerized SCIP
  - conda integration coming soon
- Java JNI interface
- Julia interface
- AMPL
- GAMS
- Matlab (see also OPTI toolbox, http://www.i2c2.aut.ac.nz/Wiki/OPTI/)

Getting help

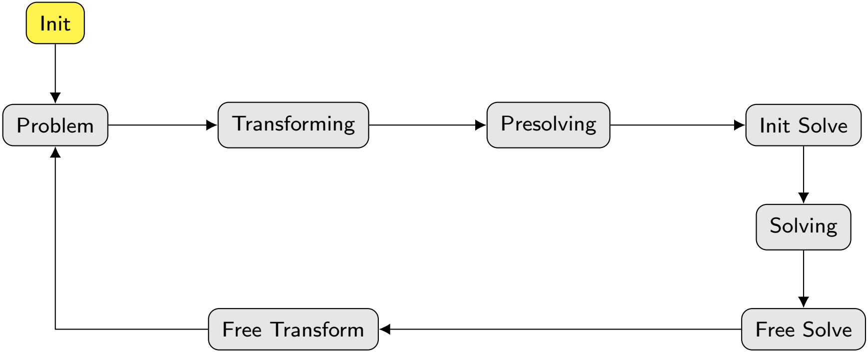
If you should ever get stuck, you can . . .

1. type `help` in the interactive shell
2. read the documentation `http://scipopt.org/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)
   - register `http://listserv.zib.de/mailman/listinfo/scip/` and post

Operational Stages

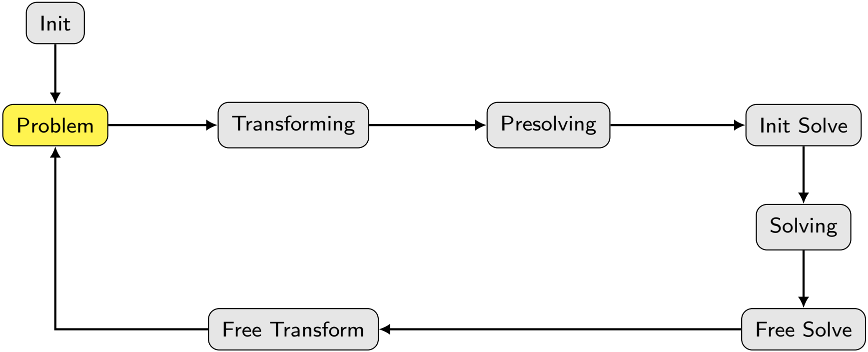


Operational Stages



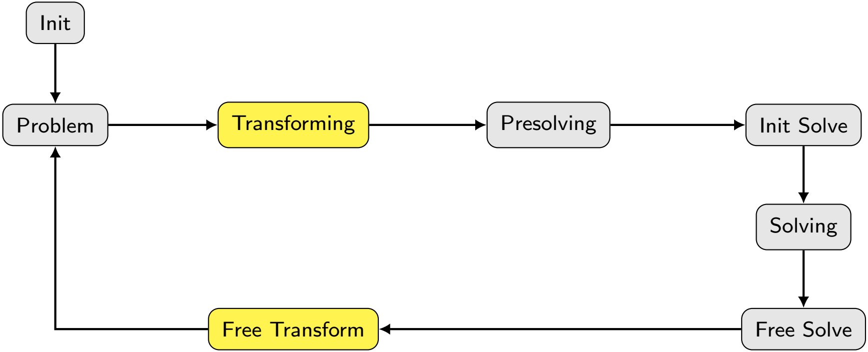
- Basic data structures are allocated and initialized.
- User includes required plugins (or just takes default plugins).

Problem Specification



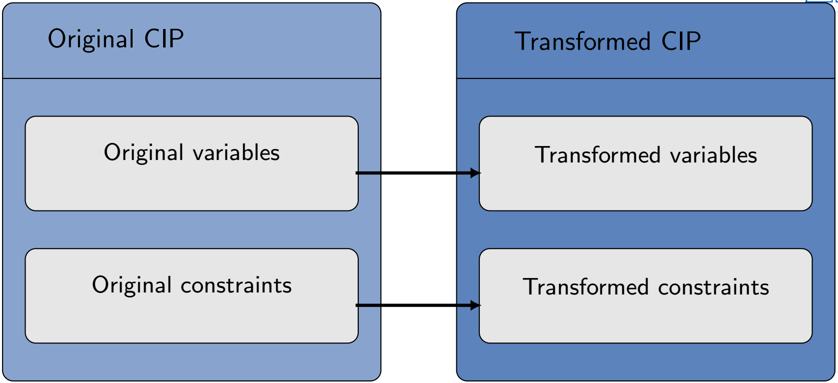
- User creates and modifies the original problem instance.
- Problem creation is usually done in file readers.

Transformation



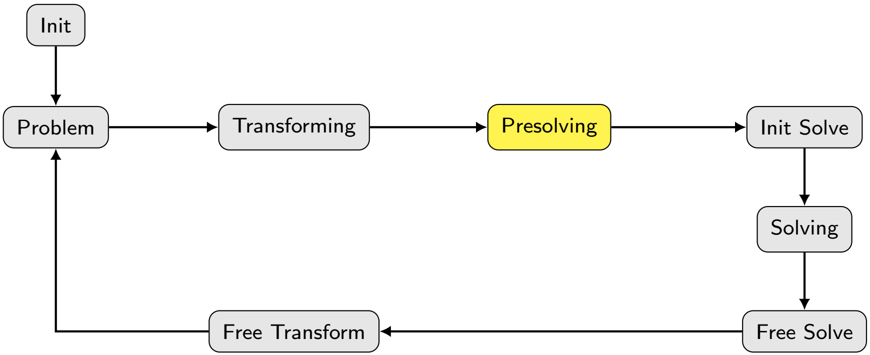
- Creates a working copy of the original problem.

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 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
```

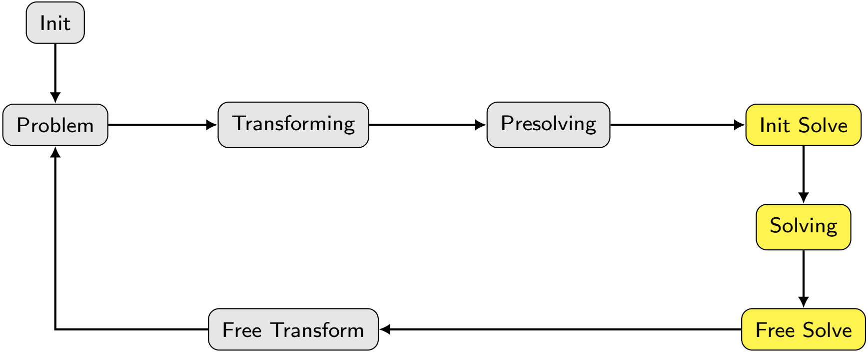
Aggressive Presolving

```
set presolving emphasis aggressive
```

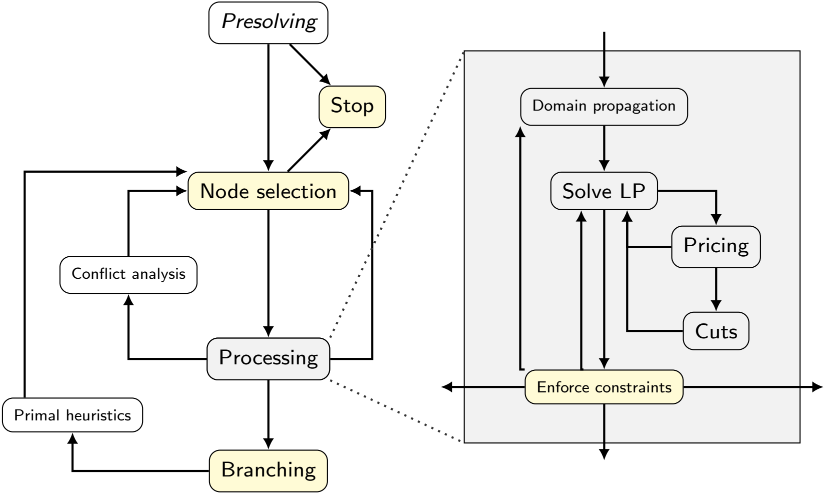
General Rule of Thumb

Only deactivate single presolving techniques if you encounter performance problems.

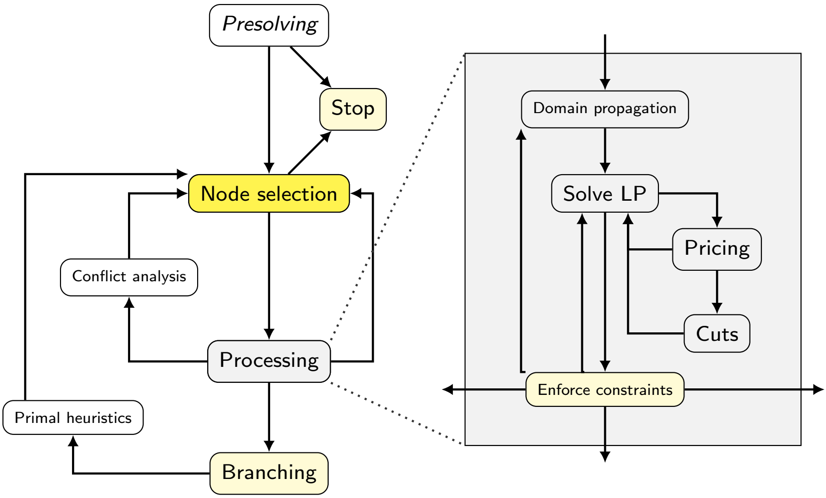
Solving


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

Flow Chart SCIP



Flow Chart SCIP



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
```

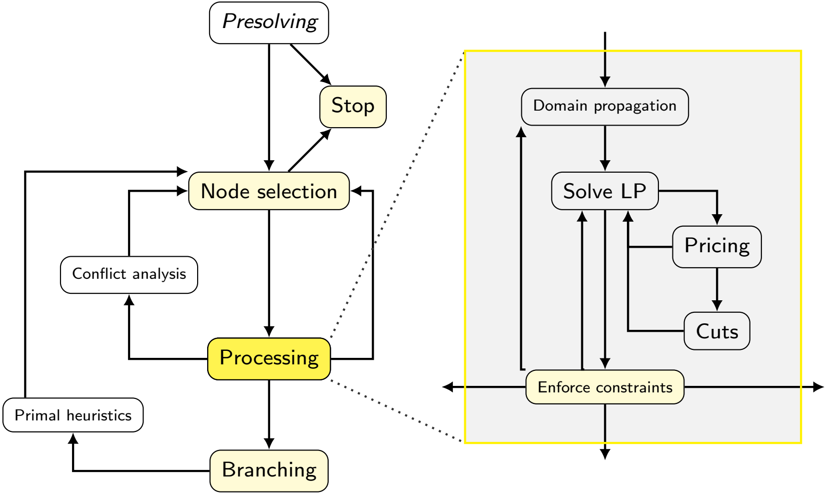
Switching Node Selectors

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

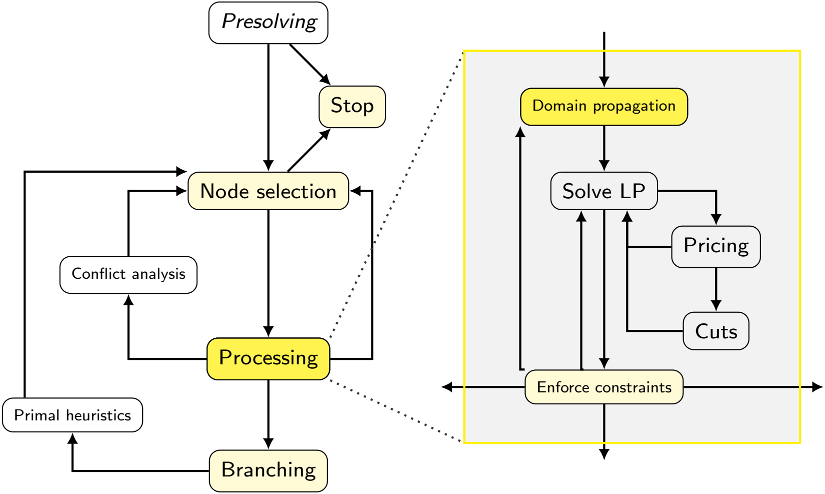
```
set nodeselection dfs stdpriority 1000000
```

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

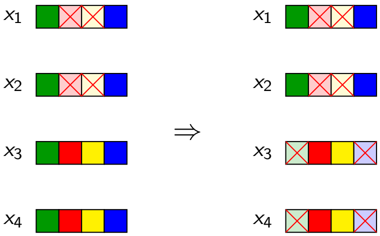
Flow Chart SCIP



Flow Chart SCIP



Domain Propagation



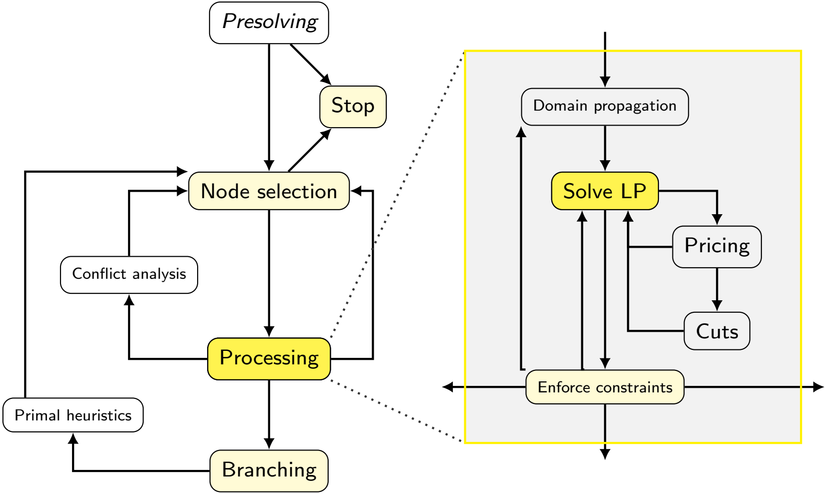
Task

- 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

Flow Chart SCIP



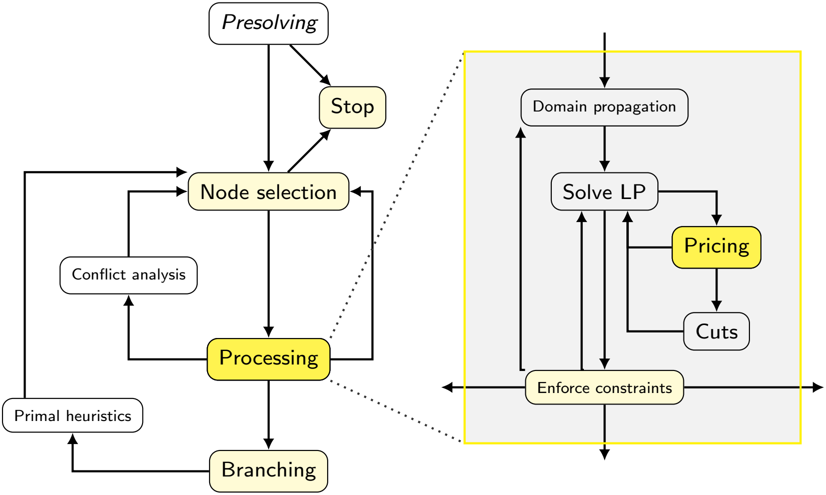
LP Solving Tips and Parameters

Most Important LP Parameters

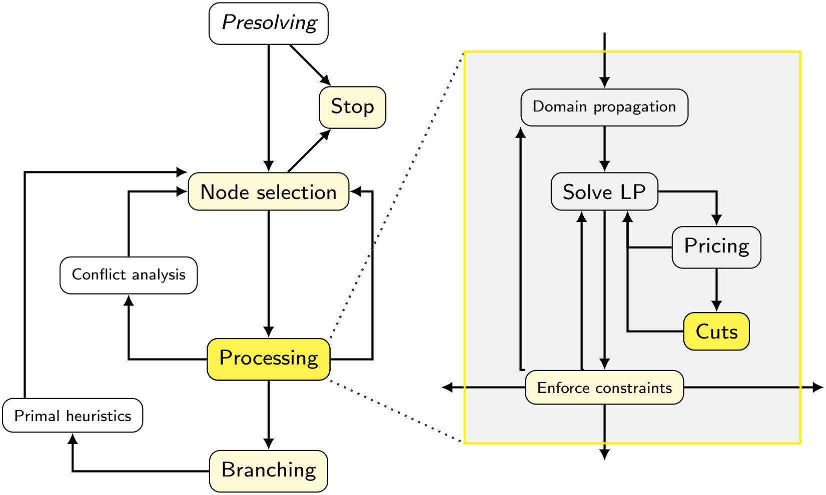
- `lp/initalgorithm`, `lp/resolvealgorithm`
  - Primal/Dual Simplex Algorithm
  - Barrier w and w/o crossover
- `lp/pricing`
  - normally LP solver specific default
  - Devex
  - Steepest edge
  - Quick start steepest edge
- `lp/threads`

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

Flow Chart SCIP



Flow Chart SCIP



Separation Tips and Parameters

Disable/Speed up/Emphasize All Separation

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

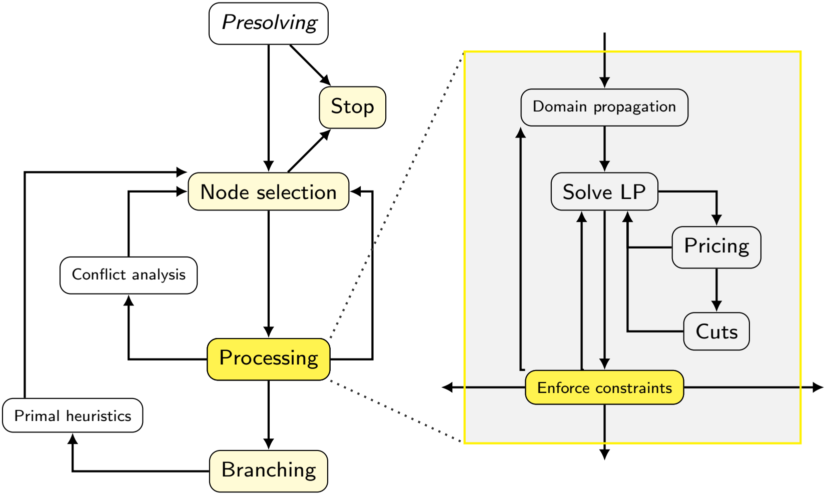
Disable Single Separation Techniques

```
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

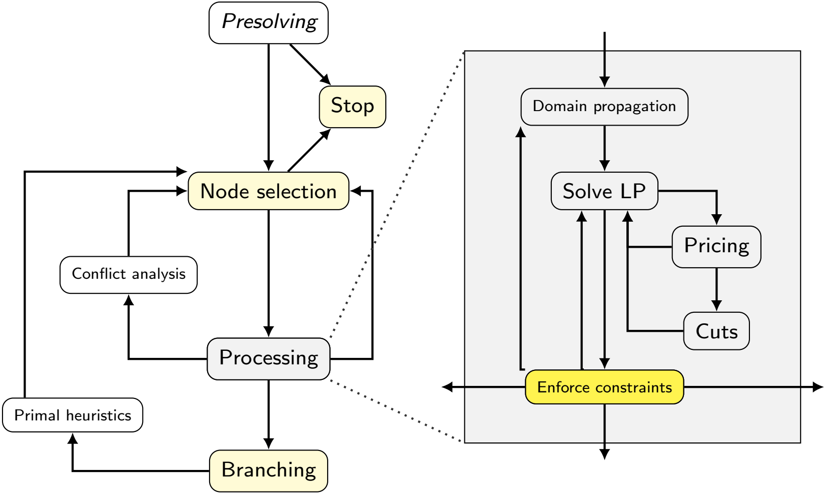
LP solution may violate a constraint not contained in the relaxation.

Enforcing is necessary for a correct implementation!

Constraint handler resolves the infeasibility by ...

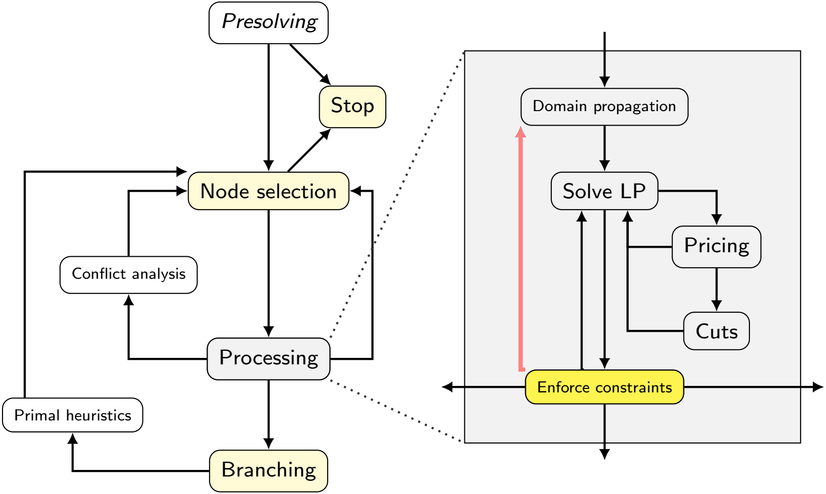
- Reducing a variable's domain,
- Separating a cutting plane (may use integrality),
- Adding a (local) constraint,
- Creating a branching,
- Concluding that the subproblem is infeasible and can be cut off, or
- Just saying "solution infeasible".

Constraint Enforcement



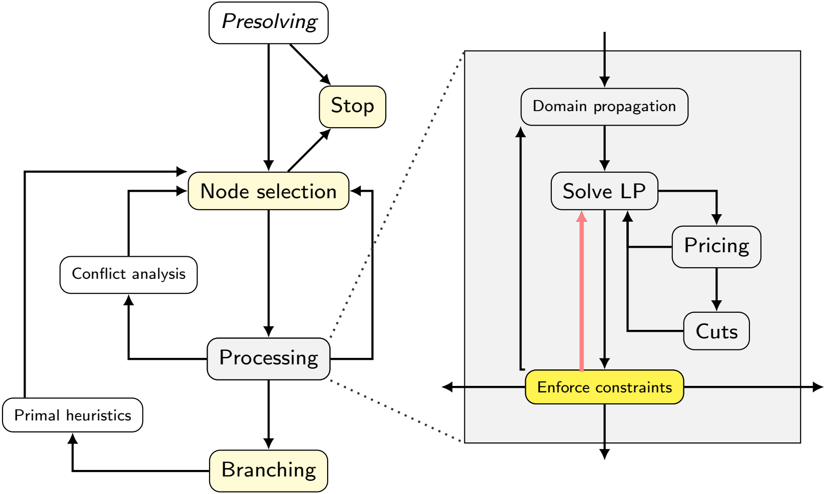
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



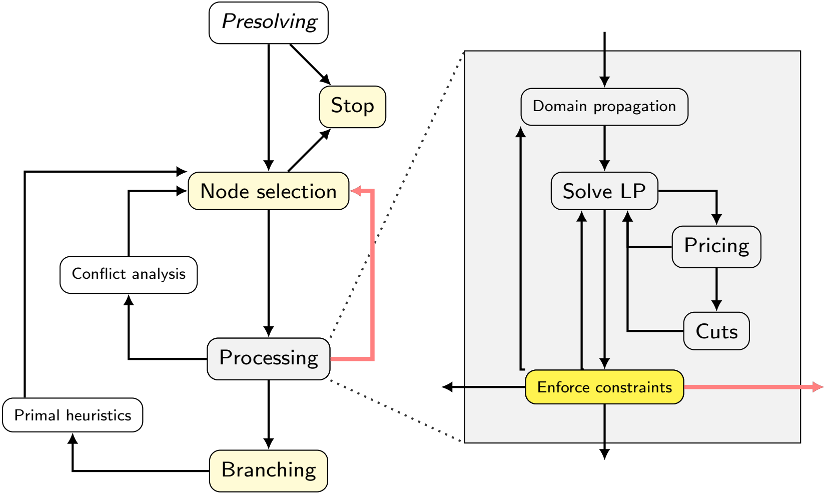
- **Reduced domain**
- **Added constraint**
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



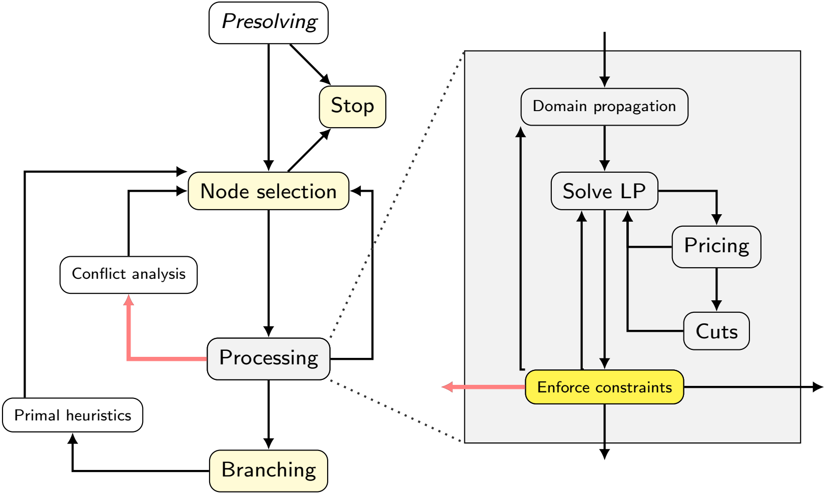
- Reduced domain
- Added constraint
- **Added cut**
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



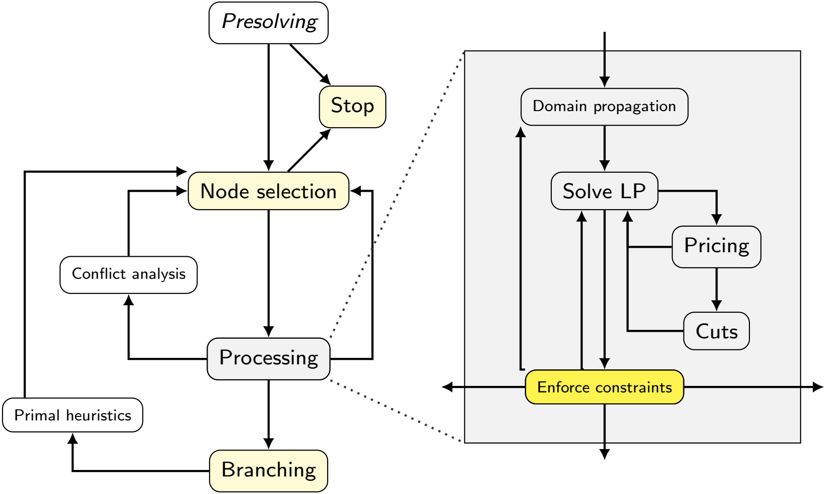
- Reduced domain
- Added constraint
- Added cut
- **Branched**
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



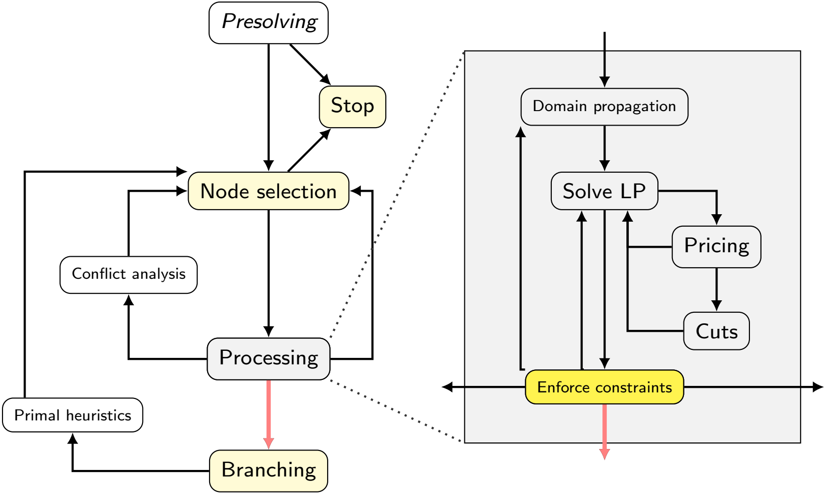
- Reduced domain
- Added constraint
- Added cut
- Branched
- **Cutoff**
- Infeasible
- Feasible

Constraint Enforcement



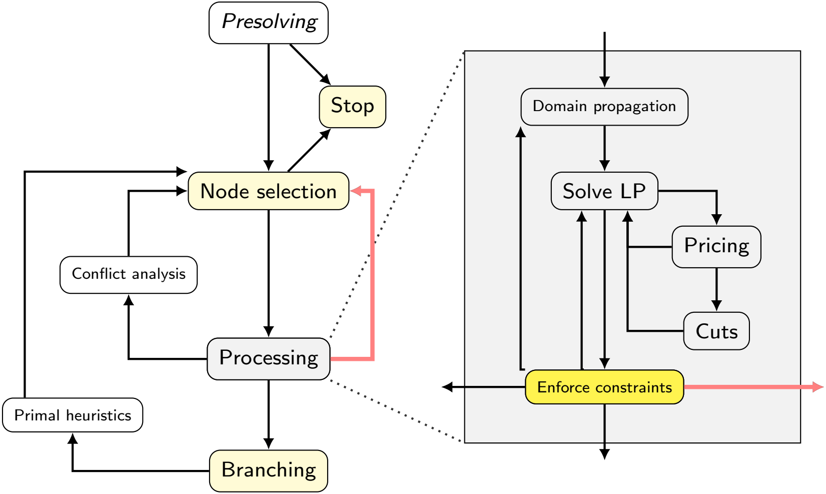
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- **Infeasible**
- **Feasible**

Constraint Enforcement



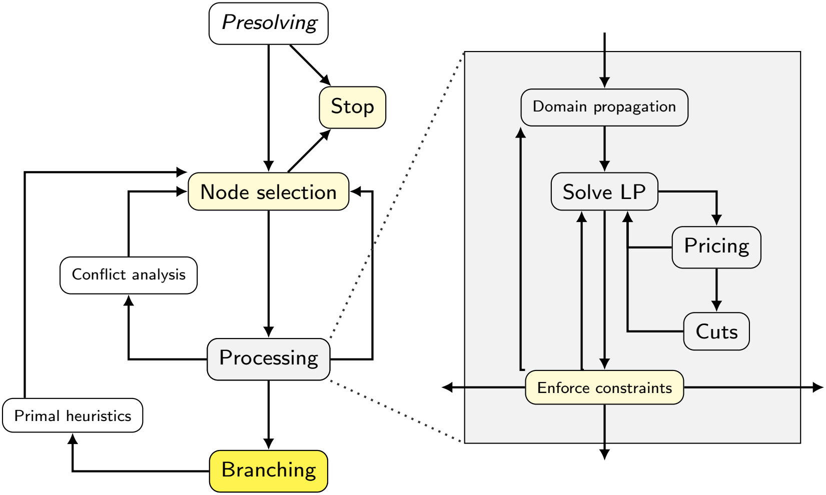
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- **Infeasible**
- Feasible

Constraint Enforcement



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

Flow Chart SCIP



Branching Rule Tips and Parameters

Branching Rule Selection

Branching rules are applied in decreasing order of priority.

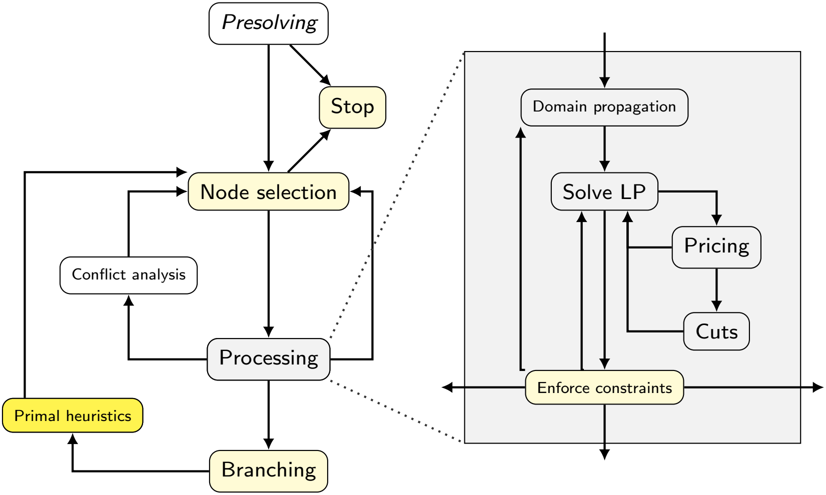
```
SCIP> display branching

 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 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.

Quiz - Part III

- Why does SCIP create the transformed problem?
- What is the difference between enforcing and checking a solution?
- During which stage do you have to include user-created plugins?

Quiz - Part III

- Why does SCIP create the transformed problem?

  Preserve original problem, ensure feasibility

- What is the difference between enforcing and checking a solution?

- During which stage do you have to include user-created plugins?

Quiz - Part III

- Why does SCIP create the transformed problem?

  Preserve original problem, ensure feasibility

- What is the difference between enforcing and checking a solution?

  Enforcing resolves infeasibility

- During which stage do you have to include user-created plugins?

Quiz - Part III

- Why does SCIP create the transformed problem?

  Preserve original problem, ensure feasibility

- What is the difference between enforcing and checking a solution?

  Enforcing resolves infeasibility

- During which stage do you have to include user-created plugins?

  The init stage

Advanced Topics And Further Reading

Important SCIP topics not covered in this talk:

- more Details in Documentation
- branch-and-price: SCIP can be extended to a problem-specific branch-cut-and-price solver
  - Q&A on youtube channel
  - see The Bin Packing Example in C
  - see also GCG
- allows for Benders decomposition since version 6.0
  - Q&A on youtube channel
- browse technical reports for details on recently added cutting plane selection, primal heuristics, symmetry breaking, and much more

Thank you for your attention