Basic Concepts of Constraint Integer Programming

Ambros Gleixner

Zuse Institute Berlin

September 30, 2015

Research Center MATHEON
Mathematics for Key Technologies

Berlin Mathematical School

MODAL
Mathematical Optimization and Data Analysis Laboratories

Outline

SCIP – Solving Constraint Integer Programs

Outline

SCIP – Solving Constraint Integer Programs

4 methodologies in optimization

An integrated method

SCIP: Solving Constraint Integer Programs

The Solving Process of SCIP

Mixed-Integer Linear Programming: MIP

Problem class

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

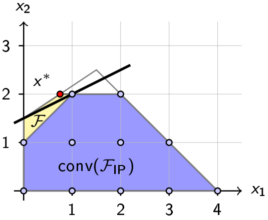
- continuous and integer variables
- linear objective function
- linear constraints

Mixed-Integer Linear Programming: MIP

Problem class

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

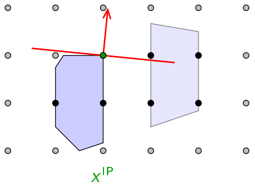
- continuous and integer variables
- linear objective function
- linear constraints



$\min\{c^T x : x \in \text{conv}(\mathcal{F}_{\text{IP}})\}$

Methods

- LP relaxation
- cutting planes
- branch-and-bound



More details: Tobias Achterberg and Bob Bixby, Oct. 5

Satisfiability Solving: SAT

Problem class

$$
\begin{aligned}
&\exists x \in \{0,1\}^n \\
s.t.\ &\bigvee_{i \in \mathcal{Y}_k} x_i \vee \bigvee_{i \in \mathcal{N}_k} \neg x_i \text{ for } k = 1, \dots, m, \\
&\qquad\qquad\qquad \mathcal{Y}_k, \mathcal{N}_k \subseteq \{1, \dots, n\}
\end{aligned}
$$

- binary variables and their negation
- (linear) clauses
- feasible assignment?

Satisfiability Solving: SAT

Problem class

$$\exists x \in \{0,1\}^n$$
$$s.t. \bigvee_{i\in\mathcal{Y}_k} x_i \vee \bigvee_{i\in\mathcal{N}_k} \neg x_i \text{ for } k = 1,\dots,m,$$
$$\mathcal{Y}_k, \mathcal{N}_k \subseteq \{1,\dots,n\}$$

- binary variables and their negation
- (linear) clauses
- feasible assignment?

Methods

- unit propagation

$$x_1 \vee x_3$$
$$x_1 \vee x_2 \vee x_4$$
$$\neg x_3$$

Satisfiability Solving: SAT

Problem class

$$\exists x \in \{0,1\}^n$$
$$s.t. \bigvee_{i \in \mathcal{Y}_k} x_i \vee \bigvee_{i \in \mathcal{N}_k} \neg x_i \text{ for } k = 1, \dots, m,$$
$$\mathcal{Y}_k, \mathcal{N}_k \subseteq \{1, \dots, n\}$$

- binary variables and their negation
- (linear) clauses
- feasible assignment?

Methods

- unit propagation

$$\begin{array}{r} x_1 \vee x_3 \\ x_1 \vee x_2 \vee x_4 \\ \neg x_3 \end{array} \Rightarrow \begin{array}{r} x_1 \\ x_1 \vee x_2 \vee x_4 \\ \neg x_3 \end{array}$$

Satisfiability Solving: SAT

Problem class

$$\exists x \in \{0,1\}^n$$
$$s.t. \bigvee_{i\in\mathcal{Y}_k} x_i \vee \bigvee_{i\in\mathcal{N}_k} \neg x_i \text{ for } k = 1,\ldots,m,$$
$$\mathcal{Y}_k, \mathcal{N}_k \subseteq \{1,\ldots,n\}$$

- binary variables and their negation
- (linear) clauses
- feasible assignment?

Methods

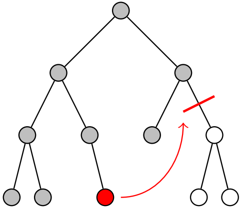
- unit propagation

$$\begin{array}{r} x_1 \vee x_3 \\ x_1 \vee x_2 \vee x_4 \\ \neg x_3 \end{array} \Rightarrow \begin{array}{r} x_1 \\ x_1 \vee x_2 \vee x_4 \\ \neg x_3 \end{array} \Rightarrow \begin{array}{r} x_1 \\ \\ \neg x_3 \end{array}$$

Satisfiability Solving: SAT

Problem class

$$\exists x \in \{0,1\}^n$$
$$s.t. \bigvee_{i \in \mathcal{Y}_k} x_i \vee \bigvee_{i \in \mathcal{N}_k} \neg x_i \text{ for } k = 1,\dots,m,$$
$$\mathcal{Y}_k, \mathcal{N}_k \subseteq \{1,\dots,n\}$$



- binary variables and their negation
- (linear) clauses
- feasible assignment?

Methods

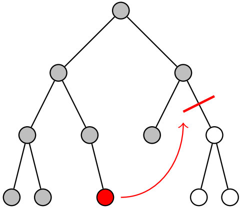
- unit propagation
- tree search
- clause learning

$$\begin{array}{c} x_1 \vee x_3 \\ x_1 \vee x_2 \vee x_4 \\ \neg x_3 \end{array} \Rightarrow \begin{array}{c} x_1 \\ x_1 \vee x_2 \vee x_4 \\ \neg x_3 \end{array} \Rightarrow \begin{array}{c} x_1 \\ \\ \neg x_3 \end{array}$$

Satisfiability Solving: SAT

Problem class

$$\exists x \in \{0,1\}^n$$
$$s.t. \bigvee_{i \in \mathcal{Y}_k} x_i \vee \bigvee_{i \in \mathcal{N}_k} \neg x_i \text{ for } k = 1, \dots, m,$$
$$\mathcal{Y}_k, \mathcal{N}_k \subseteq \{1, \dots, n\}$$



- binary variables and their negation
- (linear) clauses
- feasible assignment?

Methods

- unit propagation
- tree search
- clause learning
- restarts

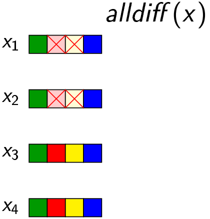
$$\begin{array}{ccccc} x_1 \vee x_3 & & x_1 & & x_1 \\ x_1 \vee x_2 \vee x_4 & \Rightarrow & x_1 \vee x_2 \vee x_4 & \Rightarrow & \\ \neg x_3 & & \neg x_3 & & \neg x_3 \end{array}$$

Constraint Programming: CP

Problem class

$$\begin{aligned} \min \quad & c(x) \\ s.t. \quad & x \in F_k \text{ for } k = 1, \dots, m \\ & (x_I, x_N) \in \mathbb{Z}^I \times X \end{aligned}$$

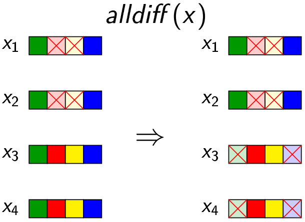
- arbitrary variable domains (usually finite: FD)
- arbitrary constraints
- arbitrary objective function



Constraint Programming: CP

Problem class

$$\begin{array}{ll} \min & c(x) \\ s.t. & x \in F_k \text{ for } k = 1, \dots, m \\ & (x_I, x_N) \in \mathbb{Z}^I \times X \end{array}$$



- arbitrary variable domains (usually finite: FD)
- arbitrary constraints
- arbitrary objective function

Methods

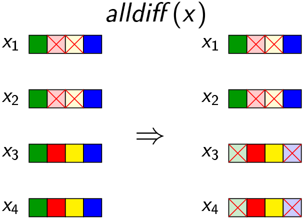
- constraint propagation

Constraint Programming: CP

Problem class

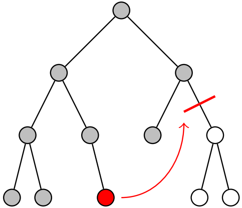
$$\begin{aligned} \min \quad & c(x) \\ s.t. \quad & x \in F_k \text{ for } k = 1, \dots, m \\ & (x_I, x_N) \in \mathbb{Z}^I \times X \end{aligned}$$

- arbitrary variable domains (usually finite: FD)
- arbitrary constraints
- arbitrary objective function



Methods

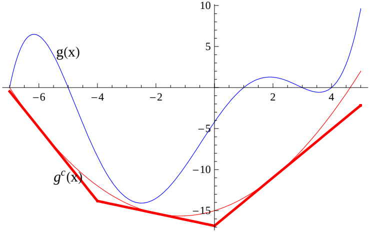
- constraint propagation
- tree search
- conflict analysis/no-good learning



Mixed-Integer Nonlinear Programming: MINLP

Problem class

$$\begin{array}{ll} \min & f(x) \\ s.t. & g(x) \leq b \\ & (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C \end{array}$$



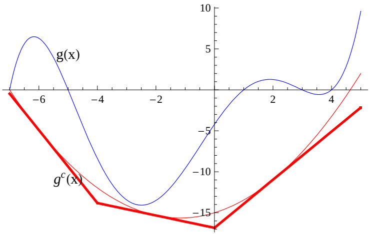
- continuous and integer variables
- nonlinear objective function
- nonlinear constraint functions

More details: Ralf Lenz, Jesco Humpola, Pierre Bonami, Oct. 1 & 8

Mixed-Integer Nonlinear Programming: MINLP

Problem class

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



- continuous and integer variables
- nonlinear objective function
- nonlinear constraint functions

Methods

- outer approximation
- convex relaxation
- bound tightening

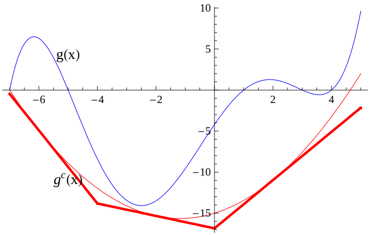
More details: Ralf Lenz, Jesco Humpola, Pierre Bonami, Oct. 1 & 8

Mixed-Integer Nonlinear Programming: MINLP

Problem class

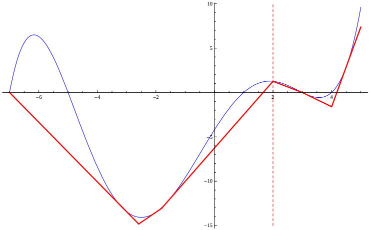
$$\begin{array}{ll} \min & f(x) \\ s.t. & g(x) \leq b \\ & (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C \end{array}$$

- continuous and integer variables
- nonlinear objective function
- nonlinear constraint functions



Methods

- outer approximation
- convex relaxation
- bound tightening
- spatial branch-and-bound

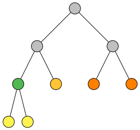


More details: Ralf Lenz, Jesco Humpola, Pierre Bonami, Oct. 1 & 8

SCIP – Solving Constraint Integer Programs

Integrating CP, MIP, SAT, and MINLP

Search: in MIP



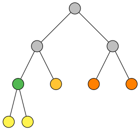
in SAT

in CP

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

\+ LP relaxation

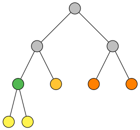
in SAT

in CP

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

\+ LP relaxation

\+ cutting planes

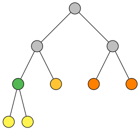
in SAT

in CP

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP
+ LP relaxation
+ cutting planes

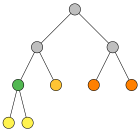
in SAT
+ clause learning

in CP

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

+ LP relaxation
+ cutting planes

in SAT

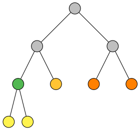
+ clause learning
+ restarts

in CP

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

+ LP relaxation
+ cutting planes

in SAT

+ clause learning
+ restarts

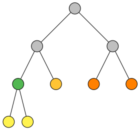
in CP

+ propagation

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

+ LP relaxation
+ cutting planes

in SAT

+ clause learning
+ restarts

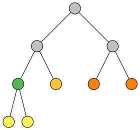
in CP

+ propagation
+ conflict analysis

in MINLP

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

+ LP relaxation
+ cutting planes

in SAT

+ clause learning
+ restarts

in CP

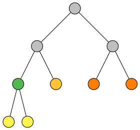
+ propagation
+ conflict analysis

in MINLP

+ outer approximation

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

+ LP relaxation
+ cutting planes

in SAT

+ clause learning
+ restarts

in CP

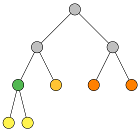
+ propagation
+ conflict analysis

in MINLP

+ outer approximation
+ bound tightening

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP

\+ LP relaxation

\+ cutting planes

in SAT

\+ clause learning

\+ restarts

in CP

\+ propagation

\+ conflict analysis

in MINLP

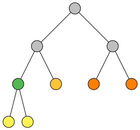
\+ outer approximation

\+ bound tightening

\+ spatial branching

Integrating CP, MIP, SAT, and MINLP

Search:



in MIP
+ LP relaxation
+ cutting planes

in SAT
+ clause learning
+ restarts

in CP
+ propagation
+ conflict analysis

in MINLP
+ outer approximation
+ bound tightening
+ spatial branching

High-level and low-level integration

▷ interaction of different algorithms

▷ combination of algorithmic techniques

e.g., Althaus, Bockmayr, Elf, Jünger, Kasper, Mehlhorn 2002;
Hooker 2007;
Achterberg 2007;
Berthold, Heinz, Vigerske 2010;
Vigerske 2013;
...

Constraint Integer Programming: CIP

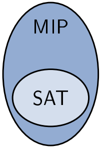
Low-level integration of solving techniques into one algorithm

- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP

Constraint Integer Programming: CIP

Low-level integration of solving techniques into one algorithm

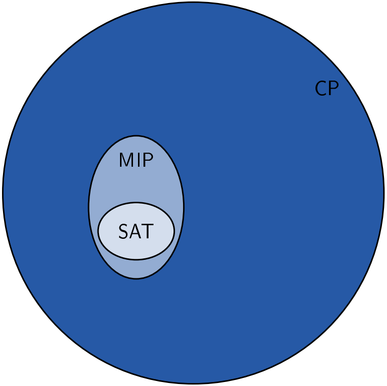
- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP



Constraint Integer Programming: CIP

Low-level integration of solving techniques into one algorithm

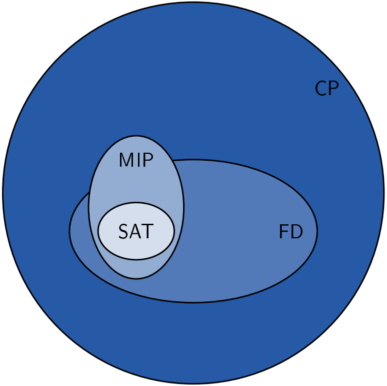
- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP



Constraint Integer Programming: CIP

Low-level integration of solving techniques into one algorithm

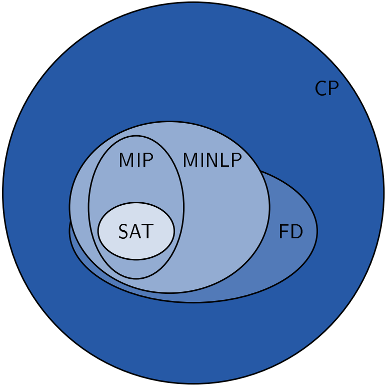
- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP



Constraint Integer Programming: CIP

Low-level integration of solving techniques into one algorithm

- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP



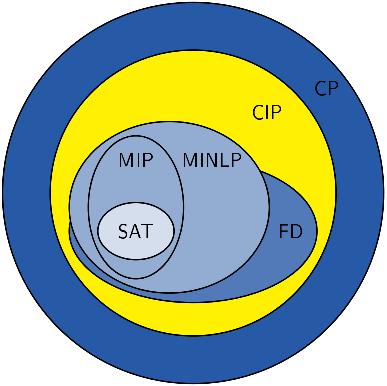
Constraint Integer Programming: CIP

Low-level integration of solving techniques into one algorithm

- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP

Problem class

- continuous and integer variables
- linear objective function
- arbitrary constraints
- condition: after fixing all integers, CIP can be solved as an LP or NLP



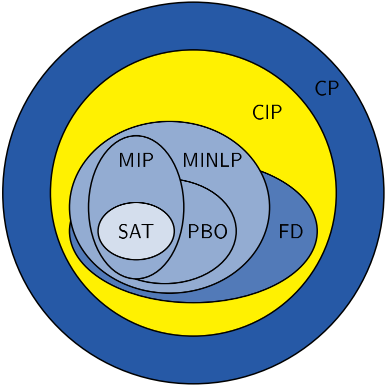
Constraint Integer Programming: CIP

Low-level integration of solving techniques into one algorithm

- CP+SAT+MIP [Achterberg 2007, 2009]
- +MINLP [Berthold, Heinz, Vigerske 2010; Vigerske 2013]
- implemented in the solver SCIP

Problem class

- continuous and integer variables
- linear objective function
- arbitrary constraints
- condition: after fixing all integers, CIP can be solved as an LP or NLP



Example: Pseudoboolean Optimization

The AND constraint

- $y = \prod_{i \in J} x_i$
- $y \in \{0, 1\}$: resultant
- $x_i \in \{0, 1\}$: operand variables

Example: Pseudoboolean Optimization

The AND constraint

- $y = \prod_{i \in J} x_i$
- $y \in \{0, 1\}$: resultant
- $x_i \in \{0, 1\}$: operand variables

Propagation rules

- $\exists i : x_i = 0 \Rightarrow y = 0$

Example: Pseudoboolean Optimization

The AND constraint

- $y = \prod_{i \in J} x_i$
- $y \in \{0, 1\}$: resultant
- $x_i \in \{0, 1\}$: operand variables

Propagation rules

- $\exists i : x_i = 0 \Rightarrow y = 0$
- $x_i = 1 \; \forall i \in J \Rightarrow y = 1$

Example: Pseudoboolean Optimization

The AND constraint

- $y = \prod_{i \in J} x_i$
- $y \in \{0, 1\}$: resultant
- $x_i \in \{0, 1\}$: operand variables

Propagation rules

- $\exists i : x_i = 0 \Rightarrow y = 0$
- $x_i = 1 \ \forall i \in J \Rightarrow y = 1$
- $y = 1 \Rightarrow x_i = 1 \ \forall i \in J$

Example: Pseudoboolean Optimization

The AND constraint

- $y = \prod_{i \in J} x_i$
- $y \in \{0, 1\}$: resultant
- $x_i \in \{0, 1\}$: operand variables

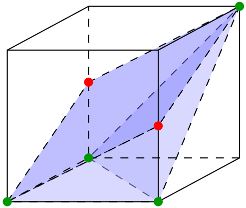
Propagation rules

- $\exists i : x_i = 0 \Rightarrow y = 0$
- $x_i = 1 \;\forall i \in J \Rightarrow y = 1$
- $y = 1 \Rightarrow x_i = 1 \;\forall i \in J$
- $y = 0 \wedge \exists k : x_i = 1 \;\forall i \in J \setminus \{k\} \Rightarrow x_k = 0$

Linearization of $y = x_1 \wedge \cdots \wedge x_n$

$$\sum_{i=1}^{n} x_i - y \leq n - 1$$

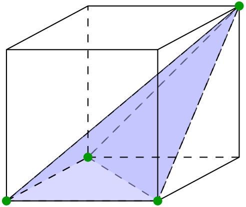
$$\sum_{i=1}^{n} x_i - n\,y \geq 0$$



- 2 constraints
- contains fractional vertices

$$\sum_{i=1}^{n} x_i - y \leq n - 1$$

$$x_i - y \geq 0 \quad \text{for } i = 1, \ldots, n$$



- n + 1 constraints
- only integer vertices

Algorithms

Only propagation

- good: fast subproblem processing
- bad: LP relaxation has no knowledge about the nonlinear structure

Algorithms

Only propagation

- good: fast subproblem processing
- bad: LP relaxation has no knowledge about the nonlinear structure

Only relaxation – put the complete linearization into the LP

- good: LP relaxation contains complete problem
- bad: can blow up the LP

Algorithms

Only propagation

- good: fast subproblem processing
- bad: LP relaxation has no knowledge about the nonlinear structure

Only relaxation – put the complete linearization into the LP

- good: LP relaxation contains complete problem
- bad: can blow up the LP

Only separation – generate the linearization as they are needed

- good: the LP only is fed with the import constraint (cuts)
- bad: LP relaxation has partial knowledge about the nonlinearities

Algorithms

Only propagation

- good: fast subproblem processing
- bad: LP relaxation has no knowledge about the nonlinear structure

Only relaxation – put the complete linearization into the LP

- good: LP relaxation contains complete problem
- bad: can blow up the LP

Only separation – generate the linearization as they are needed

- good: the LP only is fed with the import constraint (cuts)
- bad: LP relaxation has partial knowledge about the nonlinearities

SCIP can implement all strategies

default: propagation + weak linearization + separation

Outline

SCIP – Solving Constraint Integer Programs

4 methodologies in optimization

An integrated method

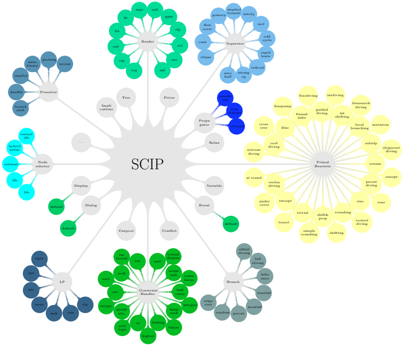
SCIP: Solving Constraint Integer Programs

The Solving Process of SCIP

Why use SCIP?

SCIP: Solving Constraint Integer Programs . . .

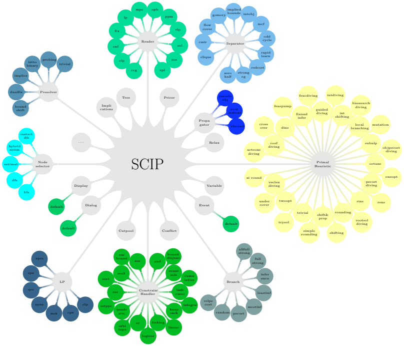
- provides a full-scale MIP and MINLP solver



Why use SCIP?

SCIP: Solving Constraint Integer Programs . . .

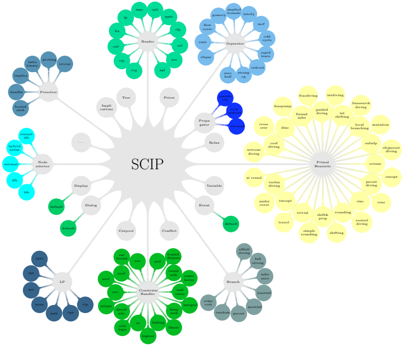
- provides a full-scale MIP and MINLP solver
- can solve general CIPs:
  constraint-based design



Why use SCIP?

SCIP: Solving Constraint Integer Programs ...

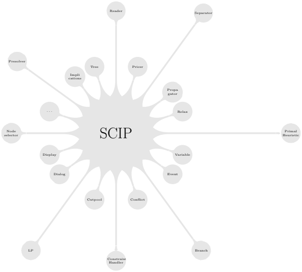
- provides a full-scale MIP and MINLP solver
- can solve general CIPs:
  constraint-based design
- can be extended:
  plugin-based design



Why use SCIP?

SCIP: Solving Constraint Integer Programs . . .

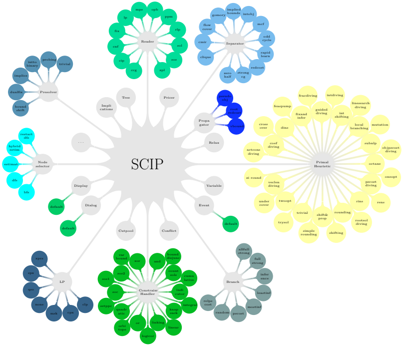
- provides a full-scale MIP and MINLP solver
- can solve general CIPs:
  constraint-based design
- can be extended:
  plugin-based design



Why use SCIP?

SCIP: Solving Constraint Integer Programs . . .

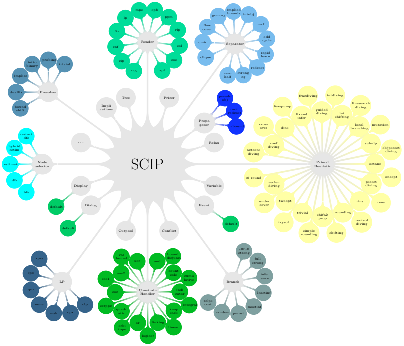
- provides a full-scale MIP and MINLP solver
- can solve general CIPs:
  constraint-based design
- can be extended:
  plugin-based design



Why use SCIP?

SCIP: Solving Constraint Integer Programs . . .

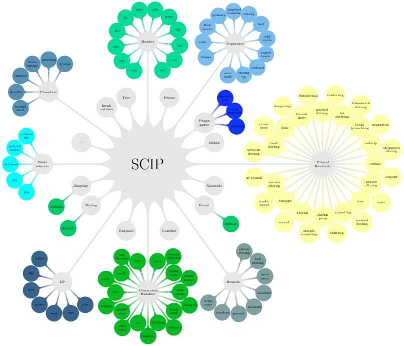
- provides a full-scale MIP and MINLP solver
- can solve general CIPs:
  constraint-based design
- can be extended:
  plugin-based design
- supports column generation
  and branch-cut-and-price



Why use SCIP?

SCIP: Solving Constraint Integer Programs . . .

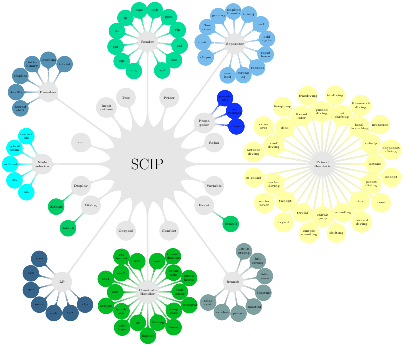
- provides a full-scale MIP and MINLP solver
- can solve general CIPs:
  constraint-based design
- can be extended:
  plugin-based design
- supports column generation
  and branch-cut-and-price
- is free for academic research



Why use SCIP?

SCIP: Solving Constraint Integer Programs ...

- ▷ provides a full-scale MIP and MINLP solver
- ▷ can solve general CIPs:
  constraint-based design
- ▷ can be extended:
  plugin-based design
- ▷ supports column generation
  and branch-cut-and-price
- ▷ is free for academic research
- ▷ is open: available in source code



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

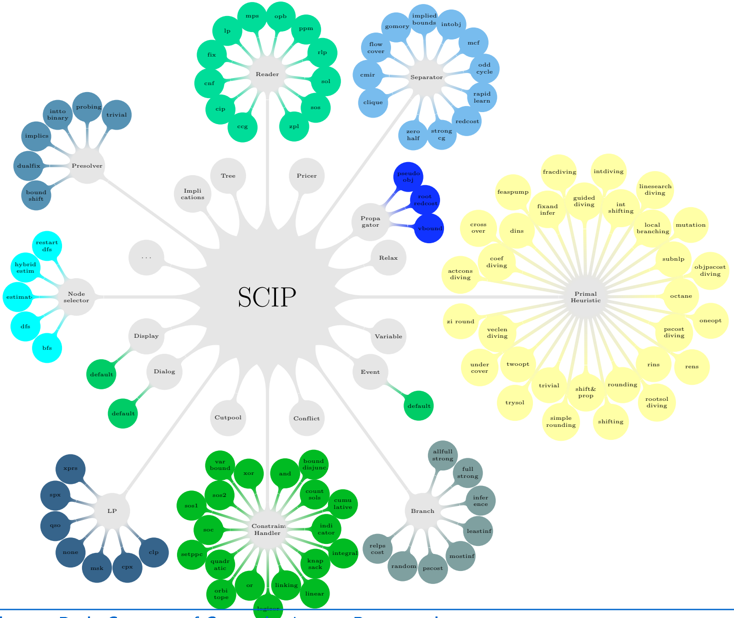
$\Rightarrow$ plugins are black boxes for the SCIP core

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

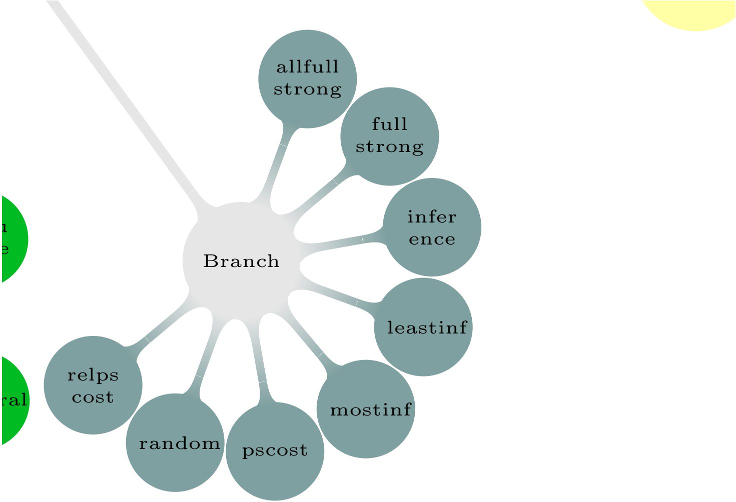
Types of Plugins

- 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

Outline

SCIP – Solving Constraint Integer Programs

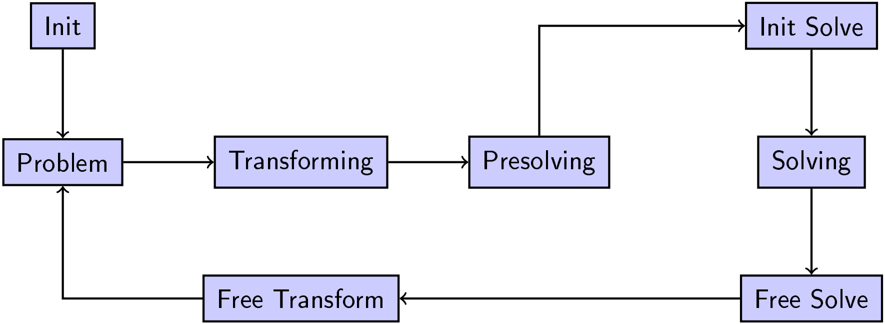
4 methodologies in optimization

An integrated method

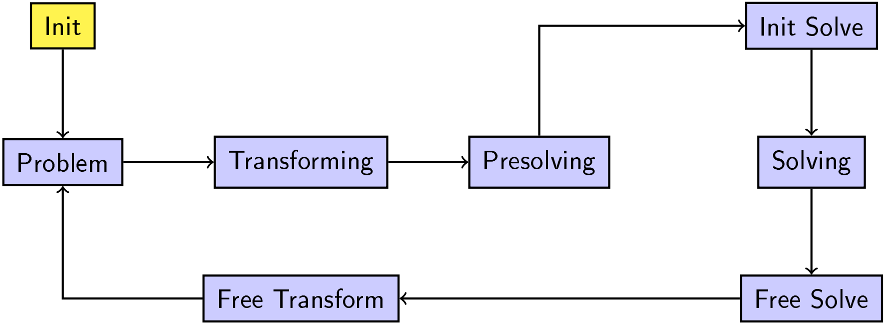
SCIP: Solving Constraint Integer Programs

The Solving Process of SCIP

Operational Stages

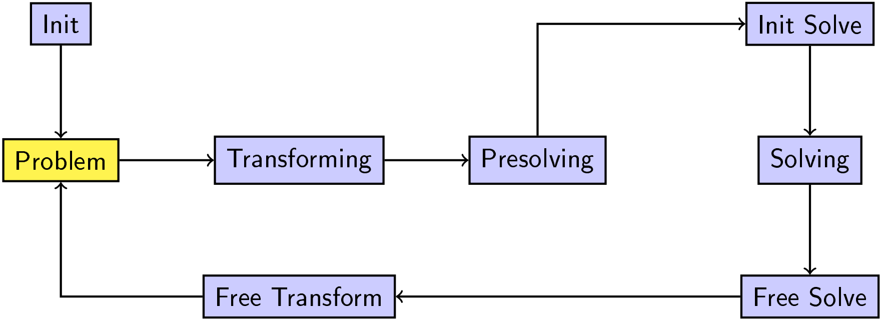


Operational Stages



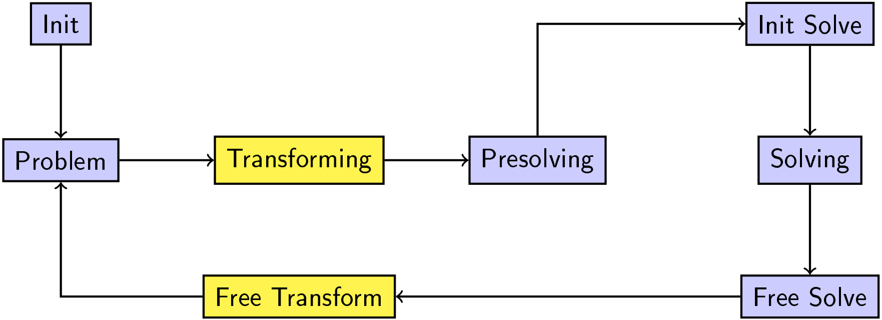
- Basic data structures are allocated and initialized.
- User includes required plugins (or just takes default plugins).

Problem Specification



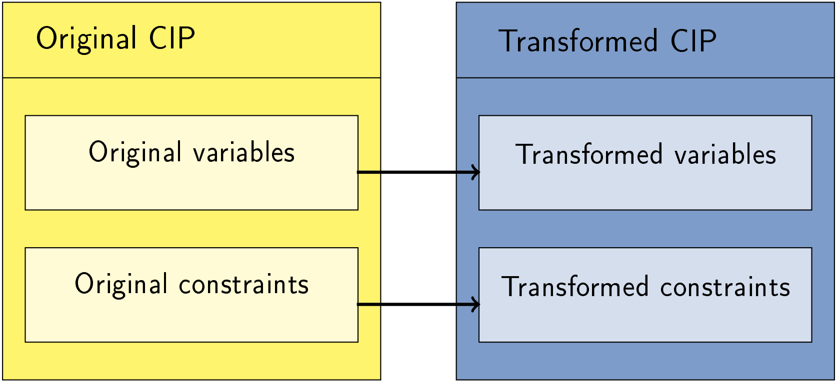
- User creates and modifies the original problem instance.
- Problem creation is usually done in file readers.

Transformation



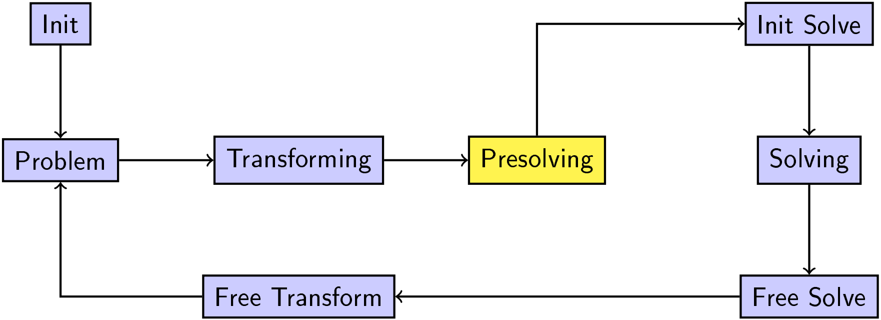
- Creates a working copy of the original problem.

Original and Transformed Problem

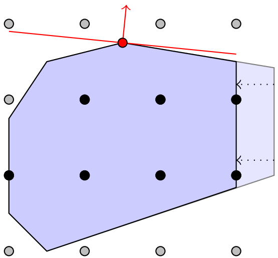


- data is copied into separate memory area
- presolving and solving operate on transformed problem
- original data can only be modified in problem modification stage

Presolving


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

Presolving



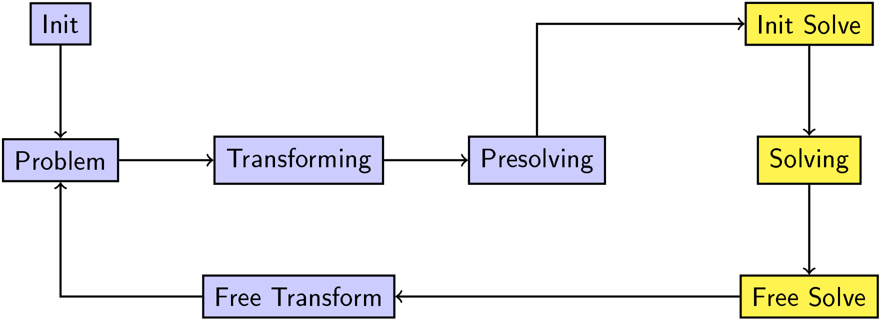
Task

- 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

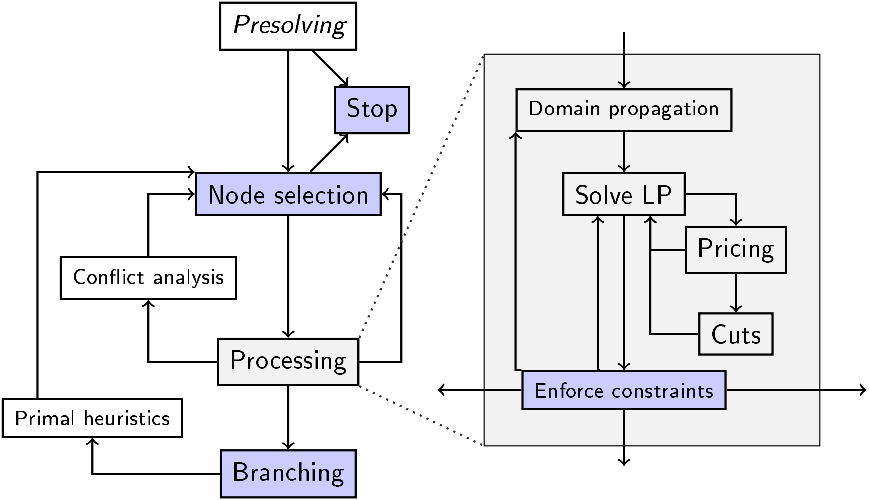
Primal Reductions:
- based on feasibility reasoning
- no feasible solution is cut off

Dual Reductions:
- consider objective function
- at least one optimal solution remains

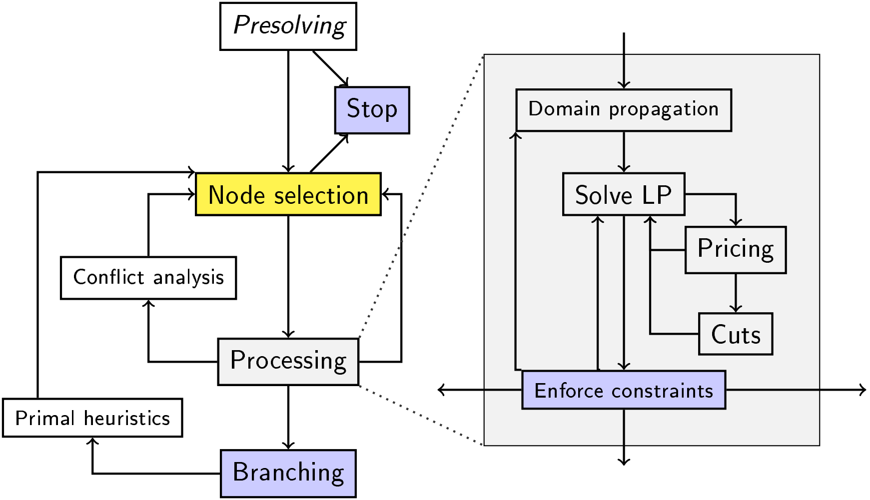
Solving


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

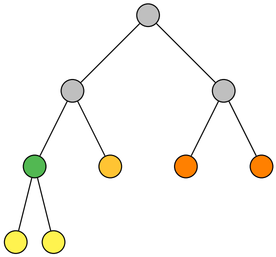
Flow Chart SCIP



Flow Chart SCIP



Node Selection



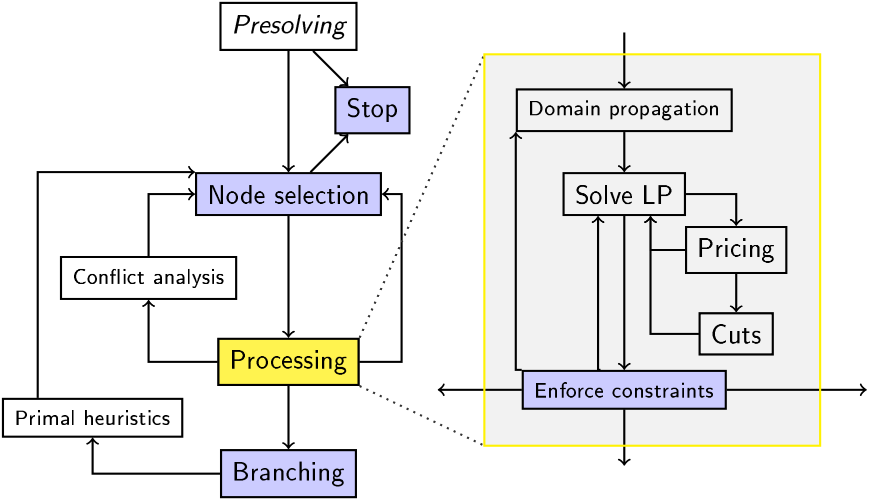
Task

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

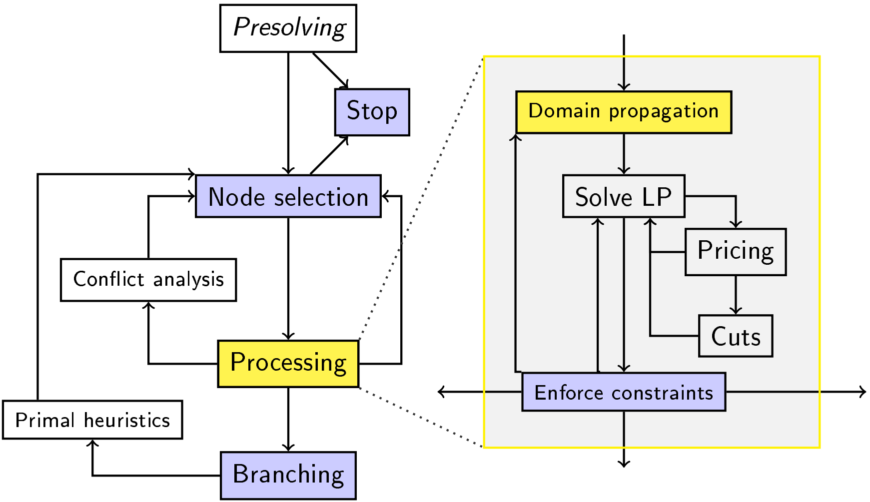
Techniques

- 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

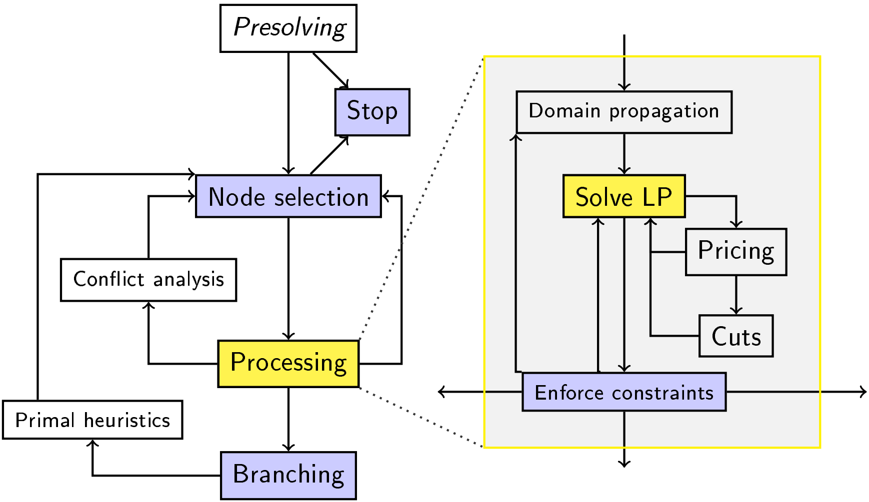
Flow Chart SCIP



Flow Chart SCIP

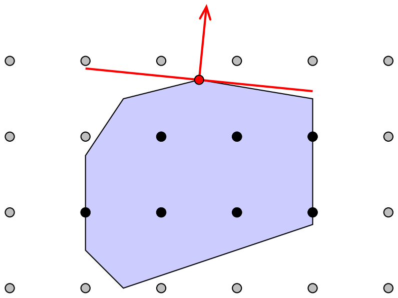


Flow Chart SCIP



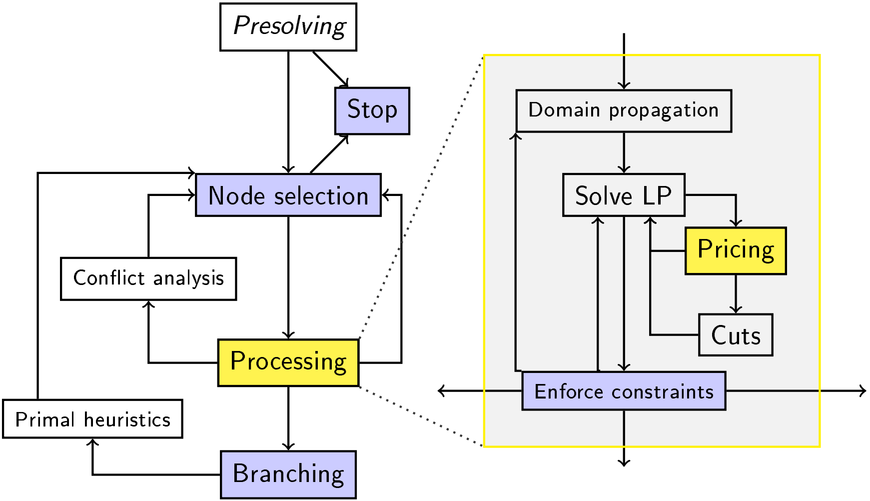
LP Solving

- LP solver is a black box
- interface to different LP solvers: SoPlex, CPLEX, XPress, Gurobi, CLP, ...
- primal/dual simplex
- barrier with/without crossover

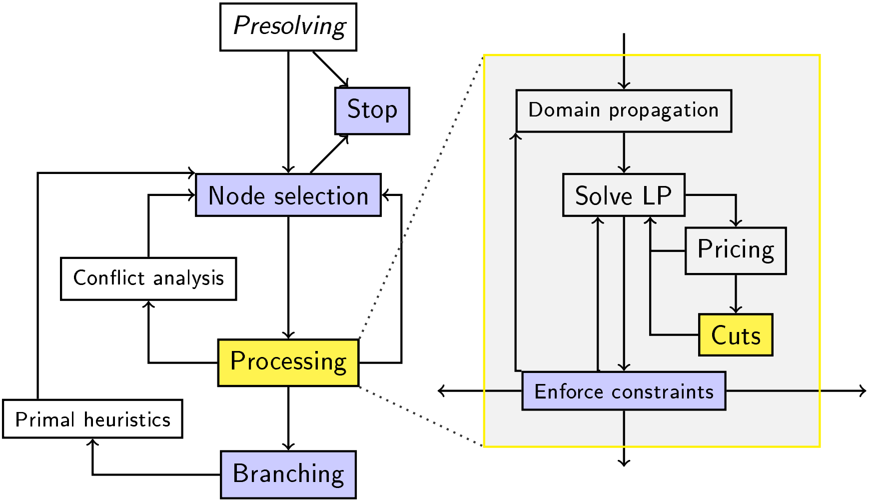


- double-check feasibility
- check condition number
- address numerical troubles by changing parameters: scaling, tolerances, solving from scratch, other simplex

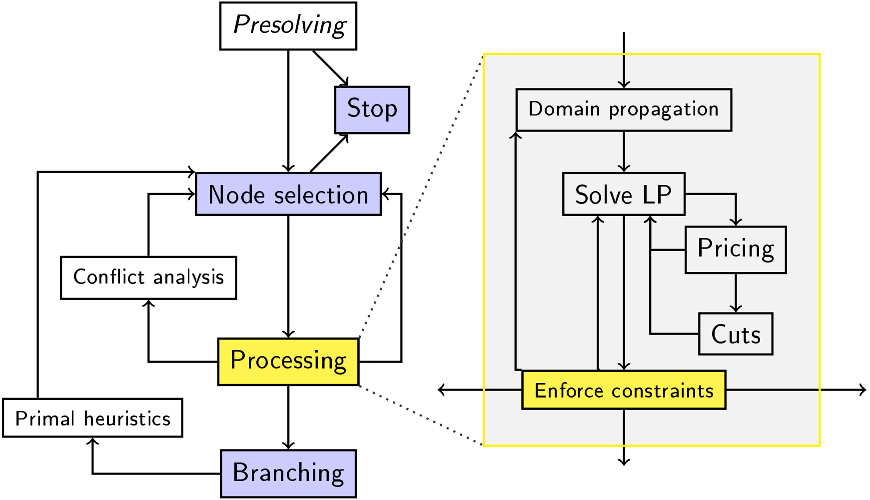
Flow Chart SCIP



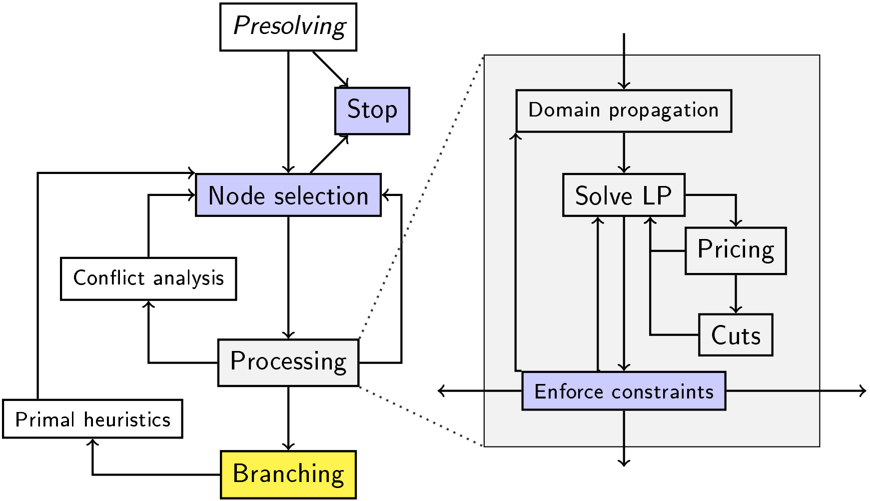
Flow Chart SCIP



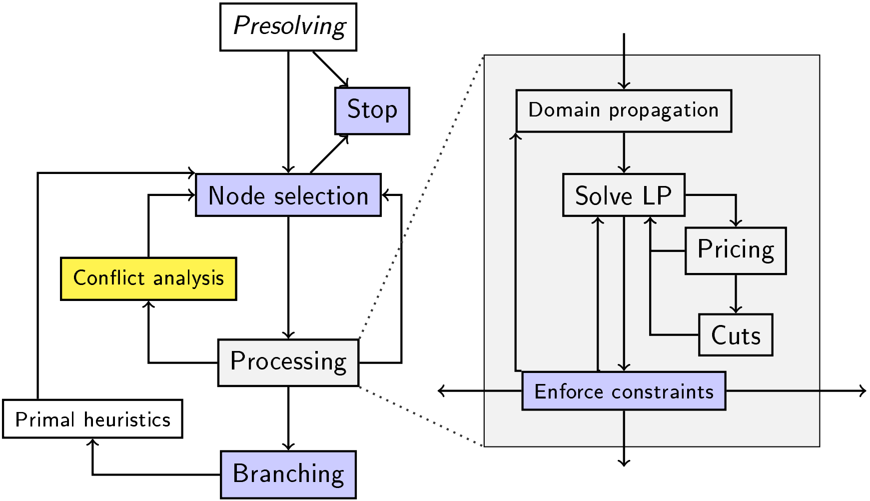
Flow Chart SCIP



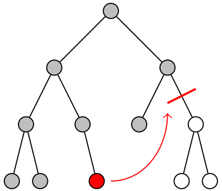
Flow Chart SCIP



Flow Chart SCIP



Conflict Analysis



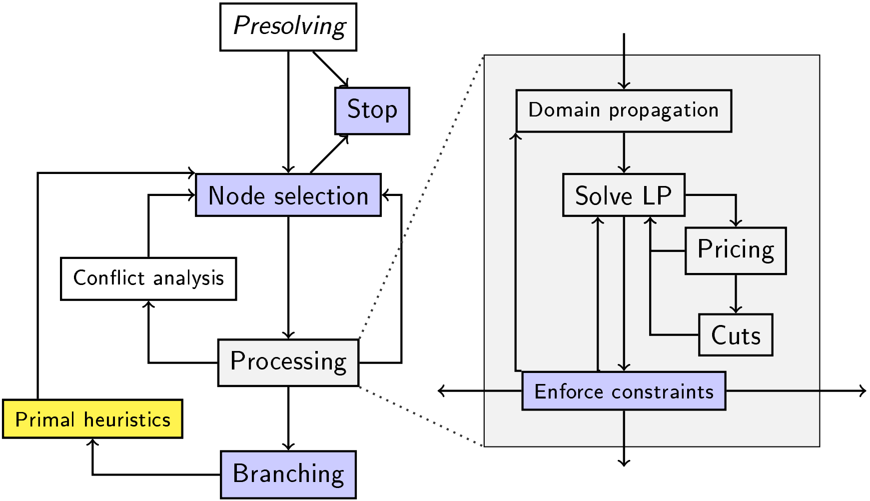
Task

- 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

Flow Chart SCIP



Summary

Take-away messages

- optimization paradigms: CP, SAT, MIP, MINLP
- CIP: an algorithmic, solution-driven integration
- SCIP: a flexible tool for computational research in optimization

Next

The most powerful plugin: constraint handlers