The SCIP Optimization Suite 10

https://scipopt.org

Mathieu Besançon, Suresh Bolusani, Antonia Chmiela, João Dionísio, Johannes Ehls, Mohammed Ghannam, Ambros Gleixner, Adrian Göß, Alexander Hoen, Christopher Hojny, Jacob von Holly-Ponientzietz, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Jurgen Lentz, Stephen J. Maher, Julian Manns, Paul M. Meinhold, Gioni Mexi, Til Mohr, Erik Mühmer, Krunal K. Patel, Marc E. Pfetsch, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Matthias Walter, Dieter Weninger, Liding Xu

EURO 2025 · Leeds, UK · June 23, 2025

Outline

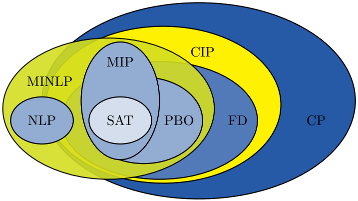
This talk has been split into two parts:

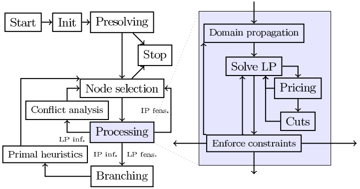
$$\underbrace{\textbf{The SCIP Optimization Suite}}_{\text{Part 1}} \quad \underbrace{\textbf{10}}_{\text{Part 2}}$$

The SCIP Optimization Suite

SCIP: Solving Constraint Integer Programs

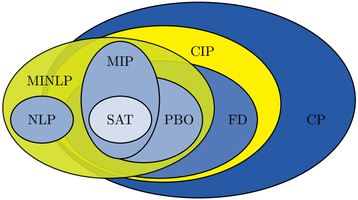
- a framework for constraint integer programming, incorporating features from
  - MILP (cutting planes, LP relaxation)
  - CP (domain propagation)
  - SAT (conflict analysis, restarts)
  - MINLP (spatial branch-and-bound, NLPs)
- a branch-cut-and-price framework
- includes full-scale solvers for MILP, MINLP, and Pseudo-Boolean optimization (→ WB-43)

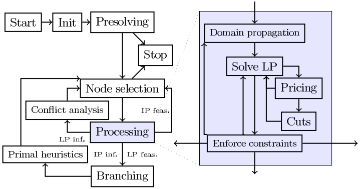




SCIP: Solving Constraint Integer Programs

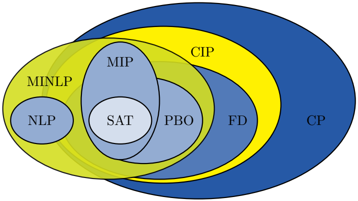
- a framework for constraint integer programming, incorporating features from
  - MILP (cutting planes, LP relaxation)
  - CP (domain propagation)
  - SAT (conflict analysis, restarts)
  - MINLP (spatial branch-and-bound, NLPs)
- a branch-cut-and-price framework
- includes full-scale solvers for MILP, MINLP, and Pseudo-Boolean optimization (→ WB-43)
- and much more: Benders decomp., exact MILP, IIS, MILP reoptimization, concurrent solving, cumulative and logical constraints, ...

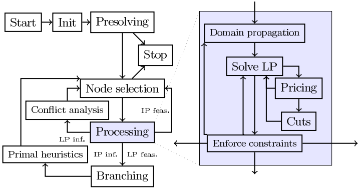




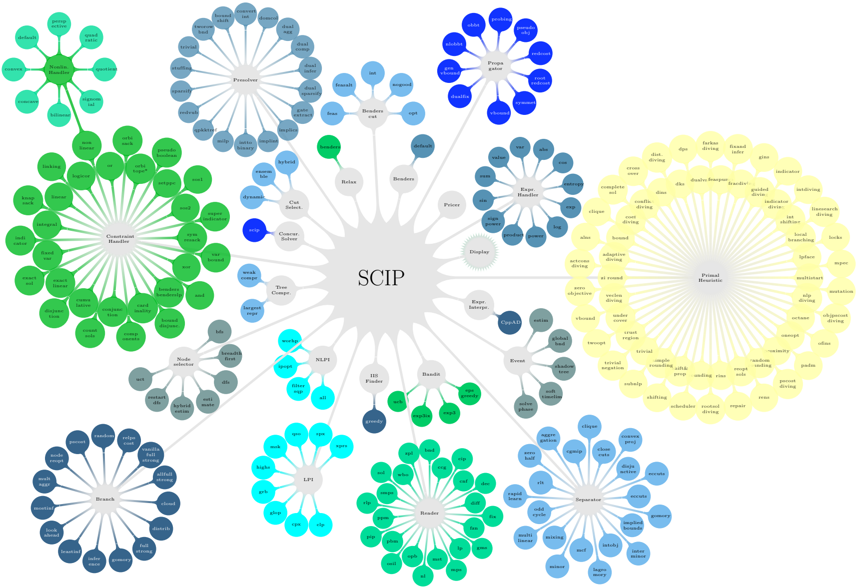
SCIP: Solving Constraint Integer Programs

- a framework for constraint integer programming, incorporating features from
  - MILP (cutting planes, LP relaxation)
  - CP (domain propagation)
  - SAT (conflict analysis, restarts)
  - MINLP (spatial branch-and-bound, NLPs)
- a branch-cut-and-price framework
- includes full-scale solvers for MILP, MINLP, and Pseudo-Boolean optimization (→ WB-43)
- and much more: Benders decomp., exact MILP, IIS, MILP reoptimization, concurrent solving, cumulative and logical constraints, . . .
- a platform for researchers to implement and test own methods in a general-purpose solver
  - plugin-based structure
  - basis for specialized extensions (GCG, SCIP-SDP, SCIP-Jack, QuBowl, . . .)
  - available open-source (Apache 2.0 license)
  - readable code



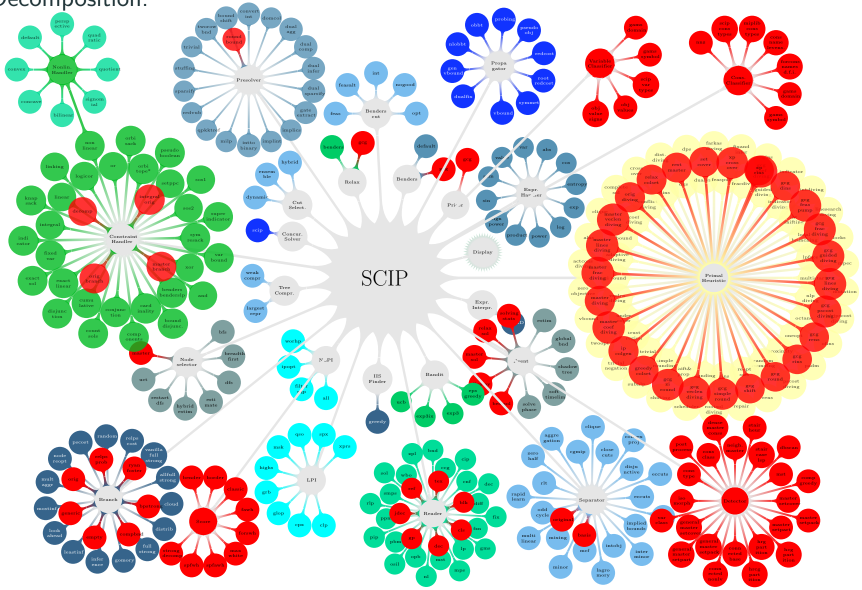


SCIP plugins



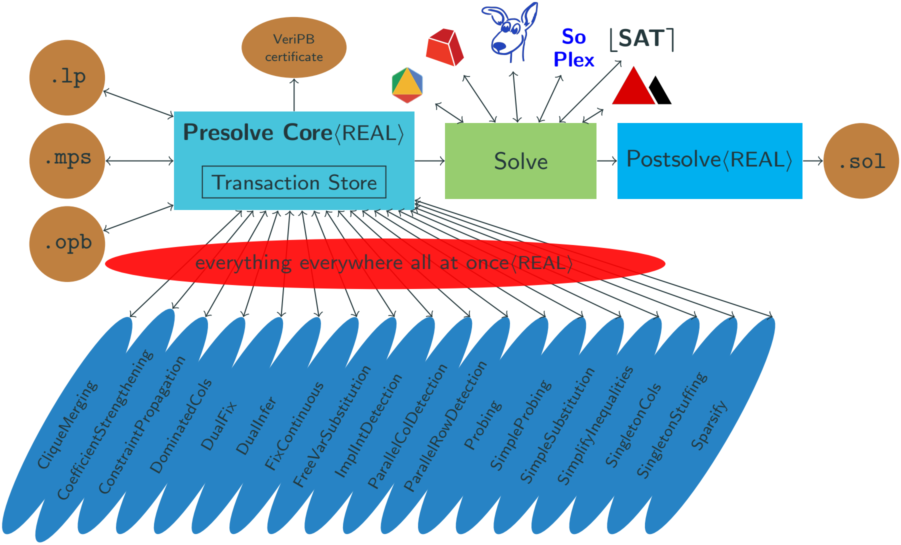
GCG: Generic Column Generation

Generic MILP solver with automatic structure detection, Dantzig-Wolfe, and Benders Decomposition.



PaPILO: Parallel Presolve for Integer and Linear Optimization

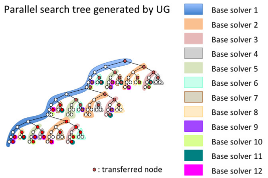
An independent library for presolving MILPs in parallel and with arbitrary precision.

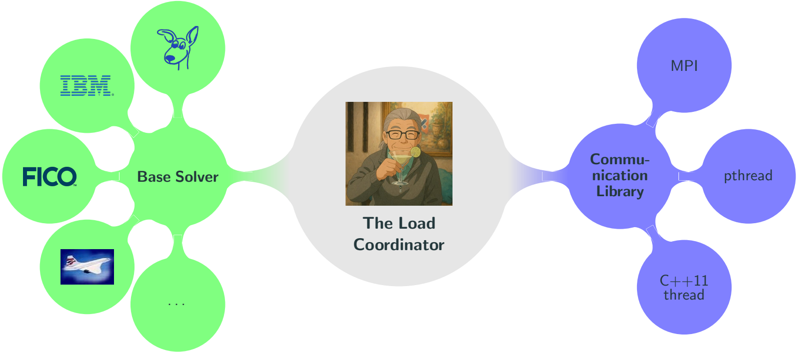


UG: Ubiquity Generator

Parallelization framework for solvers doing tree-search or other parallelizable tasks.

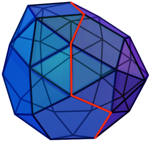
- distributed and shared memory environments
- normal and racing ramp-up, checkpointing
- more from Yuji in WB-43





SoPlex: Sequential object-oriented simPlex

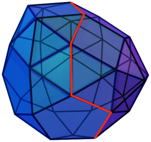
- Simplex LP solver
- high precision and exact solving
- iterative refinement



SoPlex, ZIMPL

SoPlex: Sequential object-oriented simPlex

- Simplex LP solver
- high precision and exact solving
- iterative refinement



ZIMPL: Zuse Institute Mathematical Programming Language

- Algebraic modeling language in the style of AMPL
- rational arithmetic

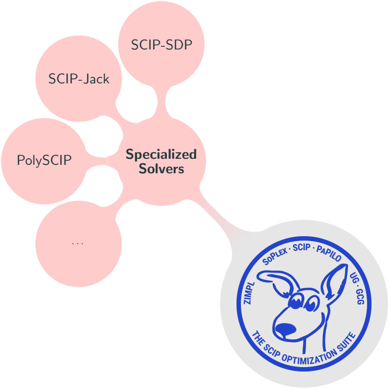
ZIB

SCIP Optimization Suite Ecosystem

https://github.com/scipopt

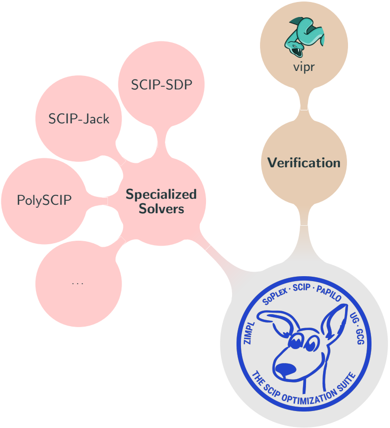
SCIP Optimization Suite Ecosystem

https://github.com/scipopt



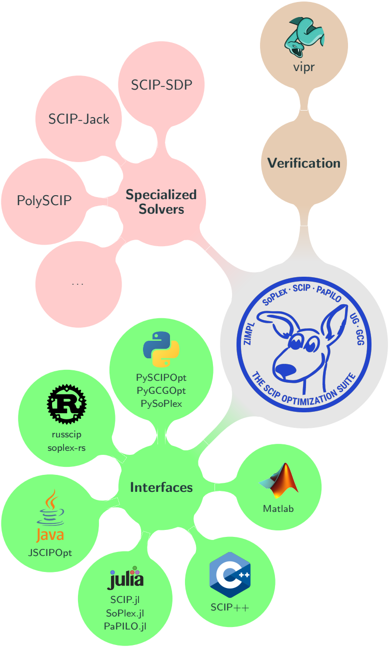
SCIP Optimization Suite Ecosystem

https://github.com/scipopt



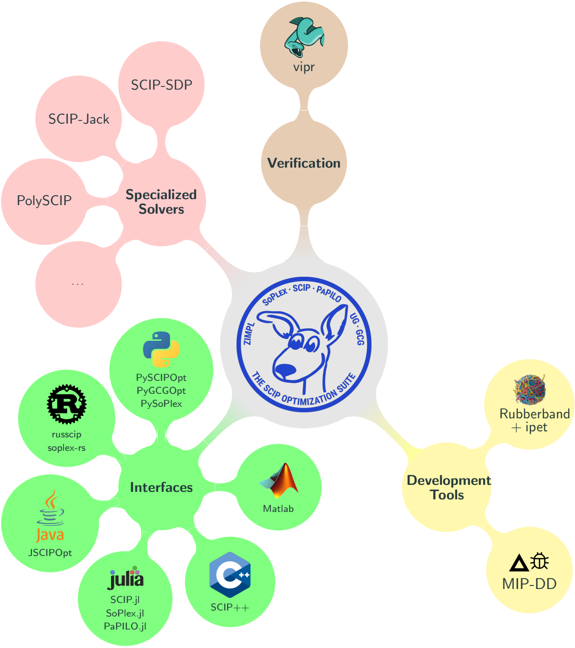
SCIP Optimization Suite Ecosystem

https://github.com/scipopt



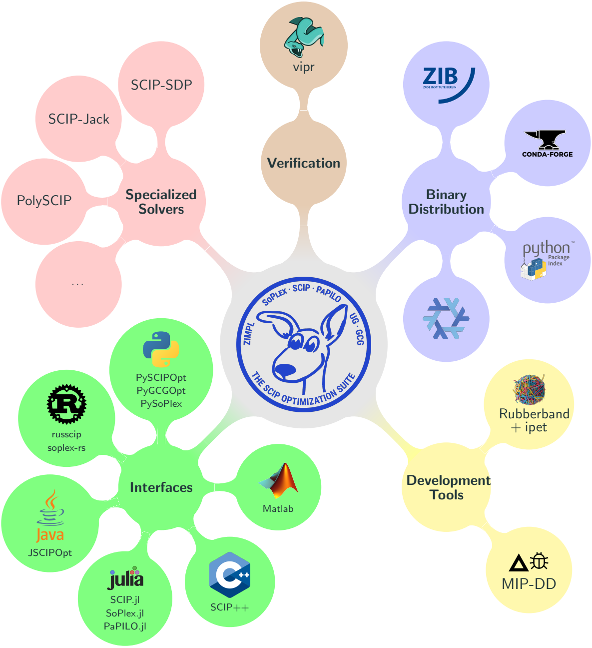
SCIP Optimization Suite Ecosystem

https://github.com/scipopt



SCIP Optimization Suite Ecosystem

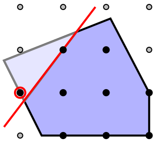
https://github.com/scipopt



10

SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost

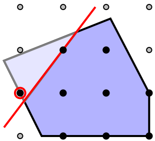


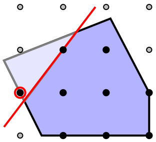
Exact MILP Solving

[Eifler, Gleixner 2023, 2024]

SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL



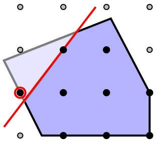


SCIP can now **solve mixed-integer linear programs over** $\mathbb{Q}$ **exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers

Exact MILP Solving

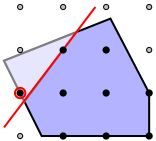
[Eifler, Gleixner 2023, 2024]



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO

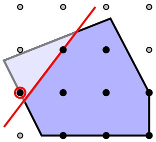
Exact MILP Solving [Eifler, Gleixner 2023, 2024]



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)

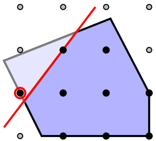
Exact MILP Solving



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding

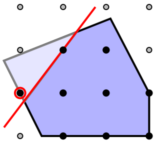
Exact MILP Solving [Eifler, Gleixner 2023, 2024]



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding

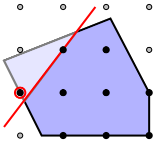
Exact MILP Solving [Eifler, Gleixner 2023, 2024]



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

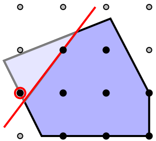
- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding
- Gomory mixed-integer cuts with safe rounding

Exact MILP Solving [Eifler, Gleixner 2023, 2024]



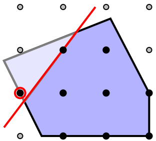
SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding
- Gomory mixed-integer cuts with safe rounding
- post-processing solutions from primal heuristics



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

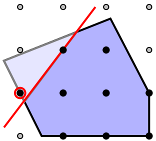
- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding
- Gomory mixed-integer cuts with safe rounding
- post-processing solutions from primal heuristics
- reliability pseudo-cost branching



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding
- Gomory mixed-integer cuts with safe rounding
- post-processing solutions from primal heuristics
- reliability pseudo-cost branching
- other separators disabled, symmetry handling disabled, etc

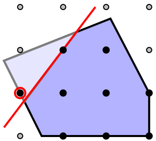
Exact MILP Solving [Eifler, Gleixner 2023, 2024]



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding
- Gomory mixed-integer cuts with safe rounding
- post-processing solutions from primal heuristics
- reliability pseudo-cost branching
- other separators disabled, symmetry handling disabled, etc
- log deductions in solving process (without presolve) for verification with VIPR

Exact MILP Solving [Eifler, Gleixner 2023, 2024]



SCIP can now **solve mixed-integer linear programs over $\mathbb{Q}$ exactly**: no rounding errors, zero tolerances

- relies on GMP, MPFR, and Boost
- exact reading of MPS, LP, CIP, OPB/WBO, and ZIMPL
- new constraint handler for linear constraints in rational numbers
- exact presolve in rational arithmetic via PaPILO
- exact LP relaxation, that can be solved with SoPlex or QSopt_ex
- cheaper numerically safe dual bounding techniques that post-process floating-point LP solves (bound shift, project-and-shift)
- LP infeasibility analysis (Farkas proof) with safe rounding
- domain propagation on linear constraints with safe rounding
- Gomory mixed-integer cuts with safe rounding
- post-processing solutions from primal heuristics
- reliability pseudo-cost branching
- other separators disabled, symmetry handling disabled, etc
- log deductions in solving process (without presolve) for verification with VIPR
- MIP-DD supports exact MILP, too

Exact MILP Solving: Performance

MIPLIB 2017 benchmark set, 3 random seeds, 2h time limit

Exact MILP Solving: Performance

MIPLIB 2017 benchmark set, 3 random seeds, 2h time limit

- default SCIP (floating-point) solves 342 instance+seed combinations

Exact MILP Solving: Performance

MIPLIB 2017 benchmark set, 3 random seeds, 2h time limit

- default SCIP (floating-point) solves 342 instance+seed combinations
- disable all features that are missing in exact SCIP $\rightarrow$ 235 solved

Exact MILP Solving: Performance

MIPLIB 2017 benchmark set, 3 random seeds, 2h time limit

- default SCIP (floating-point) solves 342 instance+seed combinations
- disable all features that are missing in exact SCIP $\rightarrow$ 235 solved
- exact SCIP solves 161

Exact MILP Solving: Performance

MIPLIB 2017 benchmark set, 3 random seeds, 2h time limit

- default SCIP (floating-point) solves 342 instance+seed combinations
- disable all features that are missing in exact SCIP → 235 solved
- exact SCIP solves 161

153 can be solved in any configuration; on these instance+seed:

- disabling SCIP features (floating-point mode) increases mean time by 89% and mean nodes by 161%

Exact MILP Solving: Performance

MIPLIB 2017 benchmark set, 3 random seeds, 2h time limit

- default SCIP (floating-point) solves 342 instance+seed combinations
- disable all features that are missing in exact SCIP → 235 solved
- exact SCIP solves 161

153 can be solved in any configuration; on these instance+seed:

- disabling SCIP features (floating-point mode) increases mean time by 89% and mean nodes by 161%
- switching to exact mode increases time by 258% and nodes by 155% (in addition)

Presolve: Implicit Integral Variables

A variable is implicit integral, if constraints (and objective) imply that it takes an integral value in any feasible (or any optimal, or at least one optimal) solution, e.g.,

- $x + y = 0$, $x \in \mathbb{Z} \quad \Rightarrow \quad y \in \mathbb{Z}$ if feasible
- $\max y$, s.t. $x + y \leq 0$, $x \in \mathbb{Z} \quad \Rightarrow \quad y \in \mathbb{Z}$ if optimal

Useful property for branching, cut strengthening, primal heuristics, domain propagation, ...

So far, SCIP's presolve could detect implicit integrality for only one variable at a time.

Presolve: Implicit Integrality via TU matrices [van der Hulst, Walter 2024, 2025]

Now, whole sets of implicit integral variables can be detected:

- partition variables into (x, y, z) such that constraints take form

$$\begin{aligned} Ax \quad + By \quad\quad\quad &\leq d \\ Ex \quad\quad\quad + Fz &\leq h \\ x \in \mathbb{Z}^n \end{aligned}$$

with A, d integral, B totally unimodular[1]

[1] every square submatrix has determinant -1, 0, or 1

Presole: Implicit Integrality via TU matrices [van der Hulst, Walter 2024, 2025]

Now, whole sets of implicit integral variables can be detected:

- partition variables into (x, y, z) such that constraints take form

$$\begin{aligned} A\,x \quad + B\,y \quad\quad\quad &\le d \\ E\,x \quad\quad\quad + F\,z &\le h \\ x \in \mathbb{Z}^n \end{aligned}$$

with A, d integral, B totally unimodular[1]

- for any fixing $x := \bar{x}$, the polyhedron $\{y : By \le d - A\,\bar{x}\}$ is integral

$\Rightarrow$ y is implicit integral

[1] every square submatrix has determinant -1, 0, or 1

Presolve: Implicit Integrality via TU matrices [van der Hulst, Walter 2024, 2025]

Now, whole sets of implicit integral variables can be detected:

- partition variables into (x, y, z) such that constraints take form

$$\begin{array}{lllll} A\,x & +B\,y & & \leq d & \text{with } A, d \text{ integral, } B \text{ totally unimodular}^{1} \\ E\,x & & +F\,z & \leq h & \\ x \in \mathbb{Z}^n & & & & \end{array}$$

- for any fixing $x := \bar{x}$, the polyhedron $\{y : By \leq d - A\,\bar{x}\}$ is integral

$\Rightarrow$ y is implicit integral

- SCIP very fast detects (transposed) *network matrices* B, a large sub-class of totally unimodular matrices

---

[1] every square submatrix has determinant -1, 0, or 1

Presolve: Implicit Integrality via TU matrices [van der Hulst, Walter 2024, 2025]

Now, whole sets of implicit integral variables can be detected:

- partition variables into (x, y, z) such that constraints take form

$$\begin{aligned} A\,x \quad + B\,y \qquad\qquad & \le d \quad \text{with } A, d \text{ integral, } B \text{ totally unimodular}^{1} \\ E\,x \qquad\qquad + F\,z & \le h \\ x \in \mathbb{Z}^n & \end{aligned}$$

- for any fixing $x := \bar{x}$, the polyhedron $\{y : By \le d - A\,\bar{x}\}$ is integral

$\Rightarrow$ y is implicit integral

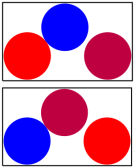
- SCIP very fast detects (transposed) *network matrices* B, a large sub-class of totally unimodular matrices
- implicit integrality now detected on 69% of MIPLIB2017 instances (SCIP 9: 20%)
- mean fraction of implicit integral variables increased from 3% to 19%

[1] every square submatrix has determinant -1, 0, or 1

Presolve: PaPILO updates

- PaPILO now licensed under Apache 2.0
- added clique merging
- faster column domination presolve by topological compression of domination arc sets

SCIP: Reflection Symmetries [Hojny 2024]

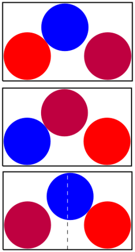


- SCIP can detect permutation symmetries, i.e., map $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ with permutation π on $\{1, \dots, n\}$ s.t.

$$\gamma(x) = (x_{\pi^{-1}(1)}, \dots, x_{\pi^{-1}(n)})$$

  and handle via SST cuts and orbitopal reduction (lex. order)

SCIP: Reflection Symmetries [Hojny 2024]



- SCIP can detect permutation symmetries, i.e., map $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ with permutation π on $\{1, \dots, n\}$ s.t.

$$\gamma(x) = (x_{\pi^{-1}(1)}, \dots, x_{\pi^{-1}(n)})$$

  and handle via SST cuts and orbitopal reduction (lex. order)

- now also reflection symmetries can be detected, i.e.,

$$\rho(x) = (s_1 x_{\pi^{-1}(1)}, \dots, s_n x_{\pi^{-1}(n)}) \qquad \text{for } s \in \{-1, 1\}^n$$

SCIP: Reflection Symmetries [Hojny 2024]

- SCIP can detect permutation symmetries, i.e., map $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ with permutation π on $\{1, \dots, n\}$ s.t.

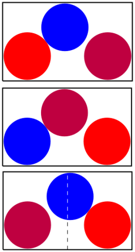
$$\gamma(x) = (x_{\pi^{-1}(1)}, \dots, x_{\pi^{-1}(n)})$$

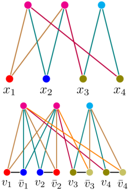
  and handle via SST cuts and orbitopal reduction (lex. order)

- now also reflection symmetries can be detected, i.e.,

$$\rho(x) = (s_1 x_{\pi^{-1}(1)}, \dots, s_n x_{\pi^{-1}(n)}) \qquad \text{for } s \in \{-1, 1\}^n$$

  - translate variable domain to be centered at origin
  - "duplicate" symmetry detection graph via negated variables and coefficients





SCIP: Reflection Symmetries [Hojny 2024]

- SCIP can detect permutation symmetries, i.e., map $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ with permutation π on $\{1, \ldots, n\}$ s.t.

$$\gamma(x) = (x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)})$$

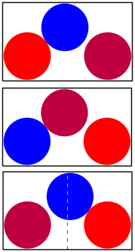
  and handle via SST cuts and orbitopal reduction (lex. order)

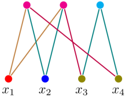
- now also reflection symmetries can be detected, i.e.,

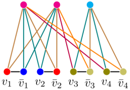
$$\rho(x) = (s_1 x_{\pi^{-1}(1)}, \ldots, s_n x_{\pi^{-1}(n)}) \qquad \text{for } s \in \{-1, 1\}^n$$

  - translate variable domain to be centered at origin
  - "duplicate" symmetry detection graph via negated variables and coefficients

- good performance improvements on testsets of geometric packing, kissing number, and energy minimization problems



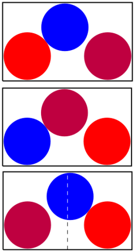




- MIPLIB2017: reflection symmetries on 6% of instances; solve more instances, but slowdown on average

- MINLPLib: only 6 instances, e.g., due to pre-existing symmetry breaking cons.

SCIP: Reflection Symmetries [Hojny 2024]



- SCIP can detect permutation symmetries, i.e., map $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ with permutation π on $\{1, \dots, n\}$ s.t.

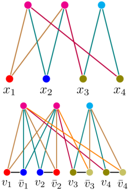
$$\gamma(x) = (x_{\pi^{-1}(1)}, \dots, x_{\pi^{-1}(n)})$$

  and handle via SST cuts and orbitopal reduction (lex. order)

- now also reflection symmetries can be detected, i.e.,

$$\rho(x) = (s_1 x_{\pi^{-1}(1)}, \dots, s_n x_{\pi^{-1}(n)}) \qquad \text{for } s \in \{-1, 1\}^n$$

  - translate variable domain to be centered at origin
  - “duplicate” symmetry detection graph via negated variables and coefficients



- good performance improvements on testsets of geometric packing, kissing number, and energy minimization problems
- MIPLIB2017: reflection symmetries on 6% of instances; solve more instances, but slowdown on average
- MINLPLib: only 6 instances, e.g., due to pre-existing symmetry breaking cons.
- symmetry detection now also for pseudo-boolean constraints

Separator: k-flower inequalities

Consider a binary product constraints

$$y_f = \prod_{v \in f} x_v, \qquad x_v \in \{0, 1\},$$

The standard relaxation includes the cut

$$y_f + \sum_{v \in f} (1 - x_v) \geq 1$$

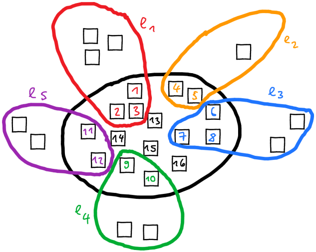
Separator: k-flower inequalities

Consider a binary product constraints

$$y_f = \prod_{v \in f} x_v, \qquad x_v \in \{0, 1\},$$

Assume k additional overlapping binary products

$$y_{e_i} = \prod_{v \in e_i} x_v, \qquad e_i \cap f \neq \emptyset, i = 1, \ldots, k.$$



The standard relaxation for f includes the cut

$$y_f + \sum_{v \in f} (1 - x_v) \geq 1$$

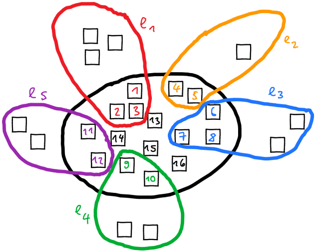
Separator: k-flower inequalities

Consider a binary product constraints

$$y_f = \prod_{v \in f} x_v, \qquad x_v \in \{0,1\},$$

Assume k additional overlapping binary products

$$y_{e_i} = \prod_{v \in e_i} x_v, \qquad e_i \cap f \neq \emptyset, i = 1, \ldots, k.$$



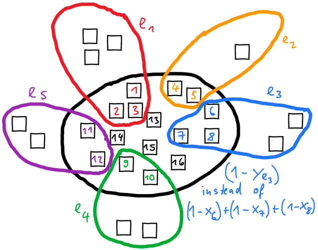
Rewrite the standard cut for f as

$$y_f + \sum_{i=1}^{k} \sum_{v \in f \cap e_i} (1 - x_v) + \sum_{v \in f \setminus \cup_{i=1}^{k} f_i} (1 - x_v) \geq 1$$

Separator: k-flower inequalities

Consider a binary product constraints

$$y_f = \prod_{v \in f} x_v, \qquad x_v \in \{0,1\},$$



Assume k additional overlapping binary products

$$y_{e_i} = \prod_{v \in e_i} x_v, \qquad e_i \cap f \neq \emptyset, i = 1, \ldots, k.$$

Rewrite the standard cut for f as

$$y_f + \sum_{i=1}^{k} \underbrace{\sum_{v \in f \cap e_i} (1 - x_v)}_{\text{replace by } 1 - y_{e_i}} + \sum_{v \in f \setminus \cup_{i=1}^{k} f_i} (1 - x_v) \geq 1$$

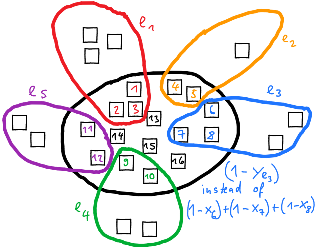
Separator: k-flower inequalities

Consider a binary product constraints

$$y_f = \prod_{v \in f} x_v, \qquad x_v \in \{0, 1\},$$

Assume k additional overlapping binary products

$$y_{e_i} = \prod_{v \in e_i} x_v, \qquad e_i \cap f \neq \emptyset, i = 1, \ldots, k.$$



The *k-flower inequality* is

$$y_f + \sum_{i=1}^{k} (1 - y_{e_i}) + \sum_{v \in f \setminus \cup_{i=1}^{k} f_i} (1 - x_v) \geq 1$$

SCIP separates these inequalities efficiently for $k = 1$ and $k = 2$.

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

- Destroy-Repair:
  - consider only columns active in current solution
  - remove some of the these columns
  - generate new columns to regain a feasible solution

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

- Destroy-Repair:
  - consider only columns active in current solution
  - remove some of the these columns
  - generate new columns to regain a feasible solution
- Specialized pricing scheme to repair feasibility:

$$\min_{(c,a)\in\mathcal{A}_q} c + \sum_{i\in I^p} \bar{u}_i a_i - \sum_{i\in I^c} \bar{u}_i a_i \quad \text{original pricing objective}$$

(partition = packing + covering)

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

- Destroy-Repair:
  - consider only columns active in current solution
  - remove some of the these columns
  - generate new columns to regain a feasible solution
- Specialized pricing scheme to repair feasibility:

$$\min_{(c,a)\in\mathcal{A}_q} c + \sum_{i\in I^p} \bar{u}_i a_i - \sum_{i\in I^c} \bar{u}_i a_i \quad \text{original pricing objective}$$

$$+ \sum_{i\in \hat{I}^{p1}} M a_i \quad \text{packing cons. already filled by partial sol.}$$

static big-M penalty

(partition = packing + covering)

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

- Destroy-Repair:
  - consider only columns active in current solution
  - remove some of the these columns
  - generate new columns to regain a feasible solution
- Specialized pricing scheme to repair feasibility:

$$\min_{(c,a)\in\mathcal{A}_q} c + \sum_{i\in I^p} \bar{u}_i a_i - \sum_{i\in I^c} \bar{u}_i a_i$$

original pricing objective

$$+ \sum_{i\in \hat{I}^{p1}} M a_i$$

packing cons. already filled by partial sol.
static big-M penalty

$$+ \sum_{i\in \hat{I}^{p0}} \beta_i a_i$$

packing cons. not filled by partial sol.
increase β_i when column with $a_i = 1$ found

$$- \sum_{i\in \hat{I}^{c0}} \beta_i a_i$$

cover cons. not filled by partial sol.
increase β_i when column with $a_i = 1$ found

dynamic β penalties, initially zero

(partition = packing + covering)

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

- Destroy-Repair:
  - consider only columns active in current solution
  - remove some of the these columns
  - generate new columns to regain a feasible solution
- Specialized pricing scheme to repair feasibility:

$$\min_{(c,a)\in\mathcal{A}_q} c + \gamma \sum_{i\in I^p} \bar{u}_i a_i - \gamma \sum_{i \in I^c} \bar{u}_i a_i$$

original pricing objective, damped by $\gamma \in [0,1]$

$$+ \sum_{i \in \hat{I}^{p1}} M a_i$$

packing cons. already filled by partial sol.
static big-M penalty

$$+ \sum_{i \in \hat{I}^{p0}} \beta_i a_i$$

packing cons. not filled by partial sol.
increase β_i when column with $a_i = 1$ found

$$- \sum_{i \in \hat{I}^{c0}} \beta_i a_i$$

cover cons. not filled by partial sol.
increase β_i when column with $a_i = 1$ found

dynamic β penalties, initially zero

(partition = packing + covering)

GCG: New Primal Heuristic IPColGen [Maher, Rönnberg 2023]

Large-Neighborhood-Search heuristic for set packing/partitioning/covering problems:

- Destroy-Repair:
  - consider only columns active in current solution
  - remove some of the these columns
  - generate new columns to regain a feasible solution
- Specialized pricing scheme to repair feasibility:

$$\min_{(c,a)\in\mathcal{A}_q} \; c + \gamma \sum_{i\in I^p} \bar{u}_i a_i - \gamma \sum_{i\in I^c} \bar{u}_i a_i$$

original pricing objective, damped by $\gamma \in [0,1]$

$$+ \sum_{i\in \hat{I}^{p1}} M a_i$$

packing cons. already filled by partial sol.
static big-M penalty

$$+ \sum_{i\in \hat{I}^{p0}} \beta_i a_i$$

packing cons. not filled by partial sol.
increase β_i when column with $a_i = 1$ found

$$- \sum_{i\in \hat{I}^{c0}} \beta_i a_i$$

cover cons. not filled by partial sol.
increase β_i when column with $a_i = 1$ found

dynamic β penalties, initially zero (partition = packing + covering)

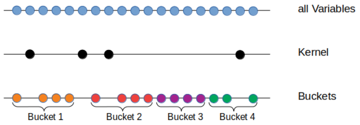
- on 160 suitable MIPLIB2017 instances: 5% improvement in gap between optimal value and primal bound

Based on Guastaroba, Savelsbergh, and Speranza (2017): Adaptive Kernel Search – A heuristic for solving Mixed Integer linear Programs

SCIP: New Primal Heuristic Kernel Search [Halbig, Göß, Weninger 2023]

Based on Guastaroba, Savelsbergh, and Speranza (2017): Adaptive Kernel Search – A heuristic for solving Mixed Integer linear Programs

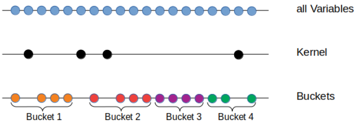
- find suitable/promising variables to define a Kernel, e.g., nonzero value in LP solution
- split remaining variables into buckets, e.g., by logarithm of reduced costs



SCIP: New Primal Heuristic Kernel Search [Halbig, Göß, Weninger 2023]

Based on Guastaroba, Savelsbergh, and Speranza (2017): Adaptive Kernel Search – A heuristic for solving Mixed Integer linear Programs

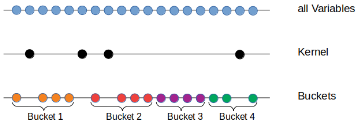
- find suitable/promising variables to define a Kernel, e.g., nonzero value in LP solution
- split remaining variables into buckets, e.g., by logarithm of reduced costs



- unite Kernel with each bucket to create several easier problems
- solve each “easy” problem after fixing all other variables
- update the Kernel after each solve by adding variables nonzero in last improving solution

Based on Guastaroba, Savelsbergh, and Speranza (2017): Adaptive Kernel Search – A heuristic for solving Mixed Integer linear Programs

- find suitable/promising variables to define a Kernel, e.g., nonzero value in LP solution
- split remaining variables into buckets, e.g., by logarithm of reduced costs

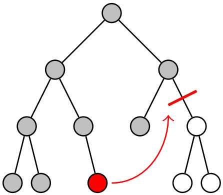


- unite Kernel with each bucket to create several easier problems
- solve each "easy" problem after fixing all other variables
- update the Kernel after each solve by adding variables nonzero in last improving solution
- additional adjustments if problem decomposition is available: ensure each bucket contains variables from all blocks

SCIP: Cut-based Conflict Analysis

[Mexi, Serrano, Berthold, Gleixner, Nordström 2024]

Encountering an infeasible branch-and-bound node, **conflict analysis** is about obtaining a constraint that would have identified the infeasibility earlier.

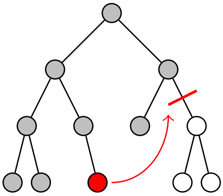


- so far, SCIP could derive bound disjunctions $\bigvee_i \{x_i \lesseqgtr b_i\}$ (SAT-based approach) or use Farkas proof from infeasible LP

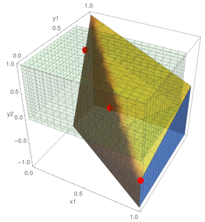
SCIP: Cut-based Conflict Analysis

[Mexi, Serrano, Berthold, Gleixner, Nordström 2024]

Encountering an infeasible branch-and-bound node, **conflict analysis** is about obtaining a constraint that would have identified the infeasibility earlier.



- so far, SCIP could derive bound disjunctions $\bigvee_i \{x_i \lesseqgtr b_i\}$ (SAT-based approach) or use Farkas proof from infeasible LP
- now, directly use the linear constraints that were responsible for the bound tightenings that lead to infeasibility
- a sequence of linear combinations, integer roundings, and MIR cut generation to derive a cut that separates the infeasible local domain
- more details in WB-43



Branching

SCIP: Probabilistic Lookahead Strong Branching

[Mexi, Shamsi, Besançon, le Bodic 2024]

- stop strong branching when expected tree size after evaluating one more candidate is not smaller than the current tree size

Branching

SCIP: Probabilistic Lookahead Strong Branching

[Mexi, Shamsi, Besançon, le Bodic 2024]

- stop strong branching when expected tree size after evaluating one more candidate is not smaller than the current tree size

SCIP: Ancestral Pseudocosts

- approximate longer-term influence of branching decisions
- include LP bound improvements from subsequent levels into pseudo-costs

Branching

SCIP: Probabilistic Lookahead Strong Branching

[Mexi, Shamsi, Besançon, le Bodic 2024]

- stop strong branching when expected tree size after evaluating one more candidate is not smaller than the current tree size

SCIP: Ancestral Pseudocosts

- approximate longer-term influence of branching decisions
- include LP bound improvements from subsequent levels into pseudo-costs

SCIP: Mix Integer and Nonlinear Branching

- allow to branch on variable in nonconvex term before integrality constraints are satisfied

Branching

SCIP: Probabilistic Lookahead Strong Branching

[Mexi, Shamsi, Besançon, le Bodic 2024]

- stop strong branching when expected tree size after evaluating one more candidate is not smaller than the current tree size

SCIP: Ancestral Pseudocosts

- approximate longer-term influence of branching decisions
- include LP bound improvements from subsequent levels into pseudo-costs

SCIP: Mix Integer and Nonlinear Branching

- allow to branch on variable in nonconvex term before integrality constraints are satisfied

GCG: Component Bound Branching

- branch entirely in reformulated problem, similar to Vanderbeck's generic branching scheme (2011), but less complex

Decomposition

More News from GCG:

- now licensed under Apache 2.0
- new JSON-based file format for decomposition: allows for nested decompositions and symmetry info
- new pricing solvers: GCG (nested decomp.) and HiGHS
- easier addition of new constraint to master problem ("extended master constraints")
- decomposition scores are now plugins

Decomposition

More News from GCG:

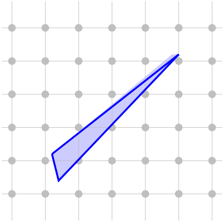
- now licensed under Apache 2.0
- new JSON-based file format for decomposition: allows for nested decompositions and symmetry info
- new pricing solvers: GCG (nested decomp.) and HiGHS
- easier addition of new constraint to master problem ("extended master constraints")
- decomposition scores are now plugins

Benders in SCIP:

- full solution can now be obtained if decomposition happens in SCIP
- allow for $\max_i \theta_i$ instead of $\sum_i \theta_i$ objective function
- distinguish master and linking variables

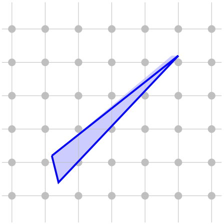
Irreducible Infeasible Subsystem

- a subset of the problem's constraints and variable bounds that cannot be satisfied jointly and that becomes feasible if reducing further



Irreducible Infeasible Subsystem

- a subset of the problem's constraints and variable bounds that cannot be satisfied jointly and that becomes feasible if reducing further
- SCIP and MIP-DD can now compute IIS by greedy algorithms, either building up from an empty problem or reducing from the full problem



Interfaces

PySCIPOpt:

- Matrix variables are now available, e.g.,

```
x = scip.addMatrixVar((2,2), vtype='C', name='x', ub=8)
scip.addMatrixCons(x + y <= z)
scip.addMatrixCons(x @ y <= x)
```

- built on NumPy, thus can use all standard NumPy ops (@, *, +, **, . . .)
- mix scalars, vectors, matrices with automatic NumPy broadcasting

Interfaces

PySCIPOpt:

- Matrix variables are now available, e.g.,

```
x = scip.addMatrixVar((2,2), vtype='C', name='x', ub=8)
scip.addMatrixCons(x + y <= z)
scip.addMatrixCons(x @ y <= x)
```

- built on NumPy, thus can use all standard NumPy ops (@, *, +, **, ...)
- mix scalars, vectors, matrices with automatic NumPy broadcasting

russcip:

- separator and constraint handler access
- new methods to add cuts

Interfaces

PySCIPOpt:

- Matrix variables are now available, e.g.,

```
x = scip.addMatrixVar((2,2), vtype='C', name='x', ub=8)
scip.addMatrixCons(x + y <= z)
scip.addMatrixCons(x @ y <= x)
```

- built on NumPy, thus can use all standard NumPy ops (`@`, `*`, `+`, `**`, ...)
- mix scalars, vectors, matrices with automatic NumPy broadcasting

russcip:

- separator and constraint handler access
- new methods to add cuts

SCIP.jl:

- event handler access
- MinUC computation

SCIP solving statistics

SCIP can prints hundreds of line of statistics on the solving process.

```
SCIP Status        : solving was interrupted [gap limit reached]
Total Time         :       0.64
  solving          :       0.63
  presolving       :       0.03 (included in solving)
  reading          :       0.01
  copying          :       0.01 (5 #copies) (minimal 0.00, maximal 0.00, average 0.00)
Original Problem   :
  Problem name     : BELL5
  Variables        : 104 (30 binary, 28 integer, 46 continuous)
  implied integral : 0 (0 binary, 0 integer, 0 continuous)
  Constraints      : 91 initial, 91 maximal
  Objective        : minimize, 74 non-zeros (abs.min = 0.1825, abs.max = 60000)
Presolved Problem  :
  Problem name     : t_BELL5
  Variables        : 30 (2 binary, 11 integer, 17 continuous)
  implied integral : 0 (0 binary, 0 integer, 0 continuous)
  Constraints      : 61 initial, 62 maximal
  Objective        : minimize, 28 non-zeros (abs.min = 1.41693, abs.max = 59000)
  Nonzeros         : 553 constraint, 0 clique table
Presolvers         :   ExecTime  SetupTime  Calls  FixedVars   AggrVars   ChgTypes  ChgBounds   AddHoles    DelCons    AddCons   Chg
  boundshift       :       0.00       0.00      0          0          0          0          0          0          0          0
  convertinttobin  :       0.00       0.00      0          0          0          0          0          0          0          0
  domcol           :       0.00       0.00      5          1          0          0          0          0          0          0
  dualagg          :       0.00       0.00      0          0          0          0          0          0          0          0
  dualcomp         :       0.00       0.00      5          0          0          0          0          0          0          0
  dualinfer        :       0.00       0.00      0          0          0          0          0          0          0          0
  dualsparsify     :       0.00       0.00      1          0          0          0          0          0          0          0
  gateextraction   :       0.00       0.00      0          0          0          0          0          0          0          0
  implics          :       0.00       0.00     12          0          0          0          0          0          0          0
  implint          :       0.00       0.00      0          0          0          0          0          0          0          0
  inttobinary      :       0.00       0.00     49          0          2          2          0          0          0          0
  milp             :       0.00       0.00      4          4          2          0         15          0          0          0
```

SCIP solving statistics

SCIP can prints hundreds of line of statistics on the solving process.

```
SCIP Status        : solving was interrupted [gap limit reached]
Total Time         :       0.64
  solving          :       0.63
  presolving
  reading
  copying
Original Problem
  Problem name
  Variables
  implied integral
  Constraints
  Objective
Presolved Problem
  Problem name
  Variables
  implied integral
  Constraints
  Objective
  Nonzeros
Presolvers                                      elCons    AddCons    Chg
  boundshift                                         0          0
  convertinttobin                                    0          0
  domcol                                             0          0
  dualagg                                            0          0
  dualcomp                                           0          0
  dualinfer                                          0          0
  dualsparsify                                       0          0
  gateextraction                                     0          0
  implics                                            0          0
  implint                                            0          0
  inttobinary                                        0          0
  milp                                               0          0
```

Now this information is available via an API and JSON.


```
"origprob" : {
   "description" : "original problem statistics table",
   "num_binary_variables" : 30,
   "num_continuous_variables" : 46,
   "num_implied_binary_variables" : 0,
   "num_implied_continuous_variables" : 0,
   "num_implied_integer_variables" : 0,
   "num_initial_constraints" : 91,
   "num_integer_variables" : 28,
   "num_maximal_constraints" : 91,
   "num_variables" : 104,
   "objective_abs_max" : 60000,
   "objective_abs_min" : 0.1825,
   "objective_non_zeros" : 74,
   "objective_sense" : "minimize",
   "problem_name" : "BELL5"
},
"presolvedprob" : {
   "clique_table_nonzeros" : 0,
   "constraint_nonzeros" : 539,
   "description" : "presolved problem statistics table",
   "num_binary_variables" : 2,
   "num_continuous_variables" : 17,
   "num_implied_binary_variables" : 0,
   "num_implied_continuous_variables" : 0,
   "num_implied_integer_variables" : 0,
   "num_initial_constraints" : 60,
```

Availability

- SCIP Optimization Suite 10 should be released in the next months.
- Everything is already publicly available in the development branches (master or develop) on `https://github.com/scipopt`.

Availability

- SCIP Optimization Suite 10 should be released in the next months.
- Everything is already publicly available in the development branches (master or develop) on `https://github.com/scipopt`.
- A release report with many details will be available again.

The SCIP Optimization Suite 10.0

Christopher Hojny · Mathieu Besançon · Suresh Bolusani ·
Antonia Chmiela · João Dionísio · Johannes Ehls ·
Mohammed Ghannam · Ambros Gleixner · Adrian Göß ·
Alexander Hoen · Jacob von Holly-Ponientzietz · Rolf van der Hulst ·
Dominik Kamp · Thorsten Koch · Jurgen Lentz ·
Stephen J. Maher · Julian Manns · Paul Matti Meinhold ·
Gioni Mexi · Til Mohr · Erik Mühmer ·
Krunal Kishor Patel · Marc E. Pfetsch · Felipe Serrano ·
Yuji Shinano · Mark Turner · Stefan Vigerske ·
Matthias Walter · Dieter Weninger · Liding Xu *

June 22, 2025

Abstract

Keywords Constraint integer programming · linear programming · mixed-integer linear programming · mixed-integer nonlinear programming · optimization solver · branch-and-cut · branch-and-price · column generation · parallelization · mixed-integer semidefinite programming

Mathematics Subject Classification 90C05 · 90C10 · 90C11 · 90C30 · 90C90 · 65Y05