# The SCIP Optimization Suite 9.0

Suresh Bolusani · Mathieu Besançon · Ksenia Bestuzheva
Antonia Chmiela · João Dionísio · Tim Donkiewicz
Jasper van Doornmalen · Leon Eifler · Mohammed Ghannam
Ambros Gleixner · Christoph Graczyk · Katrin Halbig
Ivo Hedtke · Alexander Hoen · Christopher Hojny
Rolf van der Hulst · Dominik Kamp · Thorsten Koch
Kevin Kofler · Jurgen Lentz · Julian Manns · Gioni Mexi
Erik Mühmer · Marc E. Pfetsch · Franziska Schlösser
Felipe Serrano · Yuji Shinano · Mark Turner
Stefan Vigerske · Dieter Weninger · Liding Xu *

26 February 2024

**Abstract** The SCIP Optimization Suite provides a collection of software packages for mathematical optimization, centered around the constraint integer programming (CIP) framework SCIP. This report discusses the enhancements and extensions included in SCIP Optimization Suite 9.0. The updates in SCIP 9.0 include improved symmetry handling, additions and improvements of nonlinear handlers and primal heuristics, a new cut generator and two new cut selection schemes, a new branching rule, a new LP interface, and several bugfixes. SCIP Optimization Suite 9.0 also features new Rust and C++ interfaces for SCIP, new Python interface for SoPlex, along with enhancements to existing interfaces. SCIP Optimization Suite 9.0 also includes new and improved features in the LP solver SoPlex, the presolving library PaPILO, the parallel framework UG, the decomposition framework GCG, and the SCIP extension SCIP-SDP. These additions and enhancements have resulted in an overall performance improvement of SCIP in terms of solving time, number of nodes in the branch-and-bound tree, as well as the reliability of the solver.

**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

*Extended author information is available at the end of the paper. The work for this article has been partly conducted within the *Research Campus MODAL* funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM) and has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 773897. It has also been partly supported by the German Research Foundation (DFG) within the Collaborative Research Center 805, Project A4, and the EXPRESS project of the priority program CoSIP (DFG-SPP 1798), the German Research Foundation (DFG) within the project HPO-NAVI (project number 391087700).

## 1 Introduction

The SCIP Optimization Suite comprises a set of complementary software packages designed to model and solve a large variety of mathematical optimization problems:

- the constraint integer programming solver SCIP [3], a solver for mixed-integer linear and nonlinear programs as well as a flexible framework for branch-cut-and-price,
- the simplex-based linear programming solver SoPlex [99],
- the modeling language Zimpl [60],
- the presolving library PaPILO for linear and mixed-integer linear programs,
- the automatic decomposition solver GCG [31], and
- the UG framework for parallelization of branch-and-bound solvers [84].

All six tools are freely available as open-source software packages, either using the Apache 2.0 or the GNU Lesser General Public License. There also exist two notable continuously developed extensions to the SCIP Optimization Suite: the award-winning Steiner tree solver SCIP-Jack [32] and the mixed-integer semidefinite programming solver SCIP-SDP [30]. This report describes the improvements and new features contained in version 9.0 of the SCIP Optimization Suite.

*Background* SCIP is designed as a solver for *constraint integer programs* (CIPs), a generalization of mixed-integer linear and nonlinear programs (MILPs and MINLPs). CIPs are finite-dimensional optimization problems with arbitrary constraints and a linear objective function that satisfy the following property: if all integer variables are fixed, the remaining subproblem must form a linear or nonlinear program (LP or NLP). To solve CIPs, SCIP constructs relaxations—typically linear relaxations, but also nonlinear relaxations are possible, or relaxations based on semidefinite programming for SCIP-SDP. If the relaxation solution is not feasible for the current subproblem, an *enforcement* procedure is called that takes measures to resolve the infeasibility, for example by branching or by separating cutting planes.

The most important subclass of CIPs that are solvable with SCIP are *mixed-integer programs* (MIPs) which can be purely linear (MILPs) or contain nonlinearities (MINLPs). MILPs are optimization problems of the form

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq b, \\ & \ell_i \leq x_i \leq u_i \qquad \text{for all } i \in \mathcal{N}, \\ & x_i \in \mathbb{Z} \qquad\quad \text{for all } i \in \mathcal{I}, \end{aligned} \tag{1}$$

defined by $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $\ell$, $u \in \overline{\mathbb{R}}^n$, and the index set of integer variables $\mathcal{I} \subseteq \mathcal{N} \coloneqq \{1, \ldots, n\}$. The usage of $\overline{\mathbb{R}} \coloneqq \mathbb{R} \cup \{-\infty, \infty\}$ allows for variables that are free or bounded only in one direction (we assume that variables are not fixed to $\pm\infty$). In contrast, MINLPs are optimization problems of the form

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_k(x) \leq 0 \qquad \text{for all } k \in \mathcal{M}, \\ & \ell_i \leq x_i \leq u_i \qquad \text{for all } i \in \mathcal{N}, \\ & x_i \in \mathbb{Z} \qquad\quad \text{for all } i \in \mathcal{I}, \end{aligned} \tag{2}$$

where the functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g_k : \mathbb{R}^n \to \mathbb{R}$, $k \in \mathcal{M} \coloneqq \{1, \ldots, m\}$, are possibly nonconvex. Within SCIP, we assume that $f$ is linear and that $g_k$ are specified explicitly in algebraic form using a known set of base expressions.

Due to its design as a solver for CIPs, SCIP can be extended by plugins for more general or problem-specific classes of optimization problems. The core of SCIP is formed by a central branch-cut-and-price algorithm that utilizes an LP as the default relaxation which can be solved by a number of different LP solvers, controlled through a uniform *LP interface*. To be able to handle any type of constraint, a *constraint handler* interface is provided, which allows for the integration of new constraint types, and provides support for many different well-known types of constraints out of the box. Further, advanced solving methods like primal heuristics, branching rules, and cutting plane separators can also be integrated as plugins with a pre-defined interface. SCIP comes with many such plugins needed to achieve a good MILP and MINLP performance. In addition to plugins supplied as part of the SCIP distribution, new plugins can be created by users. The design approach and solving process is described in detail by Achterberg [2].

Although it is a standalone solver, SCIP interacts closely with the other components of the SCIP Optimization Suite. ZIMPL is integrated into SCIP as a reader plugin, making it possible to read ZIMPL problem instances directly by SCIP. PAPILO is integrated into SCIP as an additional presolver plugin. The LPs that need to be solved as relaxations in the branch-and-bound process are by default solved with SOPLEX. Interfaces to most actively developed external LP solvers exist, and new interfaces can be added by users. GCG extends SCIP to automatically detect problem structure and generically apply decomposition algorithms based on the Dantzig-Wolfe or the Benders' decomposition schemes. Finally, the default instantiations of the UG framework use SCIP as a base solver in order to perform branch-and-bound in parallel computing environments with shared or distributed memory architectures.

*New Developments and Structure of the Report* This report is structured into three main parts. First, the changes and progress made in the solving process of SCIP are explained and the resulting performance improvements on MILP and MINLP instances are analyzed, both in terms of performance and robustness. A performance comparison of SCIP 9.0 against SCIP 8.0 is carried out in Section 2.

Second, improvements to the core of SCIP are presented in Section 3, which include

- improved symmetry handling on non-binary variables, symmetry handling for custom constraints, and signed permutation symmetries,
- symmetry preprocessing using the new interfaces to NAUTY and SASSY,
- a new constraint handler for signomial inequalities as well as cut-strengthening for quadratic expressions,
- a new indicator diving heuristic, extensions to the existing dynamic partition search heuristic, as well as a new online scheduling feature for primal heuristics,
- a new Lagromory separator, as well as improvements in cut selection,
- a new branching criterion called GMI branching that is incorporated into the existing scoring function and acts as a tie-breaker for the existing branching rules,
- a new interface to the HIGHS LP solver [54], and
- certain technical improvements.

Third, improvements to the other components of the SCIP Optimization Suite and extensions to the interfaces are presented. Improvements to the default LP solver SOPLEX and presolver PAPILO are explained in Sections 4 and 5, respectively. Extensions to the interfaces of SCIP are presented in Section 6. Besides improvements and extensions to existing interfaces, this section includes two new interfaces for SCIP: (1) RUSSCIP [80], a new Rust interface, and (2) SCIP++ [83], a new C++ interface, as well as a new Python interface for SOPLEX called PYSOPLEX [78]. Improvements to distributed computing with UG and to Dantzig-Wolfe decompositions with GCG are presented in Sections 7, and 8, respectively; and finally updates to the SCIP extension SCIP-SDP

for semidefinite problems are presented in Section 9. Not included in this release, but available as a beta version, is ExactSCIP [27], a new extension of SCIP that allows for the exact solution of MILPs with rational input data without roundoff errors and zero numerical tolerances.

## 2 Overall Performance Improvements for MILP and MINLP

In this section, we present computational experiments conducted by running SCIP without parameter tuning or algorithmic variations to assess the performance changes since the 8.0.0 release. We detail below the methodology and results of these experiments.

The indicators of interest to compare the two versions of SCIP on a given subset of instances are the number of solved instances, the shifted geometric mean of the number of branch-and-bound nodes, and the shifted geometric mean of the solving time. The *shifted geometric mean* of values $t_1, \ldots, t_n$ is

$$\left(\prod_{i=1}^{n}(t_i + s)\right)^{1/n} - s.$$

The shift $s$ is set to 100 nodes and 1 second, respectively.

### 2.1 Experimental Setup

As baseline we use SCIP 8.0.0, with SoPlex 6.0.0 as the underlying LP solver, and PaPILO 2.0.0 for enhanced presolving. We compare it with SCIP 9.0.0 with SoPlex 7.0.0 and PaPILO 2.2.0. Both SCIP versions were compiled using GCG 10.2.1, use Ipopt 3.14.14 as NLP subsolver built with HSL MA27 as linear system solver, Intel MKL as linear algebra package, CppAD 20180000.0 as algorithmic differentiation library, and bliss 0.77 for graph automorphisms to detect symmetry in MIPs. SCIP 9.0.0 additionally uses sassy 1.1 as a preprocessor for bliss. The time limit was set to 7200 seconds in all cases. Furthermore, for MINLP, a relative gap limit of $10^{-4}$ and an absolute gap limit of $10^{-6}$ were set.

The MILP instances were selected from MIPLIB 2017 [40], including all instances previously solved by previous SCIP versions with at least one of five random seeds or newly solved by SCIP 9.0.0 with at least one of five random seeds; this amounted to 158 instances. The MINLP instances were selected in a similar way from the MINLPLib[1] for a total of 179 instances.

All performance runs were carried out on identical machines with Intel Xeon Gold 5122 @ 3.60GHz and 96GB RAM. A single run was carried out on each machine in a single-threaded mode. Each instance was solved with SCIP using five different seeds for random number generators. This results in a testset of 790 MILPs and 810 MINLPs. Instances for which the solver reported numerically inconsistent results are excluded from the results below.

### 2.2 MILP Performance

Table 1 presents a comparison between SCIP 9.0 and SCIP 8.0 regarding their MILP performance. SCIP 9.0 improves the solving capabilities for MILP by solving 19 more instances than SCIP 8.0. In terms of the shifted geometric mean of the running time, both versions perform almost equally across all instances, with SCIP 9.0 being 2% faster

[1] `https://www.minlplib.org`

on affected instances. On the subset of harder instances in the [1000,7200] bracket, i.e., instances that take at least 1000 seconds to be solved with at least one setting, the speedup is larger and amounts to 6%. To compare average tree size across the two versions, we restrict to the both-solved subset since the number of nodes for instances that time out is not easy to interpret. In the both-solved subset, SCIP 9.0 significantly reduces the average tree size by 17%. Finally, it's worth noting that SCIP 9.0 incorporates a large number of bugfixes. While these bugfixes have introduced a slowdown, they contribute significantly to the overall reliability of the solver.

**Table 1:** Performance comparison of SCIP 9.0 and SCIP 8.0 for MILP instances

| | | SCIP 9.0.0+SoPlex 7.0.0 | | | SCIP 8.0.0+SoPlex 6.0.0 | | | relative | |
|---|---|---|---|---|---|---|---|---|---|
| Subset | instances | solved | time | nodes | solved | time | nodes | time | nodes |
| all | 785 | 637 | 433.9 | 4307 | 618 | 439.0 | 5236 | 1.01 | 1.22 |
| affected | 647 | 610 | 297.1 | 3874 | 591 | 301.8 | 4763 | 1.02 | 1.23 |
| [0,tilim] | 674 | 637 | 272.9 | 3332 | 618 | 276.7 | 4065 | 1.01 | 1.22 |
| [1,tilim] | 669 | 632 | 283.8 | 3423 | 613 | 287.7 | 4182 | 1.01 | 1.22 |
| [10,tilim] | 617 | 580 | 399.8 | 4463 | 561 | 404.9 | 5567 | 1.01 | 1.25 |
| [100,tilim] | 460 | 423 | 986.9 | 11282 | 404 | 985.4 | 14244 | 1.00 | 1.26 |
| [1000,tilim] | 278 | 241 | 2240.0 | 30971 | 222 | 2383.3 | 41101 | 1.06 | 1.33 |
| diff-timeouts | 93 | 56 | 3899.0 | 100922 | 37 | 4592.9 | 160892 | 1.18 | 1.59 |
| both-solved | 581 | 581 | 178.1 | 1897 | 581 | 176.2 | 2220 | 0.99 | 1.17 |

### 2.3 MINLP Performance

Table 2 summarizes the results for the performance of SCIP 9.0 as compared to SCIP 8.0 for the MINLP instances. Besides increasing the number of solved instances by 5, the changes introduced in SCIP 9.0 improve the performance of SCIP in both the overall solving time as well as the number of nodes needed. On the whole testset, SCIP 9.0 improves the performance by about 4% in time and by 13% in nodes, both in shifted geometric means. This improvement increases with the difficulty of the instances: When looking at the most difficult testset [1000, 7200], SCIP 9.0 outperforms SCIP 8.0 by 20% and 46% in the solving time and nodes in shifted geometric means, respectively. Furthermore, the improvement in solving time is mainly observed on nonconvex instances. In particular, SCIP 9.0 is 8% faster than SCIP 8.0 on nonconvex instances whereas both versions perform almost equally when only convex instances are considered.

## 3 SCIP

### 3.1 Symmetry Handling

Symmetries of an MILP or MINLP are maps that transform feasible solutions into feasible solutions with the same objective value. When not handled appropriately, such symmetries deteriorate the performance of (spatial) branch-and-bound algorithms since symmetric solutions are found and symmetric subproblems are explored repeatedly without providing additional information to the solver. The previous versions of SCIP have already contained many state-of-the-art algorithms to handle symmetries of binary variables and some basic cutting planes to also handle symmetries of integer or continuous variables.

SCIP 9.0 substantially extends the ability to handle symmetries in three directions. First, more sophisticated techniques are available to handle symmetries of non-binary variables. Second, the mechanism to detect symmetries has been completely restructured. While previous versions of SCIP could only detect symmetries of the available classes

**Table 2:** Performance comparison of SCIP 9.0 and SCIP 8.0 for MINLP instances

| | | SCIP 8.0.0+SoPlex 6.0.0 | | | SCIP 9.0.0+SoPlex 7.0.0 | | | relative | |
|---|---|---|---|---|---|---|---|---|---|
| Subset | instances | solved | time | nodes | solved | time | nodes | time | nodes |
| all | 839 | 810 | 32.6 | 2800 | 815 | 31.2 | 2489 | 1.04 | 1.13 |
| affected | 783 | 770 | 32.4 | 2949 | 775 | 31.4 | 2608 | 1.03 | 1.13 |
| [0,7200] | 823 | 810 | 29.3 | 2613 | 815 | 28.0 | 2323 | 1.04 | 1.12 |
| [1,7200] | 767 | 754 | 36.6 | 3288 | 759 | 34.9 | 2907 | 1.05 | 1.13 |
| [10,7200] | 529 | 516 | 97.0 | 6333 | 521 | 90.2 | 5498 | 1.08 | 1.15 |
| [100,7200] | 248 | 235 | 489.1 | 29290 | 240 | 414.0 | 21767 | 1.18 | 1.35 |
| [1000,7200] | 96 | 83 | 2020.8 | 110115 | 88 | 1682.8 | 75556 | 1.20 | 1.46 |
| diff-timeouts | 21 | 8 | 3492.6 | 72059 | 13 | 1691.3 | 21896 | 2.07 | 3.29 |
| both-solved | 802 | 802 | 25.7 | 2389 | 802 | 25.1 | 2187 | 1.03 | 1.09 |
| convex | 168 | 163 | 31.4 | 3601 | 165 | 31.0 | 3177 | 1.01 | 1.13 |
| nonconvex | 571 | 547 | 38.0 | 2783 | 550 | 35.2 | 2443 | 1.08 | 1.14 |

of constraints, SCIP 9.0 can also detect symmetries of custom constraints added by users. Moreover, SCIP 9.0 can also detect so-called signed permutation symmetries, whereas previous versions could only detect permutation symmetries. Finally, to detect symmetries, SCIP makes use of external software for detecting graph automorphisms. In the latest version, new interfaces to NAUTY [74] as well as the preprocessor SASSY [6] have been added.

#### 3.1.1 Symmetry Handling Methods

For the ease of exposition, consider an MILP $\min\{c^\top x : Ax \geq b,\ x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}$, where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. Symmetries in MILP are commonly permuting variables, i.e., a permutation $\gamma$ of $[n]$ acts on $x \in \mathbb{R}^n$ as $\gamma(x) = (x_{\gamma^{-1}(1)}, \dots, x_{\gamma^{-1}(n)})$. Permutation $\gamma$ is called a *formulation symmetry* of the MILP if there is a permutation $\pi$ of $[m]$ such that $\gamma(c) = c$, $\pi(b) = b$, $A_{\pi^{-1}(i),\gamma^{-1}(j)} = A_{i,j}$ for all $(i,j) \in [m] \times [n]$, and $\gamma(i) \in [p]$ for all $i \in [p]$. Formulation symmetries can be detected by constructing a colored graph whose color-preserving automorphisms correspond to symmetries of the MILP, see [81] and Section 3.1.2 for more details. Moreover, the definition of formulation symmetries can be extended to MINLPs by keeping the representation of nonlinear constraints via expression trees invariant [63].

In previous versions of SCIP, three main classes of symmetry handling methods have been available:

1. Propagation and separation algorithms for the symmetry handling constraints orbisack [50, 51, 56, 66], symresack [50, 51], and orbitope [9, 57, 58]; these constraints have only been able to handle symmetries of binary variables and enforce symmetry reductions based on a scheme that is determined before the solving process starts.
2. The propagation method orbital fixing [70, 75] to handle symmetries of binary variables; the corresponding symmetry reduction scheme is determined dynamically during the solving process.
3. Schreier-Sims table cuts (SST cuts) [64, 82], which are cutting planes that are added to the MILP/MINLP and can handle symmetries of arbitrary variable types.

Note that the first two classes are not compatible with each other due to the different symmetry handling schemes.

SCIP 9.0 features an implementation of a generalization of the first two classes of methods as discussed in [95]. This generalization allows to also handle symmetries of non-binary variables and to apply both classes simultaneously. At the time of adding this new feature, the performance of SCIP improved by 5.90 % on the MIPLIB 2017 benchmark test set; on the hard instances that take at least 1000.00 s to be solved, the

running time even improved by 25.40 %.

### 3.1.2 Symmetry Detection

In previous versions of SCIP, symmetries could only be detected if all types of constraints present in an MILP or MINLP are known by SCIP, i.e., constraints whose corresponding constraint handler is part of the SCIP release. In particular, this means that no symmetries could be detected and automatically handled by SCIP in the presence of custom constraints. For SCIP 9.0, the symmetry detection mechanism has been restructured. Constraint handlers now support two optional callbacks `CONSGETPERMSYMGRAPH` and `CONSGETSIGNEDPERMSYMGRAPH` that allow constraint handlers to inform SCIP about symmetries of their constraints. The former is used for the detection of permutation symmetries, whereas the latter allows to detect signed permutation symmetries that we define below. During run time, SCIP checks whether the constraint handlers of all constraints present in a problem implement the new callbacks. If this is the case, symmetries are detected; otherwise, symmetry detection is disabled.

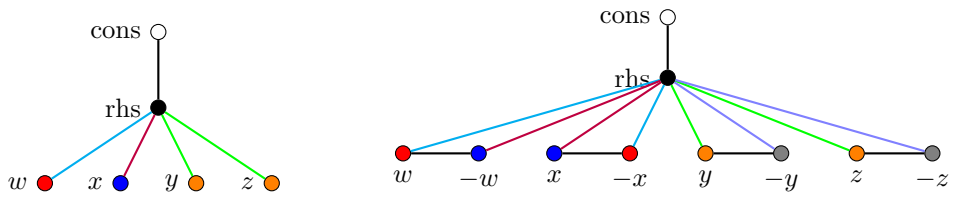
*Detection of Permutation Symmetries* As briefly explained above, permutation symmetries of an MILP or MINLP can be detected by finding automorphisms of a suitable colored graph, which we call the symmetry detection graph. In the following, we explain this mechanism and how it can be implemented using the `CONSGETPERMSYMGRAPH` callback. We illustrate the ideas using the simple MILP $\max\{y+z : -2w+2x+3y+3z \leq 4,\ y,\ z \in \{0,1\}\}$.

To detect symmetries, every constraint defines its own local symmetry detection graph. Such a graph contains a colored node for every variable that is present in the constraint as well as further colored nodes and edges that model dependencies between the different variables. The graph for a constraint qualifies as a symmetry detection graph for SCIP if it is connected and the restriction of every color-preserving automorphism to the variable nodes corresponds to a permutation symmetry of the corresponding constraint. Moreover, two symmetry detection graphs are only allowed to be isomorphic if their constraints are equivalent.

A possible symmetry detection graph for our exemplary MILP is shown in Figure 1a. The nodes for variables $y$ and $z$ receive the same color since both have the same objective coefficient and bounds; the remaining variable nodes receive different colors, since they are not symmetric to each other. Moreover, we introduce one node for the right-hand side, which is colored according to the right-hand side coefficient. The edges connect the variable nodes with the right-hand side node; they are colored according to the coefficient of the corresponding variable in the linear constraint. This construction can easily be extended to general linear constraints, see [81].

The symmetry detection graphs for individual constraints are then combined into a single symmetry detection graph. The callback `CONSGETPERMSYMGRAPH` provides a pointer to this graph and different functions can be used add nodes and edges to this "global" symmetry detection graph. To avoid re-defining variable nodes for different constraints, these nodes cannot be added within the callback. Instead, these nodes are defined centrally by SCIP and are colored according to the variable's type, its objective coefficient, and lower and upper bound. To make sure that only constraints of the same type can be symmetric to each other (compare the permutation $\pi$ above), every constraint should add a "constraint" node to its local symmetry detection graph, which serves as an identifier of the type of constraint (e.g., "linear", "knapsack", or "SOS1").

The functions for adding nodes to the global graph are `SCIPaddSymgraphValnode()`, `SCIPaddSymgraphOpnode()`, and `SCIPaddSymgraphConsnode()`. The first function adds nodes that hold a numerical value, e.g., the right-hand side node of a linear constraint. The second function can be used to add "operator" nodes that allow to model special relations between other nodes. For example, in a nonlinear constraint such an operator




**(a)** permutation symmetries **(b)** signed permutation symmetries

**Figure 1:** Illustration of examplary symmetry detection graphs.

could model (nonlinear) functions that are applied to other nodes (e.g., variables). Operators are encoded as integer numbers, i.e., the implementation needs to make sure that only equivalent operators are assigned the same integer value. The third function adds a node that stores a pointer to the corresponding constraint.

Edges can be added by `SCIPaddSymgraphEdge()`. To add edges to variable nodes, the function `SCIPgetSymgraphVarnodeidx()` can be used to get the index of the variable node in the symmetry detection graph.

*Detection of Signed Permutation Symmetries* Let $e^1, \dots, e^n$ be the standard unit vectors in $\mathbb{R}^n$. A signed permutation is a bijective map $\gamma\colon \{\pm e^1, \dots, \pm e^n\} \to \{\pm e^1, \dots, \pm e^n\}$ that satisfies $\gamma(-e^i) = -\gamma(e^i)$ for all $i \in [n]$. A signed permutation $\gamma$ acts on a vector $x \in \mathbb{R}^n$ as $\gamma(x) = (\operatorname{sgn}(\gamma^{-1}(1))\, x_{|\gamma^{-1}(1)|}, \dots, \operatorname{sgn}(\gamma^{-1}(n))\, x_{|\gamma^{-1}(n)|})$, where $\operatorname{sgn}(\cdot)$ is the sign function. That is, it permutes the entries of the vector, but it can also change the sign of some entries. Signed permutation symmetries of an MILP or MINLP can be defined analogously to permutation symmetries. Such symmetries arise, e.g., in geometric problems like packing circles into a box [19], where they model reflections along standard hyperplanes.

To detect signed permutation symmetries, constraint handlers need to implement the callback `CONSGETSIGNEDPERMSYMGRAPH`. The functionality is analogous to the one of `CONSGETPERMSYMGRAPH`, however, the automorphisms of the symmetry detection graph need to encode signed permutation symmetries now. This can be achieved by introducing not only a node for every variable $x$, but also for every negated variable $-x$. The colors of the negated variables are derived based on the negated objective coefficient and negated bounds of variable $x$. Moreover, to indicate that $x$ and $-x$ form a pair of negated variables, both must be connected by an edge in the symmetry detection graph. Finally, the definition of signed permutations above requires that only reflections along standard hyperplanes are allowed. In SCIP's implementation, however, also reflections along translated standard hyperplanes can be detected. This is achieved by not defining the colors based on the original variable bounds, but for variables whose domain is translated to be centered at the origin (except for semi-unbounded variables). Consequently, also binary variables may admit signed permutation symmetries since the variable domain is translated to $\{-\frac{1}{2}, +\frac{1}{2}\}$.

Figure 1b shows the symmetry detection graph for our illustrative example. Next to the classical permutation symmetry that exchanges $y$ and $z$, also the signed permutation that maps variable $w$ onto variable $-x$ can be detected.

To create the symmetry detection graph for signed permutation symmetries all functions to add nodes and edges as described above can be used. To access the index of a negated variable, `SCIPgetSymgraphNegatedVarnodeidx()` needs to be used.

Via the parameter `propagating/symmetry/symtype` a user can select which type of symmetries is detected. A value of 0 corresponds to permutation symmetries, and a value of 1 to signed permutation symmetries. By default, signed permutation symmetries are not detected, because currently only basic symmetry handling techniques for such symmetries are implemented.

#### 3.1.3 Symmetry Interfaces

To detect automorphisms of the previously mentioned symmetry detection graphs, the previous version of SCIP made use of the graph automorphism software BLISS [55], which is also shipped together with SCIP. SCIP 9.0 also features interfaces to NAUTY/-TRACES [74]. Depending on which software package shall be used for symmetry detection, the SCIP make command takes `SYM=`{`none`,`bliss`,`nauty`} as argument. Moreover, SCIP 9.0 allows to make use of SASSY [6], which preprocesses the symmetry detection graphs to accelerate the computation of symmetries. SASSY can be used by compiling SCIP with option `SYM=`{`sbliss`,`snauty`}.

### 3.2 Nonlinear Handlers

Nonlinear constraints (see (2)) are handled by the constraint handler `nonlinear` in SCIP. This constraint handler can delegate tasks on detecting and exploiting structure in algebraic expressions to specialized *nonlinear handlers*, see [10] for details. For example, the nonlinear handler for quotient expressions identifies expression of the form $vw^{-1}$ (where $v$ and $w$ can be arbitrary expressions) in nonlinear handlers and provides specialized bound tightening and linear under/overestimators for the function $(v, w) \mapsto vw^{-1}$.

With SCIP 9.0, a new nonlinear handler for signomial functions has been added and the nonlinear handler for quadratic expressions has been improved. In addition, a new nonlinear handler callback has been added to request the linearization of an expression in a given solution point. The nonlinear constraint handler can use this callback to tighten the linear relaxation when a new feasible solution has been found (parameter `constraints/nonlinear/linearizeheursol`, currently disabled by default).

#### 3.2.1 Signomial Handler

An $n$-variate *signomial term* is defined as $x^\alpha = \prod_{j\in[n]} x_j^{\alpha_j}$, where $\alpha \in \mathbb{R}^n$ and $x > 0$. In general, the signomial term is nonconvex. In SCIP 9.0, a new nonlinear handler is implemented that generates cutting planes for signomial constraints.

Given $x^\alpha$, the handler aims at approximating the lifted set $S \coloneqq \{(x,t) \in \mathbb{R}^n \times \mathbb{R} : t = x^\alpha\}$, which is in general given by a constraint of the extended formulation (see the 8.0.0 release report [10] for details on extended formulations). It is easy to show that $S$ can be rewritten in the form

$$S = \{(u,v) \in \mathbb{R}^{h+\ell}_+ : u^{\bar\beta} = v^{\bar\gamma}\}. \tag{3}$$

with $\bar\beta, \bar\gamma$ containing only nonnegative entries.

Given a point $(\tilde u, \tilde v)$, the handler outer approximates either $S_1 \coloneqq \{(u,v) \in \mathbb{R}^{h+\ell}_+ : u^{\bar\beta} \le v^{\bar\gamma}\}$ or $S_2 \coloneqq \{(u,v) \in \mathbb{R}^{h+\ell}_+ : u^{\bar\beta} \ge v^{\bar\gamma}\}$ by checking which of the two sets does not contain $(\tilde u, \tilde v)$. More precisely, if $(\tilde u, \tilde v) \notin S_1$, then the handler separates a linear valid inequality for $S_1$ that (possibly) cuts off $(\tilde u, \tilde v)$ and overestimates the signomial term; if $(\tilde u, \tilde v) \notin S_2$, then the handler separates a linear valid inequality for $S_2$ that (possibly) cuts off $(\tilde u, \tilde v)$ and underestimates the signomial term. In the case of an inequality constraint, only one of the sets $S_1$, $S_2$ correctly describes the feasible set of the constraint, and thus only one set is considered for separation.

The above formulation exhibits symmetry between $u$ and $v$. We only illustrate the method to approximate $S_1$, and the similar result applies to $S_2$ as well.

Since the signomial terms $u^{\bar\beta}, v^{\bar\gamma}$ are nonnegative over $\mathbb{R}^h_+$, $\mathbb{R}^\ell_+$, we can take any positive power $\eta \in \mathbb{R}_{>0}$ on both sides of (3) to obtain

$$S_1 = \{(u,v) \in \mathbb{R}^{h+\ell}_+ : u^{\beta} \le v^{\gamma}\}, \tag{4}$$

where $\beta \coloneqq \eta\bar{\beta}$, $\bar{\gamma} \coloneqq \eta\bar{\gamma}$, and $\eta = 1/\max(\sum_{j\in[h]}|\bar{\beta}_j|, \sum_{j\in[\ell]}|\bar{\gamma}_j|)$. Thus, we have that $\max(\sum_{j\in[h]}|\beta_j|, \sum_{j\in[\ell]}|\gamma_j|) = 1$.

Moreover, we assume that the range of $u$ is a hyperrectangle $U \subseteq \mathbb{R}^h_{>0}$, which is usually available as variable bounds in SCIP. The reformulated set enjoys two useful properties [100]: the terms $u^\beta, v^\gamma$ in (4) are concave functions, and the convex envelope of $u^\beta$ over $U$ is vertex polyhedral. The signomial handler aims at (1) linearizing the convex envelope of $u^\beta$, i.e., an affine underestimator for $u^\beta$, and (2) linearizing the concave function $v^\gamma$.

Let $Q$ be the vertices of $U$. Since the convex envelope of $u^\beta$ over $U$ is vertex polyhedral, the separation of an affine underestimator $a \cdot u + b$ could be solved by an LP:

$$\max_{a\in\mathbb{R}^h, b\in\mathbb{R}} \{a \cdot \tilde{u} + b \,:\, \forall q \in Q \; a \cdot q + b \le q^\gamma\}, \tag{5}$$

When $h = 1, 2$, the handler directly uses a closed form expression of the optimal $a$, $b$ without solving the LP [100].

Now, denote $g(v) := v^\gamma$. Since $g$ is concave, a straightforward way to overestimate it is to use the gradient at $\tilde{v}$ and obtain $g(\tilde{v}) + \nabla g(\tilde{v}) \cdot (v - \tilde{v})$. Then, the separated valid inequality is of the form $a \cdot u + b \le g(\tilde{v}) + \nabla g(\tilde{v}) \cdot (v - \tilde{v})$. In the implementation, such inequality is further transformed into an overestimator of $x^\alpha$ through scaling.

Table 3 shows the impact of the signomial handler on SCIP performance on 152 MINLPLib instances that contain signomial terms. We report the ratio of shifted geometric means of time and nodes and the number of solved instances.

**Table 3:** Performance statistics of the signomial handler over SCIP default.

| 152 selected MINLPLib instances | | |
|---|---|---|
| Time (s) | Nodes | Solved |
| **0.92** | **0.93** | **78 vs 75** |

The computational cost of estimating a signomial term primarily stems from solving the LP (5), whose size is exponential in $h$. An advanced parameter that governs the maximum allowable value of $h$ that the signomial handler can manage is `nlhdlr/signomial/advanced/maxnundervars`. Currently, the signomial handler is disabled by default. Users can enable it via the `nlhdlr/signomial/enabled` parameter.

The current implementation of the signomial handler lacks a specialized bound tightening method for variables within signomial terms. Given the critical impact of variable range ($U$) on cut quality, a further development of the handler involves refining these bounds through propagation techniques.

### 3.2.2 Strengthening Cuts for Quadratic Expressions

To separate nonconvex quadratic constraints, the constraint handler `nlhdlr_quadratic`, which was introduced in SCIP 8.0, can generate intersection cuts by setting the parameter `nlhdlr/quadratic/useintersectioncuts = TRUE` (currently disabled by default). Until now, the intersection cuts were built by using only the current LP relaxation and the violated quadratic constraint. To additionally leverage integrality information, SCIP 9.0 allows to strengthen the cutting planes using *monoidal strengthening* [8] if some of the variables in the problem need to be integer. As [15] showed, the strengthened intersection cuts significantly outperform the pure intersection cuts whenever monoidal strengthening can be applied.

### 3.3 Primal Heuristics

#### 3.3.1 Indicatordiving

*Semi-continuous* variables are variables that take either the value 0 or any value within a specific range:

$$x \in \{0\} \cup [\ell, u] \text{ with } 0 < \ell \leq u \text{ and } u \in \mathbb{R}_+ \cup \infty.$$

Such variables are used, for example, in modeling supply chains where a facility either can produce nothing or, if enabled, has to produce at least an amount $\ell$.

Semi-continuous variables can be formulated in SCIP with an additional binary variable $z \in \{0, 1\}$ as

$$\begin{aligned} x &\in [0, u], \\ \ell z &\leq x, \qquad (6) \\ z = 0 &\implies x \leq 0. \qquad (7) \end{aligned}$$

Thereby, (7) is a so-called indicator constraint, that is, $x \leq 0$ must hold if $z = 0$. Linear constraint (6) models the lower bound on $x$.

If $u$ is finite, one can reformulate the indicator constraint (7) with a linear big-$M$ constraint, such as $x \leq uz$. If $u$ is infinite, this is not directly possible. However, one could add an artificial upper bound with the risk of cutting off optimal solutions and causing numerical issues due to the large upper bound $M$. A new diving heuristic, `indicatordiving`, has been developed to find solutions also in the presence of indicator constraints modeling semi-continuous variables with infinite upper bounds $u$.

Diving heuristics iteratively fix variables and solve the modified LPs simulating a depth-first-search in an auxiliary tree. A description of the generic diving procedure used in SCIP can be found in [39]. Other diving heuristics in SCIP typically take only integer variables with fractional LP solution value into account. In contrast, `indicatordiving` additionally examines all binary indicator variables $z$ corresponding to violated indicator constraints and which are integral in the LP solution but not fixed already.

Each such variable is assigned a score to determine the variable that should be fixed next. For indicator variables, the score is given by

$$\phi \coloneqq \begin{cases} -1, & \text{if } \hat{x} \in \{0\} \cup [\ell, u], \\ 100 \cdot (\ell - \hat{x})/\ell, & \text{if } \hat{x} \in (0, u), \end{cases}$$

where $\hat{x}$ is the current LP solution value. The indicator variable $z$ with the highest score gets fixed to 1 if the LP value $\hat{x}$ is at least 50% of the lower bound $\ell$. Otherwise, it gets fixed to 0. As soon as all the indicator variables are integral in the LP solution or all the indicator constraints are fulfilled, other candidate variables are considered, for which the score and rounding direction of `farkasdiving` are used.

#### 3.3.2 Extension of Dynamic Partition Search

Since SCIP 7.0, the decomposition information can be passed to SCIP in addition to the instance, which can be leveraged in heuristics, for example. A detailed description of decompositions and their handling in SCIP can be found in the release report for that version [33].

Dynamic Partition Search (`DPS`) introduced in SCIP 8.0 is a heuristic that requires a decomposition. `DPS` splits an MILP (1) into several subproblems according to a decomposition. Thereby, the linking constraints and their right-/left-hand sides are also split by introducing new parameters $p_q$ for each block $q$, called *partition*. Such a

partition is central to the DPS. When the heuristic is called during node preprocessing, the partition is initialized with a uniform distribution of the constraint sides over the blocks.

In SCIP 9.0, the DPS has been extended with an option to get called at the end of the node processing and, therefore, can use the LP solution for initialization of the partition. The parameter `heuristics/dps/timing` controls the calling point and, thus, the initialization. A detailed description of DPS and additional heuristics exploiting decomposition information can be found in [44].

#### 3.3.3 Learning to Control Primal Heuristics Online

Since the performance of heuristics is highly problem-dependent and most of them can be very costly, it is necessary to handle them strategically. Thus, sometimes it is preferable to have dynamic, self-improving procedures rather than relying on static methods to control primal heuristics.

SCIP has already used two adaptive heuristics in previous versions that use bandit algorithms to decide which heuristics to additionally run: Adaptive Large Neighborhood Search (ALNS) [45] and Adaptive Diving [46]. Building upon this, SCIP 9.0 now includes a general online learning approach [14], which dynamically adapts the application of primal heuristics to the unique characteristics of the current instance. In particular, both *Large Neighborhood Search* and *Diving* heuristic types are controlled together, making this the first work where two different classes of heuristics are treated simultaneously by a single learning agent. In SCIP 9.0, this is implemented as a heuristic called `scheduler`. Since this framework was designed to replace the classical heuristic handling as a whole as opposed to being run as an additional heuristic, `scheduler` is disabled by default in SCIP 9.0.

### 3.4 Cutting Planes

This section discusses the updates to the separation routine in SCIP, both for cut generation and cut selection. Separation in SCIP is performed in rounds. In a round, various valid inequalities that cut off the current LP relaxation's fractional solution are generated and stored either in a global cut pool (for cuts that are valid globally) or a separation store (for cuts that are valid only locally). Then, these cuts are filtered and added to the LP relaxation before re-solving the relaxation and proceeding to the next round of separation. Note that other components of SCIP such as branching maybe executed in between two separation rounds. These rounds are performed until a stopping criterion is met (e.g., maximum number of rounds or cuts added, or dual bound of the relaxation stalling).

#### 3.4.1 Cut Generation

*Lagromory Separator* There are two potential issues with the round-based approach mentioned above.

1. The generation of higher-ranked cuts, e.g., higher-ranked Gomory Mixed-Integer (GMI) cuts [18], may result in numerical troubles during the solving process.
2. Multiple rounds of separation may be needed to achieve dual bound improvement in the presence of dual degeneracy [34].

The first issue is addressed in the literature via a relax-and-cut framework-based separation techniques [43, 68, 12, 28]. The second issue has received less attention until now despite being critical to the usefulness of separation in the solvers. The new separator in

SCIP, the Lagromory separator, addresses both these issues. It is a relax-and-cut framework-based separator which is built based on the separation technique presented in [28].

In the basic version of the relax-and-cut framework, which is also discussed in [28], when a separator is called at a node with fractional LP solution, certain cuts are generated but not added directly to the LP relaxation. These cuts are added to the objective function of the node LP in a Lagrangian fashion using Lagrangian (penalty) multipliers. Then, this Lagrangian dual problem is solved via an iterative approach by updating the Lagrangian multipliers in every iteration, requiring an LP solving in every iteration. When an LP is solved in an iteration, it is equivalent to exploring a new basis of the node LP. Then, additional cuts are generated with respect to this newly explored basis and are added to the objective function of the node LP again. This procedure is repeated until certain termination criterion is met. While this approach in [28] was proven to improve the dual bound at a given single node, it turned out to be ineffective in the context of the entire branch-and-cut tree. To overcome this crucial hurdle, the Lagromory separator in SCIP also implements various novel enhancements.

- Theoretical enhancements include stabilization and regularization of the vector of Lagrangian multipliers. This vector is integral to and iteratively updated in the relax-and-cut framework. Stabilization using a core vector is an essential component in the literature of decomposition methods such as the Benders' and Dantzig-Wolfe decompositions. Regularization of vectors (e.g., by projecting the vectors into $\ell_1$, $\ell_2$, etc, norm balls) is a commonly applied technique in the literature of nonlinear optimization.
- Computational enhancements include the threshold for dual degeneracy beyond which the separator is executed; the working limits on the number of LP iterations, number of cuts generated per explored basis of the LP relaxation; etc.

The separator was tested on the MIPLIB 2017 benchmark library [40]. It speeds up the solving process of harder instances that require at least 1000 seconds for solving to optimality. On the other hand, it increases the solving time for many easier instances resulting in no overall improvement of the default SCIP performance. The Lagromory separator is OFF by default due to this reason. An interested user may switch it ON by changing the parameter `separation/lagromory/freq` to a non-negative number.

### 3.4.2 Cut Selection

The cut selector plugin introduced in SCIP 8.0 enabled multiple research directions on the problem of cut selection, which recently underwent a revived scrutiny. The previously hard-coded algorithm was replaced by the default cut selector `cutsel/hybrid`, which in particular scores cuts with a weighted sum of four criteria: efficacy, integer support, objective parallelism, and directed cutoff distance (with the last one having a zero default weight). The scored cuts are then filtered iteratively by an orthogonality criterion. This removes all non-orthogonal cuts (within some tolerance), starting from the highest-scoring cut until every cut has been processed, and the remaining set of cuts is pairwise near-orthogonal. The importance of cut selection and some limitations of the current criteria were in particular shown theoretically and computationally in [94, 92].

These lines of work point to the conclusion that the current cut selection algorithm fails to capture and adapt to important instance properties. In [90], a new cut selector coined *ensemble* was developed to capture more of these properties than the current weighted sum of the default `cutsel/hybrid`. The three core aspects of the cut selection loop described above are the *filtering* of cuts, their *scoring*, and the *stopping criteria* for the cut loop. All three aspects were extended and modified in the new ensemble cut selector, resulting in a large number of parameters that were activated and adjusted

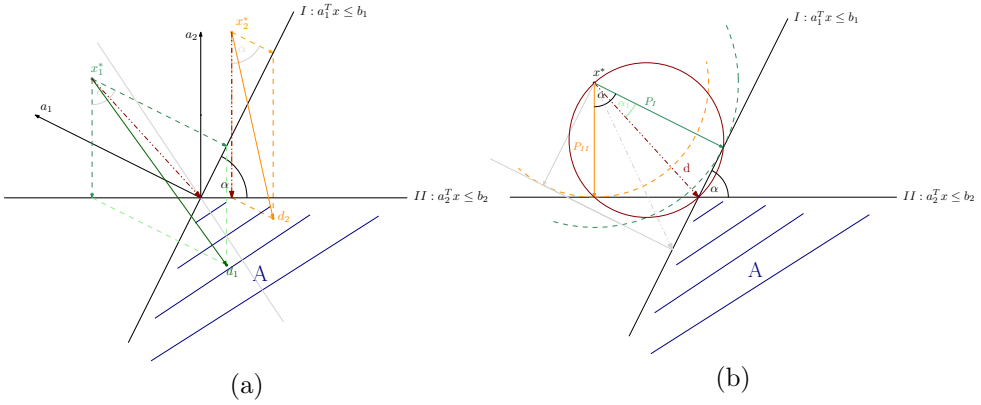



**Figure 2:** (a) Pairwise cut efficacies, shown for two cases of LP solutions in yellow and green for cutting planes I and II. (b) Orthogonality-based cutfiltering, shown by the offset between the pairwise efficacies in red and gray of cut I paired with either cut II or the orthogonally rotated cut II.

through the blackbox hyperparameter optimization tool SMAC [65]. These parameters include pseudo-cost based cut scoring, sparsity based cut scoring, density based cut filtering, parallelism based scoring penalties, and a stopping criterion based on the number of nonzeros in the added cuts. Because of the filter on cut density, `cutsel/ensemble` performs unreliably on MINLPs, where some nonlinear constraints demand the use of high-density cuts for good solver performance. It is therefore applied with a lower priority than `cutsel/hybrid` which remains the default, but can be activated by changing `cutselection/ensemble/priority` to a number greater than 8000, the value of `hybrid`.

Further, a novel dynamic filtering method is introduced in `cutsel/dynamic`, which aims to enhance the current near-orthogonal threshold methodology used in the `cutsel/hybrid` selector. The geometric rationale for this approach can be understood by considering how the LP improvements for a pair of cutting planes, in terms of efficacy, are influenced by the position of their intersection relative to the current LP optimal solution $x^*$ (refer to Figure 2a).

This is exemplified by demonstrating that a mere summation of individual cut efficacies can be misleading in evaluating the actual efficacy of a pair of cuts. The true efficacy, indicated by red arrows in Figure 2a, represents the minimal distance between the feasible region $A$ and the current LP solution post-application.

Figure 2a illustrates a fundamental issue when using pure efficacy to evaluate multiple cuts simultaneously. This underscores the motivation for orthogonality-based filtering as showcased in Figure 2b. Specifically, if the current LP solution is outside the fan formed by the intersection of the cuts (yellow), then one of the cuts becomes entirely ineffective, provided that the vertex created by this pair is not the optimal point in the subsequent iteration. Conversely, if the LP solution is within this fan, the aggregated efficacy of the cuts does not accurately reflect their true effectiveness. This discrepancy is due to the degree of non-orthogonality between the cuts.

To address these limitations, the stringent orthogonality threshold of the default selector was relaxed. The new dynamic criterion is depending on individual cut efficacies and aims to position the LP solution within the intersection fan of the cut pairs (as depicted in Figure 2b). Additionally, users can specify a minimum efficacy improvement relative to the previous cutting plane via the *mingain* parameter. The *filtermode* parameter 'f' in `cutsel/dynamic` facilitates rescoring of cuts between filtering rounds, using the pairwise efficacy instead of the usual scoring mechanism. Preliminary test results have not indicated a general improvment over the default cut selection; hence, this selector is assigned a lower priority than `cutsel/hybrid` as well.

### 3.5 Branching

SCIP 9 introduces a new branching criterion both implemented as a stand-alone rule and integrated within the default hybrid branching rule [4]. For a more thorough overview of the results and idea presented in this section, see [91].

#### 3.5.1 GMI Branching

GMI cuts [42] are a standard tool for solving MILPs. It is recommended to see [7, 17, 91] for an overview of how GMI cuts are derived and applied in practice. The motivation to use GMI cuts to influence branching decisions stems from the observation that GMI cuts are derived from a split disjunction. A *split disjunction* is defined by an integer $\pi_0 \in \mathbb{Z}$ and an integral vector $\pi \in \mathbb{Z}^p \times \{0\}^{n-p}$, which has zero entries for coefficients of continuous variables. The disjunction is then given by:

$$\{x \in \mathbb{R}^n \mid \pi_0^T x \leq \pi_0\} \cup \{x \in \mathbb{R}^n \mid \pi_0^T x \geq \pi_0 + 1\}.$$

Since no integer point is contained in the split $\{x \in \mathbb{R}^n \mid \pi_0 < \pi_0^T x < \pi_0\}$, all feasible points lie in the above split disjunction. Thus, a cut that only cuts off points from the continuous relaxation that are inside the split is valid for the original problem. Such cuts are called *split cuts* and GMI cuts are a special case of this family of cutting planes. An example split with a valid split cut is visualized in Figure 3.

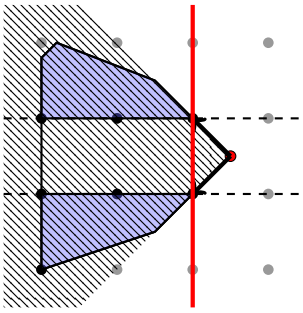


**Figure 3:** Example intersection cut that is also a split cut.

To best link GMI cuts and branching, note that for $\pi = e_i$ and $\pi_0 = \lfloor \bar{x}_i \rfloor$, where $e_i$ denotes the $i$-th unit vector and $\bar{x}$ the current LP solution, the split corresponds exactly to the region that is excluded by branching on a fractional variable $x_i$, $i \in \mathcal{I}$. The simple logic of this branching rule is then the following: Variables whose splits would generate a deep cut might also be good branching candidates. Therefore, the new Gomory branching rule generates all GMI cuts (or up to some maximum number of candidates if set), and branches on the variable whose associated split produces the most efficacious GMI cut.

#### 3.5.2 Using GMI Cuts for Reliability Pseudo-Cost (Hybrid) Branching

The default branching rule of SCIP [4] has been extended to include two new terms in the weighted sum scoring rule. Currently, the variable that is branched on by SCIP is the one with the highest branching score (ignoring cases of epsilon-close results). The score for each variable is computed via a weighted sum rule that combines the following measures:

- The frequency that the variable appears in a conflict
- The average length of the conflicts that the variable appears in

– The frequency that the variable's branching history has resulted in other variables becoming fixed
– The frequency that the variable's branching history has resulted in infeasible subproblems
– The number of NL constraints the variable features in
– The pseudo-cost associated with the variable from its history of previous branching decisions.

This weighted sum rule is appended with two terms controlled by the weight parameters `gmiavgeffweight` and `gmilasteffweight`. In a separation round, SCIP now stores the normalized efficacy of generated GMI cuts. The normalization is a simple division by the largest efficacy of any GMI cut generated in the separation round, resulting in a value in the range $[0, 1]$ for each variable from which the tableau row produced a GMI cut. For each variable in the problem, SCIP now records a running average of the normalized GMI cut efficacies from tableau rows associated with the respective variable. Additionally, SCIP now also records the last normalized value for each variable. The parameter `gmiavgeffweight` is then the weight of the running average in the weighted sum rule for each variable. The parameter `gmiavgeffweight` is equivalently the weight of the most recent recorded value for each variable. By default, `gmiavgeffweight` is set to 0, and `gmilasteffweight` is set to $10^{-5}$, effectively acting as a tie-breaker.

### 3.6 LP Interfaces

HiGHS *LPI* SCIP 9.0 provides the possibility of using the open-source LP solver HiGHS [54] [2] . The interface provides the basic functionality, yet it does not fully exploit all capabilities of HiGHS.

### 3.7 Technical Improvements

*AMPL .nl reader* Added support for logical constraints in binary variables and basic logical operators (and, or, not, equal).

*OBBT propagator* Variables of linear constraints that are controlled by indicator constraints can now also be taken into account for bound tightening. This feature is disabled by default, but can be enabled via parameter `propagating/obbt/indicators`.

## 4 SoPlex

Most importantly, SoPlex 6.0 now supports *incremental precision boosting* [26] for solving LPs exactly over the rational numbers, in addition to and in combination with the existing LP iterative refinement approach [38, 37]. The algorithm for exact solving can be selected using the boolean parameters `precision_boosting` and `iterative_refinement`. By default, both are set to `true`, in which case SoPlex uses a combined algorithm with an outer precision boosting loop and an inner iterative refinement loop. For further details and computational experiments we refer to [25]. The new default for exact, rational LP solving increases the robustness of the algorithm on numerically challenging problems and allows to solve more problems exactly. Furthermore, a new Python interface for SoPlex 6.0, called PySoPlex, has been developed, see Section 6.7.

[2] Available in source code at `https://github.com/ERGO-Code/HiGHS`.

# 5 PaPILO

PaPILO, a C++ library, is a solver-independent presolving library that provides presolving routines for MIP and LP and is part of the presolving routines in SCIP. It also supports multi-precision arithmetic, which makes PaPILO an essential part of the presolving process of ExactSCIP [27], the numerically exact version of SCIP [16, 23, 24].

PaPILO 2.2 now supports *proof logging* as a new feature, i.e., the generation of machine-verifiable certificates by a solver in order to prove the correctness of its computation. Proof logging was originally introduced by the SAT community to ensure the correctness of a solver's computation, since even state-of-the-art solvers falsely claim infeasibility or optimality or return infeasible "solutions" [5, 16, 59, 88, 36]. Examples of such proof formats are DRAT [96, 47, 48], GRIT [21], and LRAT [20]. ExactSCIP has adapted proof logging to certify the branch-and-cut process using the Vipr format [13], but since Vipr currently does not provide the necessary functionality to verify presolving reductions, ExactSCIP only prints the certificate for the presolved problem.

As a first step, PaPILO 2.2 provides the ability to generate proofs for presolving of binary programs in the VeriPB format[3], which was developed for *Pseudo-Boolean* (PB) problems [11, 41]. VeriPB readily supports a *reverse unit propagation* rule and a *redundancy-based strengthening* rule for verifying dual arguments, and in our effort to certify presolving transformations, it has recently been extended by an *objective update rule* to support modification of the objective during presolving. We refer to [49] for a detailed explanation of how each presolving reduction in PaPILO can be certified using the VeriPB language. In order to print a certificate in PaPILO the boolean parameter `verification_with_veripb` must be set to true. Since VeriPB only supports PB problems, proof logging is currently only supported for this problem class.

Table 4 reports the performance impact of proof logging. These experiments are based on the selection of binary programs from MIPLIB 2017 [40], called MIPLIB 01 [22], and the instances of the PB16 competition [79], each split into optimization (opt) and decision (dec) instances. We only exclude the large-scale instances ivu06-big and supportcase11 with a runtime of more than 2 hours in PaPILO. Times are aggregated using the geometric mean shifted by 1 second. The overhead of proof logging ranges from 27% to 54% on both test sets. For 99% of the decision instances, the *overhead per applied transaction* is below 0.186 milliseconds on both test sets. This shows the viability of proof logging in practice especially considering that proof logging runs sequentially while the presolvers in PaPILO run in parallel.

| Test set | instances | default (in seconds) | w/proof log (in seconds) | relative |
|---|---|---|---|---|
| PB16-dec | 1398 | 0.050 | 0.077 | 1.54 |
| MIPLIB 01-dec | 295 | 0.498 | 0.631 | 1.27 |
| PB16-opt | 532 | 0.439 | 0.565 | 1.29 |
| MIPLIB 01-opt | 144 | 0.337 | 0.473 | 1.40 |

**Table 4:** Runtime comparison of PaPILO with and without proof logging.

# 6 Interfaces

## 6.1 AMPL

The AMPL[4] interface of SCIP now supports parameters specified in AMPL command scripts via option `scip_options`. The value of `scip_options` is expected to be a sequence

[3] available at `https://gitlab.com/MIAOresearch/software/VeriPB`

[4] `https://www.ampl.com`

of parameter names and values, separated by a space, for example

```
option scip_options 'limits/time 10 display/verblevel 1';
```

### 6.2 JSCIPOpt

The Java interface to SCIP, JSCIPOpt, is again actively maintained and requires a C++ compiler (rather than just a C compiler) to compile. The following functionality of the SCIP C API (or the `objscip` C++ API) is newly available from Java:

- branch priorities,
- concurrent solving,
- changing the objective coefficient of a variable (contributed by the GitHub user xunzhang (Hong Wu)),
- message handlers (using the `objscip` API – the C++ interface class can be transparently subclassed from Java),
- getting the current dual bound,
- getting the solving status (contributed by the GitHub user patrickguenther),
- interrupting the solving process,
- creating a partial solution (contributed by the GitHub user fuookami (Sakurakouji Sakuya)).

In addition, the following bugs were fixed:

- running SWIG during the compilation only worked on *nix systems due to the unnecessary use of the external POSIX command `mv`,
- building against SCIP 8.0.0 or newer was failing due to a missing `#include` statement,
- even when defining `SCIP_DIR` at build time, a different SCIP could be silently used if the passed `SCIP_DIR` was not valid for some reason (this is now an error),
- a JSCIPOpt library dynamically linked to the SCIP library was not binary-compatible with different versions of SCIP due to the use of macros hardcoding structure layouts (macros now are avoided unless SCIP is statically linked into JSCIPOpt),
- the `SCIP_Longint` type was incorrectly mapped to a 32-bit Java type (now correctly mapped to a 64-bit Java type).

### 6.3 PySCIPOpt

The Python interface to SCIP, PySCIPOpt [69], is now automatically shipped with a standard installation of SCIP when installed using PyPI. This automatic SCIP installation is currently available only for machines running x86_64 architecture. The Python versions and OS combinations supported include CPython 3.6+ for manylinux2014 (includes Ubuntu / Debian) and MacOS, and CPython 3.8+ for Windows. Linking PySCIPOpt against a custom installation of SCIP is still possible and encouraged, however now requires cloning PySCIPOpt's repository, available on GitHub [77], and installing from source.

A new Python package, PySCIPOpt-ML [93], is now available. The package uses PySCIPOpt to automatically formulate machine learning models into MIPs. This functionality allows users to easily optimize MIPs with embedded ML constraints, simplifying the process of deciding on a formulation and extracting the relevant information from the machine learning model.

### 6.4 SCIP.jl

SCIP.JL is the Julia interface to SCIP and provides access to the solver in two ways. First, it provides an access to all functions of the C interface mirrored by CLANG.JL and accessed via Julia `ccall`. Second, it exposes a high-level interface implementing the MATHOPTINTERFACE API [62] and callable, e.g., through the JUMP modeling language [67]. The high-level interface now includes an access to the heuristic, branching, and cut selection plugins, making them available in an idiomatic Julia style, in addition to the constraint handler and separator plugins. The heuristic and cut generation plugins are also available through the standardized MATHOPTINTERFACE callback mechanism.

### 6.5 russcip

With SCIP 9.0, we introduce the first version of the Rust interface for SCIP, RUSSCIP [80]. The interface builds on Rust's solid foundation for type and memory safety. Being a system's programming language, it allows for low-level access to the C-API of SCIP, binding directly without the need for copying data across the language barrier. The interface is split into two parts: an unsafe part, which provides full access to the C-API through the module `ffi` and a limited but safe wrapper that allows access to part of the API. Currently, the following plugins are implemented on the safe interface that allow you to addi custom branching rules, primal heuristics, and variable pricers, and control SCIP through event handlers, all designed to guarantee compliance with SCIP's return types at compile-time.

The interface is still in its early stages and we are working on adding more plugins and improving the ergonomics of the safe interface. The following is a small example of how to use RUSSCIP:

```
use russcip::prelude::*;

fn main() {
    // Create model
    let mut model = Model::default()
    .hide_output()
    .set_obj_sense(ObjSense::Maximize);

    // Add variables
    let x1 = model.add_var(0., f64::INFINITY, 3., "x1", VarType::Integer);
    let x2 = model.add_var(0., f64::INFINITY, 4., "x2", VarType::Integer);

    // Add constraint "c1": 2 x1 + x2 <= 100
    model.add_cons(vec![x1.clone(), x2.clone()],
     &[2., 1.], -f64::INFINITY, 100., "c1");

    let solved_model = model.solve();

    let status = solved_model.status();
    println!("Status:{:?}", status);

    let obj_val = solved_model.obj_val();
    println!("Objective:{}", obj_val);

    let sol = solved_model.best_sol().unwrap();
    let vars = solved_model.vars();
```

```
    for var in vars {
        println!("{}={}",&var.name(), sol.val(var));
    }
}
```

Further examples related to defining and using custom plugins can be found in the repository's [80] tests.

### 6.6 SCIP++

SCIP++ is a C++ wrapper for SCIP's C interface. It automatically manages the memory, provides a simple interface to create linear expressions and inequalities, and provides type-safe methods to set parameters. It can be used in combination with SCIP's C interface, especially for features not yet present in SCIP++. The following is a small example.

```
#include <scippp/model.hpp>
using namespace scippp;
int main() {
    Model model("Simple");
    auto x1 = model.addVar("x_1", 1);
    auto x2 = model.addVar("x_2", 1);
    model.addConstr(3 * x1 + 2 * x2 <= 1, "capacity");
    model.setObjsense(Sense::MAXIMIZE);
    model.solve();
}
```

### 6.7 PySoPlex

PySoPlex [78] is a newly-introduced Python wrapper for SoPlex's C interface. The installation process is similar to that of PySCIPOpt, so, a user needs to install SoPlex first, set the `SOPLEX_DIR` environment variable, and then install the PySoPlex wrapper. The installation process has been successfully tested on Linux and Mac OS platforms. The following is a small example.

```
import pytest
from pysoplex import Soplex, INTPARAM, BOOLPARAM, VERBOSITY

# create solver instance
s = Soplex()

# read instance file, solve LP, and get objective value
success = s.readInstanceFile("PATH_TO_INSTANCE.mps.gz")
# specify "lifting" parameter
s.setBoolParam(BOOLPARAM.LIFTING, 1)
# specify "verbosity" level
s.setIntParam(INTPARAM.VERBOSITY, VERBOSITY.ERROR)
s.optimize()
obj_val = s.getObjValueReal()
print(obj_val)
```

# 7 The UG Framework

UG was originally designed to parallelize powerful state-of-the-art branch-and-bound based solvers (we call these "*base solvers*"). Two of the most intensively developed parallel solvers are FiberSCIP (for a shared memory computing environment) and ParaSCIP (for a distributed computing environment), both using SCIP as the base solver. ParaSCIP solved two open instances on a supercomputer from MIPLIB (MIPLIB2003) for the first time in 2010 [85]. To achieve this, supercomputer jobs had to be restarted frequently from snapshots of the branch-and-bound tree. To verify the results, we aimed to solve the instances with a single job on the supercomputer, which required the development of new features and intense debugging of ParaSCIP. Since debugging on distributed environments is inefficient, FiberSCIP was developed, which has the same parallelization algorithms as that of ParaSCIP (since UG abstracts the parallelization library), but can run on a single PC. The results of FiberSCIP were first presented in the MIPLIB2010 paper [61] (therein, FiberSCIP is referred to as UG[SCIP/SPX]). Even though FiberSCIP was already working in 2010, the FiberSCIP paper [87] was submitted only 3 years later, since the software went through intensive tests. The supplement of the FiberSCIP paper includes only a small fraction of the computational results we had conducted. Due to the major debugging effort of UG and SCIP via FiberSCIP, ParaSCIP could solve more than 20 open instances [86] from MIPLIB and none of these results have been proven wrong so far. Thus, next to its main purpose of parallelization, a major contribution of UG has been an improved stability of SCIP. For example, complete thread-safety of SCIP was only achieved due to the development of FiberSCIP.

Since UG provides a systematic way to parallelize a state-of-the-art sequential or multi-threaded solver to run on a large scale distributed memory environment, with version 1.0, UG is generalized to a software framework for a *high-level task parallelization framework*[5]. That is, with version 1.0, UG will not only parallelize the tree search of branch-and-bound based solvers, but allow the parallelization of other kind of solvers. On top of that, UG version 1.0 will also allow more flexibility and customization when parallelizing a branch-and-bound based solver for a specific purpose. For an example, see the recent adaptation CMAP-LAP (Configurable Massively Parallel solver framework for LAttice Problems) of UG to solve lattice problems [89].

With the new beta version of UG 1.0, which is released with the SCIP Optimization Suite 9.0, UG has caught up with interface changes in SCIP and includes a few more bugfixes. It does not include many new features. However, the possibility to appropriately specify an optimality gap limit has been added.

## 7.1 Setting optimality gap limit

For FiberSCIP/ParaSCIP, SCIP parameters can be set by using command line options,"-sl, -sr, -s" as below:

```
../bin/fscip
syntax: ../bin/fscip fscip_param_file problem_file_name [-l <logfile>] [-q]
      [-sl <settings>]  [-s <settings>] [-sr <root_settings>] [-w <prefix_warm>]
      [-sth <number>] [-fsol <solution_file>] [-isol <initial solution file]
-l <logfile>           : copy output into log file
-q                     : suppress screen messages
-sl <settings>         : load parameter settings (.set) file for LC presolving
-s <settings>          : load parameter settings (.set) file for solvers
```

[5]For concept of UG's high-level task parallelization framework, see `https://ug.zib.de/doc-1.0.0/html/CONCEPT.php`

```
-sr <root_settings>    : load parameter settings (.set) file for root
-w <prefix_warm>       : warm start file prefix ( prefix_warm_nodes.gz and
                         prefix_warm_solution.txt are read )
-sth <number>          : the number of solver threads used
-fsol <solution file> : specify output solution file
-isol <intial solution file> : specify initial solution file
```

Therefore, parallel solving algorithms of FiberSCIP/ParaSCIP can be controlled just by setting these parameters. The optimality gap limit could be set by using the "-s" option. However, using default settings, FiberSCIP/ParaSCIP executes presolving in the LoadCoordinator, which is the controller thread (process) of FiberSCIP (ParaSCIP, respectively), and the presolved instance is passed on to all solvers [87]. Therefore, the "-s" option applies to solving the presolved instance, not the original one. This can be a problem when trying to set a gap limit, for instance. With a previous release, a UG parameter was added to handle this appropriately, but it turned out to not work well and has now been removed again. Instead, with this version, an optimality gap limit can be set for the original instance. To do so, the gap limit should be set in the SCIP parameter setting file that is specified with the "-sl" option. The LoadCoordinator will then handle the gap limit appropriately.

## 8 The GCG Decomposition Solver

GCG is an extension that turns SCIP into a branch-and-price or branch-and-Benders-cut solver for mixed-integer linear programs. GCG can automatically detect a model structure that allows for a Dantzig-Wolfe reformulation or Benders decomposition. The reformulation process and the corresponding algorithmics like Benders cut and column generation is done automatically without interaction from the users. They just need to provide the model. The latest version is GCG 3.6. Here are the few changes since version 3.5 upon which we reported along with SCIP version 8.0.

GCG 3.6 mainly contains code base improvements, with no major algorithmic changes. Most importantly for developers, the API has mildly changed: The prefix `DEC_` was replaced with `GCG_` to achieve a consistent naming.

The model structures detected by GCG are called decompositions. The detection process itself was described in the SCIP 6.0 release report. Part of this process is based on classifiers which group constraints and variables for their potential roles in a decomposition. From the usually many decompositions found, users can select manually or let GCG select based on different scores. The score implementation was refactored: Previously, it was cumbersome to add user-defined scores to GCG, but with GCG 3.6, new scores can be added as plugins. Each score must provide a function that calculates a score value for a (partial) decomposition. The macro `GCG_DECL_SCORECALC` is provided to declare the method that implements the scoring and is automatically called by GCG.

The `display` dialog command has been extended and can now be used to print information about registered scores and classifiers. Furthermore, GCG 3.6 now supports compiling with Microsoft Visual C++ (MSVC) and it is possible to use CMake to simplify the build process.

The Python interface PyGCGOpt mirrors the above changes. Users can now customize the detection process even more by adding own classifiers and scores.

# 9 SCIP-SDP

SCIP-SDP is a solver for handling mixed-integer semidefinite programs, w.l.o.g, written in the following form

$$\begin{aligned} \inf \quad & b^\top y \\ \text{s.t.} \quad & \sum_{k=1}^{m} A^k y_k - A^0 \succeq 0, \\ & \ell_i \leq y_i \leq u_i \qquad\qquad \forall\, i \in [m], \\ & y_i \in \mathbb{Z} \qquad\qquad\quad \forall\, i \in I, \end{aligned} \tag{8}$$

with symmetric matrices $A^k \in \mathbb{R}^{n\times n}$ for $i \in \{0,\ldots,m\}$, $b \in \mathbb{R}^m$, $\ell_i \in \mathbb{R}\cup\{-\infty\}$, $u_i \in \mathbb{R}\cup\{\infty\}$ for all $i \in [m] \coloneqq \{1,\ldots,m\}$. The set of indices of integer variables is given by $I \subseteq [m]$, and $M \succeq 0$ denotes that a matrix $M$ is positive semidefinite.

SCIP-SDP uses an SDP-based branch-and-bound approach based on SCIP (default). It also supports the possibility to use linear inequalities in an LP-based approach. Which one is faster, depends on the instance.

The development of SCIP-SDP proceeded along a series of dissertations: Mars [71], Gally [29], and Matter [72]. Corresponding articles are [30] (existence of Slater points in branch-and-bound, dual fixing) and [73] (presolving techniques). In the following, we give an overview of the main changes since version 4.0.0, which was reported on in the SCIP 8 report. The current version of SCIP-SDP is 4.3.0.

## 9.1 Symmetry Handling of MISDPs

One can handle permutation symmetries of (8) in the sense of Section 3.1. We will sketch how this works and refer to [52] for details.

To define symmetries of (8), for a matrix $A \in \mathbb{R}^{n\times n}$ and a permutation $\sigma$ of $[n]$, let

$$\sigma(A)_{ij} \coloneqq A_{\sigma^{-1}(i),\sigma^{-1}(j)} \quad \forall\, i,\, j \in [n].$$

**Definition 9.1.1.** *A permutation $\pi$ of variable indices $[m]$ is a* formulation symmetry *of (8) if there exists a permutation $\sigma$ of the dimensions $[n]$ such that*

1. *$\pi(I) = I$, $\pi(\ell) = \ell$, $\pi(u) = u$, and $\pi(b) = b$*
   *($\pi$ leaves integer variables, variable bounds, and the objective coefficients invariant),*
2. *$\sigma(A^0) = A^0$ and, for all $i \in [m]$, $\sigma(A^i) = A^{\pi^{-1}(i)}$.*

Such symmetries can be detected by using graph automorphism algorithms, see [52]. Examples of (formulation) symmetries computed for a testset of MISDP instances can also be found in [52]. Note that we do not exploit symmetries in the matrix solutions of the SDPs like it has been done in [35, 53], for example.

Table 5 presents computational results – we refer to [52] for the setup and details. We observe a speed-up of about 4.00 % for all instances and of about 34.00 % for the 21 instances that contain symmetry.

Since the appearance of [52], SCIP-SDP has been changed to allow for using the callback access to symmetry computation in SCIP, see Section 3.1.2. Thereby, we have also changed the symmetry detection graph. It now only contains a single node for each nonzero entry of the matrix that is connected to the 'dimension'-nodes. The corresponding graph for the following MISDP is given in Figure 1.

$$\inf\left\{y_1 + y_2 \,:\, \begin{pmatrix} 3&1&0\\1&1&0\\0&0&0 \end{pmatrix} y_1 + \begin{pmatrix} 0&0&0\\0&1&1\\0&1&3 \end{pmatrix} y_2 - \begin{pmatrix} 1&0&0\\0&1&0\\0&0&1 \end{pmatrix} \succeq 0,\ 0 \leq y_1,\, y_2 \leq 1,\ y_1,\, y_2 \in \mathbb{Z}\right\}.$$

**Table 5:** Results on a testset of 184 MISDP instances with/without using symmetry handling in SCIP. We report the shifted geometric means of the running times in seconds and number of nodes. Column "symtime" and "# gens" report the averge time for symmetry handling (including detection) and number of generators, respectively. The "all optimal" block reports results for the 168 instances that were solved by both methods. The last column gives the shifted geometric mean running time only for the 21 instances that contain some symmetry.

| | all (184) | | | all optimal (168) | | only symmetric (21) |
|---|---|---|---|---|---|---|
| | time (s) | symtime (s) | # gens | time (s) | #nodes | time (s) |
| without | 130.6 | – | – | 95.0 | 778.3 | 45.07 |
| with | 125.3 | 0.44 | 99 | 90.8 | 760.6 | 29.84 |

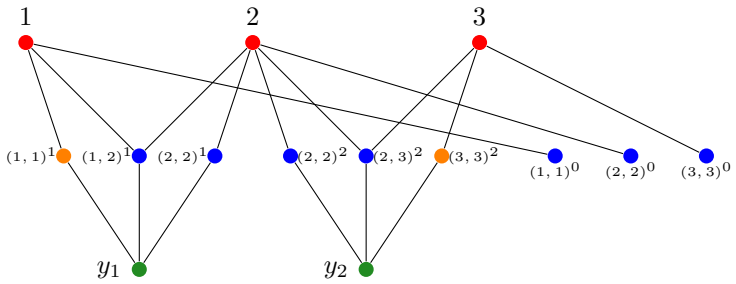



**Figure 4:** Illustration of symmetry detection graph.

In Figure 4, the colors of the nodes allow to distinguish different types and values. The topmost nodes represent the dimensions. Node $(i,j)^k$ represents the symmetric entries $(i,j)$ and $(j,i)$ (or diagonal entry $(i,i)$) of matrix $A^k$. The colors of these nodes correspond to distinct coefficients in the matrices. The only non-trivial color-preserving automorphism of the graph exchanges $y_1 \leftrightarrow y_2$, $(1,1)^1 \leftrightarrow (3,3)^2$, $(1,2)^1 \leftrightarrow (2,3)^2$, $(2,2)^1 \leftrightarrow (2,2)^2$, $(1,1)^0 \leftrightarrow (3,3)^0$, $1 \leftrightarrow 3$, and keeps node 2 fixed. This leads to the variable permutation $\pi$, which exchanges $y_1$ and $y_2$, and the matrix permutation $\sigma$, which exchanges 1 and 3.

### 9.2 Conflict Analysis for MISDPs

The original idea of conflict analysis was to learn from infeasible nodes in a branch-and-bound-tree. To this end, Achterberg [1] transferred ideas from SAT-solving to MILPs. One further way is to try to learn cuts from solutions of the duals, which is called "dual ray/solution analysis" in Witzig et al. [98] and Witzig [97].

To briefly explain the application to MISDPs, consider the SDP relaxation of (8). Given a positive semidefinite $\hat{X} \in \mathbb{R}^{n\times n}$, we observe that the inner product with a positive semidefinite matrix $M \in \mathbb{R}^{n\times n}$ is nonnegative:

$$\langle \hat{X}, M \rangle \coloneqq \sum_{i,j=1}^{n} \hat{X}_{ij} M_{ij} \geq 0.$$

Thus, defining $A(y) \coloneqq \sum_{k=1}^{m} A^k y_i$, we get

$$\langle \hat{X}, A(y) \rangle = \sum_{k=1}^{m} \langle \hat{X}, A^k \rangle y_k \geq 0 \tag{9}$$

for every feasible solution $y$ of (8). Note that this is a (redundant) linear inequality in $y$. The idea is to use it in the propagation of variable bounds and not explicitly add it to (8).

There are two natural ways to obtain good candidates for $\hat{X}$. If the relaxation is feasible, we obtain a solution $(\hat{X}, \hat{r}^\ell, \hat{r}^u)$ of the dual

$$\begin{aligned} \sup \quad & \langle A^0, X\rangle + \ell^\top r^\ell - u^\top r^u \\ \text{s.t.} \quad & \langle A^j, X\rangle + r^\ell_j - r^u_j = b_j \quad \forall\, j \in [m], \\ & X \succeq 0,\ r^\ell,\ r^u \geq 0. \end{aligned}$$

Similarly, if the relaxation is infeasible and a constraint qualification holds, one can obtain a dual ray satisfying

$$\begin{aligned} & \langle A^j, X\rangle + r^\ell_j - r^u_j = 0 \qquad\qquad \forall\, j \in [m], \\ & \langle A^0, X\rangle + \ell^\top r^\ell - u^\top r^u > 0, \\ & X \succeq 0,\ r^\ell,\ r^u \geq 0. \end{aligned}$$

One can prove that (9) is infeasible with respect to the local bounds $\ell$ and $u$ and can therefore provide a proof of infeasibility, see [76].

SCIP-SDP generates a conflict constraint (9) for each feasible or infeasible node, stores them as constraints, and performs bound propagation. This leads to a speed-up and node reduction of about 8% on the same testset used in the previous section. We refer to [76] for more details.

### 9.3 Further Changes

Similar to SCIP, the license of SCIP-SDP has changed to Apache 2.0.

Several improvements have been made to speed up some of the presolving methods. One can use ARPACK instead of Lapack for eigenvalue computations (use `ARPACK = true` when using makefiles); this is usually slower for the typical sizes of MISDPs. Moreover, when running the LP-based approach (`misc/solvesdps = 0`), now CMIR inequalities are generated by default. To implement conflict analysis, the handling of dual solutions has been extended and improved.

## 10 Final Remarks

The SCIP Optimization Suite 9.0 release provides new functionality along with improved performance and reliability. In SCIP, the changes to the symmetry detection feature include new techniques for handling symmetries of non-binary variables, restructuring of the mechanism to detect symmetries of the custom constraints, and detection of the signed permutation symmetries. New interfaces to NAUTY [74] as well as the preprocessor SASSY [6] have been added. A new nonlinear handler for signomial functions and improvements to the existing nonlinear handler for quadratic expressions were implemented. A new diving heuristic for handling indicator constraints that are used to represent the semi-continuous variables, an extension of the dynamic partition search heuristic, and a new adaptive heuristic that dynamically adapts the application of large neighborhood search and diving heuristics to the characteristics of the current instance were also implemented. Furthermore, a new separator called the Lagromory separator for generating potentially lower-ranked cuts and reducing the dual bound stalling due to the dual degeneracy, and two new cut selection schemes were included: `cutsel/ensemble` that adapts with respect to the given instance properties and `cutsel/dynamic` that aims

to enhance the near-orthogonal threshold methodology used in the default cut selection scheme. A new branching criterion called the GMI branching was implemented. It is available both as a stand-alone rule and also integrated within the default hybrid branching rule of SCIP. It considers a new scoring component based on the GMI cuts corresponding to the fractional variable in a given LP solution. Finally, a new interface to the HiGHS LP solver along with technical improvements to the AMPL reader and OBBT propagator were implemented.

Regarding usability, various interfaces were improved and new interfaces were added. The AMPL interface to SCIP was extended to support parameters from the AMPL command scripts. The Java interface to SCIP, JSCIPOpt, is being maintained actively and was extended with new functionality. The Python interface to SCIP, PySCIPOpt, can now be fully installed using PyPI. A new Python package, PySCIPOpt-ML, is available to automatically formulate machine learning models into MIPs. The Julia interface to SCIP, SCIP.jl, was extended to be able to access additional plugins of SCIP. Two new interfaces to SCIP, Rust interface called russcip and C++ interface called SCIP++, are also available, along with a new Python interface to SoPlex called PySoPlex.

The LP solver SoPlex now supports incremental precision boosting for exact LP solving over the rational numbers. It is available as a stand-alone as well as in combination with the existing LP iterative refinement approach. The presolving library PaPILO now has a new feature called proof logging that allows the generation of machine-verifiable certificates for presolving of binary problems to be able to prove the correctness of the computations. The parallel framework UG now has a new beta version of UG 1.0 that includes the latest interface changes of SCIP along with new bugfixes. A new feature to appropriately set the gap limit has also been added. The GCG decomposition solver now includes improvements to its code base, can be compiled with Microsoft Visual C++ (MSVC), and supports CMake as a build system. The SCIP extension SCIP-SDP has been improved to include new symmetry handling techniques and conflict analysis for MISDPs, along with other improvements in its presolving methods and cut generation techniques.

These developments yield an overall performance improvement of both MILP and MINLP benchmarking instances. SCIP 9.0 is able to solve 19 more MILP instances as compared to SCIP 8.0 with a speedup of 2% on the affected instances. This speedup further increases to 6% when only the hard instances requiring at least 1000 seconds by at least one setting are considered. The number of nodes required for MILPs that were solved by both versions of SCIP also reduce considerably by 17% in SCIP 9.0. These performance gains are more prominent for MINLPs. SCIP 9.0 solves 5 more MINLP instances as compared to SCIP 8.0 with performance improvements of 4% in time (for all MINLPs) and 13% in the number of nodes (for the MINLPs that were solved by both the versions of SCIP). When looking at hard instances requiring at least 1000 seconds by at least one setting, the gains are further increased to 20% and 46% in time and number of nodes, respectively. Furthermore, when restrcited to nonconvex instances only, SCIP 9.0 is faster by 8%. Hence, SCIP 9.0 has become faster and more reliable as compared to SCIP 8.0.

**Acknowledgements**

The authors want to thank all previous developers and contributors to the SCIP Optimization Suite and all users that reported bugs and often also helped reproducing and fixing the bugs. Thanks to Herman Appelgren, Gerald Gamrath, Stephen J. Maher, Daniel Rehfeldt, and Michael Winkler, for various contributions. Thanks to Julian Hall for supporting the creation of the HiGHS LP interface.

**Contributions of the Authors**

The material presented in the article is highly related to code and software. In the following, we try to make the corresponding contributions of the authors and possible contact points more transparent.

- JvD implemented the generalization of symmetry handling methods for binary variables to general variables (Section 3.1.1).
- CH implemented the symmetry detection framework via symmetry detection graph callbacks in constraint handlers (Section 3.1.2).
- MP implemented the interface to NAUTY, and the interface to allow SASSY as preprocessor for BLISS (Section 3.1.3).
- MP and GM implemented the interface to allow SASSY as preprocessor for NAUTY (Section 3.1.3).
- LX implemented the nonlinear handler for signomial functions (Section 3.2.1).
- AC and FeS implemented monoidal strengthening for quadratic constraints (Section 3.2.2).
- KH and AH implemented the indicator diving heuristic (Section 3.3.1).
- KH extended the DPS heuristic (Section 3.3.2).
- AC implemented the online scheduling procedure of primal heuristics (Section 3.3.3).
- SB implemented the Lagromory separator (Section 3.4.1).
- MT and MB worked on the ensemble cut selector (Section 3.4.2).
- CG worked on cut statistics and dynamic cut selection (Section 3.4.2).
- MT and MB worked on the GMI branching and hybrid branching improvements (Section 3.5.1 and Section 3.5.2).
- AG, GM, and AH developed the HiGHS LP interface (Section 3.6).
- SV extended the AMPL `.nl` reader (Section 3.7).
- KH extended the OBBT propagator (Section 3.7).
- LE worked on the SoPlex solver (Section 4).
- AH implemented the proof-logging in PaPILO (Section 5).
- SV extended the AMPL interface (Section 6.1).
- KK is the current maintainer of JSCIPOpt and has worked on the developments listed in Section 6.2 (other than the ones where different developer names are mentioned).
- JD actively maintained and contributed to PySCIPOpt (Section 6.3).
- MT worked on the pip installation of PySCIPOpt through PyPI (Section 6.3).
- MB worked on the Julia interface SCIP.jl (Section 6.4).
- MG implemented the RUSSCIP interface (Section 6.5).
- IH programmed the initial release of SCIP++ (Section 6.6).
- SB implemented the PySoPlex interface (Section 6.7).
- YS worked on the UG framework (Section 7)
- Concerning GCG (Section 8), EM and JL refactored and improved the code (general API changes and scores) and added the MSVC support; JL and TD extended PyGCGOpt.
- CH and MP implemented symmetry handling for SCIP-SDP and MP implemented conflict analysis in SCIP-SDP (Section 9).

– RvdH contributed to various bug fixes and file reader updates.
– DK fixed several bugs and contributed to other fixes regarding reliability. Furthermore, DK implemented an algorithm to simplify the debugging process by generating a potentially smaller-sized instance and settings that can reproduce the bugs faster.
– JM and FrS worked on the continuous integration system, regular testing, binary distributions, and website development. JM is the contact person for these aspects.

## References


[1] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Opt.*, 4(1):4–20, 2007. doi:10.1016/j.disopt.2006.10.006.

[2] T. Achterberg. *Constraint Integer Programming.* Dissertation, Technische Universität Berlin, 2007.

[3] T. Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi:10.1007/s12532-008-0001-1.

[4] T. Achterberg and T. Berthold. Hybrid branching. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 6th International Conference, Proceedings of CPAIOR 2009 Pittsburgh, PA, USA, 2009*, pages 309–311. Springer, 2009.

[5] Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Nightingale. Metamorphic testing of constraint solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *Lecture Notes in Computer Science*, pages 727–736. Springer, 2018.

[6] M. Anders, P. Schweitzer, and J. Stieß. Engineering a preprocessor for symmetry detection. In L. Georgiadis, editor, *21st International Symposium on Experimental Algorithms, SEA 2023, July 24-26, 2023, Barcelona, Spain*, volume 265 of *LIPIcs*, pages 1:1–1:21, 2023. doi:10.4230/LIPICS.SEA.2023.1.

[7] K. Andersen, G. Cornuéjols, and Y. Li. Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts. *Management Science*, 51(11):1720–1732, 2005.

[8] E. Balas and R. G. Jeroslow. Strengthening cuts for mixed integer programs. *European Journal of Operational Research*, 4(4):224–234, 1980.

[9] P. Bendotti, P. Fouilhoux, and C. Rottner. Orbitopal fixing for the full (sub-)orbitope and application to the unit commitment problem. *Mathematical Programming*, 186:337–372, 2021. doi:10.1007/s10107-019-01457-1.

[10] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, 2022. `https://optimization-online.org/?p=18429`.

[11] B. Bogaerts, S. Gocht, C. McCreesh, and J. Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77: 1539–1589, 2023.

[12] V. F. Cavalcante, C. C. de Souza, and A. Lucena. A relax-and-cut algorithm for the set partitioning problem. *Computers & operations research*, 35(6):1963–1981, 2008.

[13] K. K. H. Cheung, A. M. Gleixner, and D. E. Steffy. Verifying integer programming results. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO '17)*, volume 10328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2017.

[14] A. Chmiela, P. Lichoki, A. Gleixner, and S. Pokutta. Online learning for scheduling MIP heuristics. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 114–123. Springer, 2023.

[15] A. Chmiela, G. Muñoz, and F. Serrano. Monoidal strengthening and unique lifting in MIQCPs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 87—-99. Springer, 2023.

[16] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5 (3):305–344, 2013.

[17] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical programming*, 112(1):3–44, 2008.

[18] G. Cornuéjols, F. Margot, and G. Nannicini. On the safety of Gomory cut generators. *Mathematical Programming Computation*, 5:345–395, 2013.

[19] A. Costa, P. Hansen, and L. Liberti. On the impact of symmetry-breaking constraints on spatial branch-and-bound for circle packing in a square. *Discrete Applied Mathematics*, 161(1):96–106, 2013. doi:10.1016/j.dam.2012.07.020.

[20] L. Cruz-Filipe, M. J. H. Heule, W. A. Hunt Jr., M. Kaufmann, and P. Schneider-Kamp. Efficient certified RAT verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017.

[21] L. Cruz-Filipe, J. P. Marques-Silva, and P. Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135. Springer, 2017.

[22] J. Devriendt. MIPLIB 0-1 instances in OPB format. Dataset on Zenodo, 2020.

[23] L. Eifler and A. Gleixner. A computational status update for exact rational mixed integer programming. In M. Singh and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, pages 163–177, Cham, 2021. Springer International Publishing.

[24] L. Eifler and A. Gleixner. Safe and verified gomory mixed integer cuts in a rational MIP framework. *SIAM Journal on Optimization*, 2023. accepted for publication.

[25] L. Eifler, J. Nicolas-Thouvenin, and A. Gleixner. Combining precision boosting with LP iterative refinement for exact linear optimization. Preprint 2311.08037, arXiv, 2023. URL `https://arxiv.org/abs/2311.08037`.

[26] D. G. Espinoza. *On Linear Programming, Integer Programming and Cutting Planes*. PhD thesis, Georgia Institute of Technology, 2006.

[27] Exact SCIP. A development version of scip with exact rational arithmetic. `https://github.com/scipopt/scip/tree/exact-rational`, 2024.

[28] M. Fischetti and D. Salvagnin. A relax-and-cut framework for Gomory mixed-integer cuts. *Mathematical Programming Computation*, 3:79–102, 2011.

[29] T. Gally. *Computational Mixed-Integer Semidefinite Programming*. Dissertation, TU Darmstadt, 2019.

[30] T. Gally, M. E. Pfetsch, and S. Ulbrich. A framework for solving mixed-integer semidefinite programs. *Optimization Methods and Software*, 33(3):594–632, 2018. doi:10.1080/10556788.2017.1322081.

[31] G. Gamrath and M. E. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 239–252. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13193-6_21.

[32] G. Gamrath, T. Koch, S. J. Maher, D. Rehfeldt, and Y. Shinano. SCIP-Jack—a solver for STP and variants with parallelization extensions. *Mathematical Programming Computation*, 9(2):231–296, 2017. doi:10.1007/s12532-016-0114-x.

[33] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. L. Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, 2020. `http://www.optimization-online.org/DB_HTML/2020/03/7705.html`.

[34] G. Gamrath, T. Berthold, and D. Salvagnin. An exploratory computational analysis of dual degeneracy in mixed-integer programming. *EURO Journal on Computational Optimization*, 8(3-4):241–261, 2020.

[35] K. Gatermann and P. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. *Journal of Pure and Appl. Algebra*, 192(1-3):95–128, 2004.

[36] X. Gillard, P. Schaus, and Y. Deville. SolverCheck: Declarative testing of constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2019.

[37] A. Gleixner and D. E. Steffy. Linear programming using limited-precision oracles. *Mathematical Programming*, 183:525–554, 2020. doi:10.1007/s10107-019-01444-6.

[38] A. Gleixner, D. E. Steffy, and K. Wolter. Iterative refinement for linear programming. *INFORMS Journal on Computing*, 28(3):449–464, 2016. doi:10.1287/ijoc.2016.0692.

[39] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 6.0. Technical report, Optimization Online, 2018. URL `http://www.optimization-online.org/DB_HTML/2018/07/6692.html`.

[40] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. Mittelmann, D. Ozyurt, T. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017: Data-driven compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 13: 443–490, 2021. doi:10.1007/s12532-020-00194-3.

[41] S. Gocht, R. Martins, J. Nordström, and A. Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, 2022.

[42] R. Gomory. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

[43] M. Guignard. Efficient cuts in Lagrangean 'relax-and-cut' schemes. *European Journal of Operational Research*, 105(1):216–223, 1998.

[44] K. Halbig, A. Göß, and D. Weninger. Exploiting user-supplied decompositions inside heuristics. Technical report, Optimization Online, 2023. URL `https://optimization-online.org/?p=23386`.

[45] G. Hendel. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation*, 14(2):185–221, 2022.

[46] G. Hendel, M. Miltenberger, and J. Witzig. Adaptive algorithmic behavior for solving mixed integer programs using bandit algorithms. In *International Conference on Operations Research*, pages 513–519. Springer, 2018.

[47] M. J. H. Heule, W. A. Hunt Jr., and N. Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, 2013.

[48] M. J. H. Heule, W. A. Hunt Jr., and N. Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2013.

[49] A. Hoen, A. Oertel, A. Gleixner, and J. Nordström. Certifying MIP-based presolve reductions for 0-1 integer linear programs. Preprint, arXiv, 2024. URL `https://arxiv.org/abs/2401.09277`.

[50] C. Hojny. Packing, partitioning, and covering symresacks. *Discrete Applied Mathematics*, 283:689–717, 2020. doi:10.1016/j.dam.2020.03.002.

[51] C. Hojny and M. E. Pfetsch. Polytopes associated with symmetry handling. *Mathematical Programming*, 175(1):197–240, 2019. doi:10.1007/s10107-018-1239-7.

[52] C. Hojny and M. E. Pfetsch. Handling symmetries in mixed-integer semidefinite programs.

In A. A. Cire, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 69–78, Cham, 2023. Springer. doi:10.1007/978-3-031-33271-5_5.

[53] H. Hu, R. Sotirov, and H. Wolkowicz. Facial reduction for symmetry reduced semidefinite and doubly nonnegative programs. *Mathematical Programming*, 2022. To appear.

[54] Q. Huangfu and J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10:119–142, 2015. doi:10.1007/s12532-017-0130-5.

[55] T. Junttila and P. Kaski. bliss: A tool for computing automorphism groups and canonical labelings of graphs. `http://www.tcs.hut.fi/Software/bliss/`, 2012.

[56] V. Kaibel and A. Loos. Finding descriptions of polytopes via extended formulations and liftings. In A. R. Mahjoub, editor, *Progress in Combinatorial Optimization*. Wiley, 2011.

[57] V. Kaibel and M. E. Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming*, 114(1):1–36, 2008. doi:10.1007/s10107-006-0081-5.

[58] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8 (4):595–610, 2011. doi:10.1016/j.disopt.2011.07.001.

[59] E. Klotz. Identification, assessment, and correction of ill-conditioning and numerical instability in linear and integer programs. In A. Newman and J. Leung, editors, *Bridging Data and Decisions*, TutORials in Operations Research, pages 54–108. INFORMS, 2014. doi:10.1287/educ.2014.0130.

[60] T. Koch. *Rapid Mathematical Prototyping*. Dissertation, Technische Universität Berlin, 2004.

[61] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. Miplib 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi:10.1007/s12532-011-0025-9. URL `https://doi.org/10.1007/s12532-011-0025-9`.

[62] B. Legat, O. Dowson, J. D. Garcia, and M. Lubin. MathOptInterface: a data structure for mathematical optimization problems. *INFORMS Journal on Computing*, 34(2):672–689, 2022.

[63] L. Liberti. Symmetry in mathematical programming. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *IMA Series*, pages 236–286. Springer, New York, 2012.

[64] L. Liberti and J. Ostrowski. Stabilizer-based symmetry breaking constraints for mathematical programs. *Journal of Global Optimization*, 60:183–194, 2014. doi:10.1007/s10898-013-0106-6.

[65] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL `http://jmlr.org/papers/v23/21-0888.html`.

[66] A. Loos. *Describing Orbitopes by Linear Inequalities and Projection Based Tools*. Dissertation, Otto-von-Guericke-Universität Magdeburg, 2010.

[67] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat, and J. P. Vielma. Jump 1.0: recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, pages 1–9, 2023.

[68] A. Lucena. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140: 375–410, 2005.

[69] S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano. PySCIPOpt: Mathematical programming in Python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi:10.1007/978-3-319-42432-3_37.

[70] F. Margot. Exploiting orbits in symmetric ILP. *Mathematical Programming*, 98(1–3):3–21, 2003. doi:10.1007/s10107-003-0394-6.

[71] S. Mars. *Mixed-Integer Semidefinite Programming with an Application to Truss Topology Design*. Dissertation, FAU Erlangen-Nürnberg, 2013.

[72] F. Matter. *Sparse Recovery Under Side Constraints Using Null Space Properties*. Dissertation, TU Darmstadt, 2022.

[73] F. Matter and M. E. Pfetsch. Presolving for mixed-integer semidefinite optimization. *INFORMS J. Opt.*, 5(2):131–154, 2022. doi:10.1287/ijoo.2022.0079.

[74] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

[75] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. Orbital branching. *Mathematical Programming*, 126(1):147–178, 2011. doi:10.1007/s10107-009-0273-x.

[76] M. E. Pfetsch. Dual conflict analysis for mixed-integer semidefinite programs. Preprint, Optimization Online, 2023. `https://optimization-online.org/?p=23813`.

[77] PySCIPOpt. The Python interface for SCIP. `https://www.github.com/scipopt/PySCIPOpt`, 2017.

[78] PySoPlex. The python interface for SoPlex. `https://www.github.com/scipopt/PySoPlex`, 2023.

[79] O. Roussel. Pseudo-Boolean competition 2016, 2016. URL `http://www.cril.univ-artois.fr/PB16/`.

[80] russcip. The Rust interface for SCIP. `https://www.github.com/scipopt/russcip`, 2023.

[81] D. Salvagnin. A dominance procedure for integer programming. Master's thesis, Universtà degli studi di Padova, 2005.

[82] D. Salvagnin. Symmetry breaking inequalities from the Schreier-Sims table. In W.-J. van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 521–529. Springer, 2018. doi:10.1007/978-3-319-93031-2_37.

[83] SCIP++. The C++ interface for SCIP. `https://www.github.com/scipopt/SCIPpp`, 2023.

[84] Y. Shinano. The Ubiquity Generator framework: 7 years of progress in parallelizing branch-and-bound. In N. Kliewer, J. F. Ehmke, and R. Borndörfer, editors, *Operations Research Proceedings 2017*, pages 143–149. Springer, 2018. doi:10.1007/978-3-319-89920-6_20.

[85] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch. Parascip: A parallel extension of scip. In C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, editors, *Competence in High Performance Computing 2010*, pages 135–148, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-24025-6.

[86] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler. Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 770–779, 2016. doi:10.1109/IPDPS.2016.56.

[87] Y. Shinano, S. Heinz, S. Vigerske, and M. Winkler. FiberSCIP: A shared memory parallelization of SCIP. *INFORMS Journal on Computing*, 30(1):11–30, 2018. doi:10.1287/ijoc.2017.0762.

[88] D. E. Steffy. *Topics in exact precision mathematical programming*. PhD thesis, Georgia Institute of Technology, 2011. URL `http://hdl.handle.net/1853/39639`.

[89] N. Tateiwa, Y. Shinano, K. Yamamura, A. Yoshida, S. Kaji, M. Yasuda, and K. Fujisawa. CMAP-LAP: Configurable massively parallel solver for lattice problems. ZIB-Report 21-16, Zuse Institute Berlin, 2021.

[90] M. Turner, T. Berthold, and M. Besançon. A context-aware cutting plane selection algorithm for mixed-integer programming. *Operations Research Proceedings (accepted)*, 2023.

[91] M. Turner, T. Berthold, M. Besançon, and T. Koch. Branching via cutting plane selection: Improving hybrid branching. *arXiv preprint arXiv:2306.06050*, 2023.

[92] M. Turner, T. Berthold, M. Besançon, and T. Koch. Cutting plane selection with analytic centers and multiregression. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 52–68. Springer, 2023.

[93] M. Turner, A. Chmiela, T. Koch, and M. Winkler. PySCIPOpt-ML: Embedding trained machine learning models into mixed-integer programs. *arXiv preprint arXiv:2312.08074*, 2023.

[94] M. Turner, T. Koch, F. Serrano, and M. Winkler. Adaptive Cut Selection in Mixed-Integer Linear Programming. *Open Journal of Mathematical Optimization*, 4:5, 2023. doi:10.5802/ojmo.25.

[95] J. van Doornmalen and C. Hojny. A unified framework for symmetry handling. preprint available at `https://optimization-online.org/?p=20822`, 2022.

[96] N. Wetzler, M. J. H. Heule, and W. A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.

[97] J. Witzig. *Infeasibilitity Analysis for MIP*. Dissertation, TU Berlin, 2021.

[98] J. Witzig, T. Berthold, and S. Heinz. Experiments with conflict analysis in mixed integer programming. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR*, volume 10335 of *LNCS*, pages 211–222. Springer, 2017. doi:10.1007/978-3-319-59776-8_17.

[99] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. Dissertation, Technische Universität Berlin, 1996.

[100] L. Xu, C. D'Ambrosio, L. Liberti, and S. H. Vanier. Cutting planes for signomial programming, 2022.

## Author Affiliations

Suresh Bolusani
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: bolusani@zib.de
ORCID: 0000-0002-5735-3443

Mathieu Besançon
Université Grenoble Alpes, Inria, LIG, 38000 Grenoble, France, and
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: besancon@zib.de
ORCID: 0000-0002-6284-3033

Ksenia Bestuzheva
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: bestuzheva@zib.de
ORCID: 0000-0002-7018-7099

Antonia Chmiela
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: chmiela@zib.de
ORCID: 0000-0002-4809-2958

João Dionísio
CMUP and Department of Computer Science, Faculty of Sciences, University of Porto, R Campo Alegre, 4169–007 Porto, Portugal
E-mail: joao.goncalves.dionisio@gmail.com
ORCID: 0009-0005-5160-0203

Tim Donkiewicz
RWTH Aachen University, Lehrstuhl für Operations Research, Kackertstr. 7, 52072 Aachen, Germany
E-mail: tim.donkiewicz@rwth-aachen.de
ORCID: 0000-0002-5721-3563

Jasper van Doornmalen
Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-mail: m.j.v.doornmalen@tue.nl
ORCID: 0000-0002-2494-0705

Leon Eifler
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: eifler@zib.de
ORCID: 0000-0003-0245-9344

Oliver Gaul
RWTH Aachen University, Lehrstuhl für Operations Research, Kackertstr. 7, 52072 Aachen, Germany
E-mail: oliver.gaul@rwth-aachen.de
ORCID: 0000-0002-2131-1911

Mohammed Ghannam
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: ghannam@zib.de
ORCID: 0000-0001-9422-7916

Ambros Gleixner
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: gleixner@zib.de
ORCID: 0000-0003-0391-5903

Christoph Graczyk
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: graczyk@zib.de
ORCID: 0000-0001-8990-9912

Katrin Halbig
Friedrich-Alexander Universität Erlangen-Nürnberg, Department of Data Science, Cauerstr. 11, 91058 Erlangen, Germany
E-mail: katrin.halbig@fau.de
ORCID: 0000-0002-8730-3447

Ivo Hedtke
Schenker AG, Global Data & AI, Kruppstr. 4, 45128 Essen, Germany
E-mail: ivo.hedtke@dbschenker.com
ORCID: 0000-0003-0335-7825

Alexander Hoen
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: hoen@zib.de
ORCID: 0000-0003-1065-1651

Christopher Hojny
Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-mail: c.hojny@tue.nl
ORCID: 0000-0002-5324-8996

Rolf van der Hulst
University of Twente, Department of Discrete Mathematics and Mathematical Programming, P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: r.p.vanderhulst@utwente.nl
ORCID: 0000-0002-5941-3016

Dominik Kamp
University of Bayreuth, Chair of Economathematics, Universitaetsstr. 30, 95440 Bayreuth, Germany

E-mail: dominik.kamp@uni-bayreuth.de
ORCID: 0009-0005-5577-9992

Thorsten Koch
Technische Universität Berlin, Chair of Software and Algorithms for Discrete Optimization, Straße des 17. Juni 135, 10623 Berlin, Germany, and
Zuse Institute Berlin, Department A$^2$IM, Takustr. 7, 14195 Berlin, Germany
E-mail: koch@zib.de
ORCID: 0000-0002-1967-0077

Kevin Kofler
DAGOPT Optimization Technologies GmbH
E-mail: kofler@dagopt.com

Jurgen Lentz
RWTH Aachen University, Lehrstuhl für Operations Research, Kackertstr. 7, 52072 Aachen, Germany
E-mail: jurgen.lentz@rwth-aachen.de
ORCID: 0009-0000-0531-412X

Julian Manns
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: manns@zib.de

Gioni Mexi
Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: mexi@zib.de
ORCID: 0000-0003-0964-9802

Erik Mühmer
RWTH Aachen University, Lehrstuhl für Operations Research, Kackertstr. 7, 52072 Aachen, Germany
E-mail: erik.muehmer@rwth-aachen.de
ORCID: 0000-0003-1114-3800

Marc E. Pfetsch
Technische Universität Darmstadt, Fachbereich Mathematik, Dolivostr. 15, 64293 Darmstadt, Germany
E-mail: pfetsch@mathematik.tu-darmstadt.de
ORCID: 0000-0002-0947-7193

Franziska Schlösser
Fair Isaac Germany GmbH, Takustr. 7, 14195 Berlin, Germany
E-mail: franziskaschloesser@fico.com

Felipe Serrano
COPT GmbH, Berlin, Germany
E-mail: serrano@copt.de
ORCID: 0000-0002-7892-3951

Yuji Shinano
Zuse Institute Berlin, Department A$^2$IM, Takustr. 7, 14195 Berlin, Germany
E-mail: shinano@zib.de
ORCID: 0000-0002-2902-882X

Mark Turner
Zuse Institute Berlin, Department A$^2$IM, Takustr. 7, 14195 Berlin, Germany

E-mail: turner@zib.de
ORCID: 0000-0001-7270-1496

Stefan Vigerske
GAMS Software GmbH, c/o Zuse Institute Berlin, Department AIS$^2$T, Takustr. 7, 14195 Berlin, Germany
E-mail: svigerske@gams.com
ORCID: 0009-0001-2262-0601

Dieter Weninger
Friedrich-Alexander Universität Erlangen-Nürnberg, Department of Mathematics, Cauerstr. 11, 91058 Erlangen, Germany
E-mail: dieter.weninger@fau.de
ORCID: 0000-0002-1333-8591

Liding Xu
École polytechnique, LIX CNRR, Rue Honoré d'Estienne d'Orves. 1, 9120 Palaiseau, France
E-mail: lidingxu.ac@gmail.com
ORCID: 0000-0002-0286-1109