Zuse Institute Berlin

Takustrasse 7
D-14195 Berlin-Dahlem
Germany

AMBROS GLEIXNER, LEON EIFLER, TRISTAN GALLY, GERALD GAMRATH, PATRICK GEMANDER, ROBERT LION GOTTWALD, GREGOR HENDEL, CHRISTOPHER HOJNY, THORSTEN KOCH, MATTHIAS MILTENBERGER, BENJAMIN MÜLLER, MARC E. PFETSCH, CHRISTIAN PUCHERT, DANIEL REHFELDT, FRANZISKA SCHLÖSSER, FELIPE SERRANO, YUJI SHINANO, JAN MERLIN VIERNICKEL, STEFAN VIGERSKE, DIETER WENINGER, JONAS T. WITT, JAKOB WITZIG

The SCIP Optimization Suite 5.0

ZIB Report 17-61 (December 2017)

Zuse Institute Berlin
Takustrasse 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

The SCIP Optimization Suite 5.0

Ambros Gleixner[1], Leon Eifler[1], Tristan Gally[2],
Gerald Gamrath[1], Patrick Gemander[3], Robert Lion Gottwald[1],
Gregor Hendel[1], Christopher Hojny[2], Thorsten Koch[1],
Matthias Miltenberger[1], Benjamin Müller[1], Marc E. Pfetsch[2],
Christian Puchert[4], Daniel Rehfeldt[1], Franziska Schlösser[1],
Felipe Serrano[1], Yuji Shinano[1], Jan Merlin Viernickel[1], Stefan Vigerske[1],
Dieter Weninger[3], Jonas T. Witt[4], Jakob Witzig[1]

December 20, 2017

Abstract This article describes new features and enhanced algorithms made available in version 5.0 of the SCIP Optimization Suite. In its central component, the constraint integer programming solver SCIP, remarkable performance improvements have been achieved for solving mixed-integer linear and nonlinear programs. On mixed-integer linear programs, SCIP 5 is about 41% faster than SCIP 4 and over twice as fast on instances that take at least 100 seconds to solve. For mixed-integer nonlinear programs, SCIP 5 is about 17% faster overall and 23% faster on instances that take at least 100 seconds to solve. This boost is due to algorithmic advances in several parts of the solver such as cutting plane generation and management, a new adaptive coordination of large neighborhood search heuristics, symmetry handling, and strengthened McCormick relaxations for bilinear terms in MINLPs. Besides discussing the theoretical background and the implementational aspects of these developments, the report describes recent additions for the other software packages connected to SCIP, in particular for the linear programming solver SoPlex, the Steiner tree solver SCIP-Jack, the mixed-integer semidefinite programming solver SCIP-SDP, and the parallelization framework UG.

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

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

[1] *Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin, Germany,* {gleixner,eifler,gamrath,robert.gottwald,hendel,koch,miltenberger,benjamin.mueller,rehfeldt,schloesser,serrano,shinano,viernickel,vigerske,witzig}@zib.de

[2] *Technische Universität Darmstadt, Fachbereich Mathematik, Dolivostr. 15, 64293 Darmstadt, Germany,* {gally,hojny,pfetsch}@mathematik.tu-darmstadt.de

[3] *Friedrich-Alexander Universität Erlangen-Nürnberg, Department Mathematik, Cauerstr. 11, 91058 Erlangen, Germany,* {patrick.gemander,dieter.weninger}@fau.de

[4] *RWTH Aachen University, Chair of Operations Research, Kackertstr. 7, 52072 Aachen, Germany,* {puchert,witt}@or.rwth-aachen.de

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). 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).

1 Introduction

The SCIP Optimization Suite is a collection of software packages for modeling and solving a large variety of mathematical optimization problems. It consists of five individual tools, all of which are available in source code and can be downloaded free for usage in academic research:

- the modeling language Zimpl [59],
- the linear programming (LP) solver SoPlex [106], which provides an implementation of the revised simplex method with support for arithmetically exact optimization over the rational numbers [43],
- the constraint integer programming solver SCIP [3], which is built on a modular branch-cut-and-price framework fully equipped to function as a fast standalone solver for mixed-integer linear and nonlinear programs,
- the UG framework for parallelization of branch-and-bound solvers [94], and
- the generic branch-cut-and-price solver GCG [37].

This paper gives an overview of new features and improved algorithms provided by version 5.0 of the SCIP Optimization Suite.

Background The common release and distribution is motivated by the close interaction between the different components. A problem formulated with the modeling language Zimpl can directly be loaded into SCIP. By default, SCIP is linked to SoPlex for solving linear programs (LPs) as subproblems. GCG builds on the plugin-based design of SCIP in order to extend it by generic column generation routines on automatically detected problem decompositions. And finally, the UG framework can be linked to SCIP in order to create a parallel branch-and-bound solver on shared and distributed memory architectures.

While the scope of the SCIP Optimization Suite is certainly broader, a major focus lies on the solution of *mixed-integer linear programs* (MIPs) and more generally *mixed-integer nonlinear programs* (MINLPs). MIPs are optimization problems of the form

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

defined by $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, ℓ, $u \in \bar{\mathbb{R}}^n$, and $\mathcal{I} \subseteq \mathcal{N} := \{1, \ldots, n\}$ being the index set of integer variables. The usage of $\bar{\mathbb{R}} := \mathbb{R} \cup \{-\infty, \infty\}$ allows for variables that are unbounded or bounded only in one direction. MIPs are a special case of MINLPs, which can be written in the form

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

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_k : \mathbb{R}^n \to \mathbb{R}$, $k \in \mathcal{M} := \{1, \ldots, m\}$ are possibly nonconvex functions. Within SCIP, we assume that f and g_k are given explicitly in algebraic form, using only operators that are known to SCIP.

The core of SCIP implements a centrally coordinated branch-cut-and-price algorithm into which advanced techniques can be integrated via predefined callback mechanisms. Details on the organization of the solving process are described by Achterberg [2] and, specifically for the MINLP extensions, by Vigerske and Gleixner [100].

SCIP actually is a solver for an even larger class of optimization problems named *constraint integer programs* (CIPs). Inspired by the modeling flexibility known from constraint programming, the definition of constraint integer programming, however, is driven by algorithmic consideration. CIPs may be defined by arbitrary constraints as long as the following property holds: If all integer variables are fixed, the remaining problem must take the form of a standard linear or nonlinear program. In hindsight, this open, yet solver-oriented perspective, that was taken from the very beginning, has enabled SCIP to evolve into a highly flexible tool for diverse research interests. Only two examples that serve to demonstrate this point are PolySCIP, an extension for multicriteria MIPs by Schenker et al. [15], and SCIP-SDP, a solver for problems with positive-semidefiniteness constraints by Gally et al. [36].

Summary of New Developments Although less than a year has passed since the previous major release, SCIP 5 features a long list of algorithmic enhancements. The following is a selection of only the most important ones:

- a major revision of the generation and management of MIP cutting planes (Sections 2.4.1, 2.4.2, and 2.4.3),
- a new technique to analyze and exploit dual solutions from bound exceeding LP relaxations (Sections 2.5),
- a new coordination scheme for large neighborhood search heuristics (Sections 2.6.3),
- for the first time, automatic detection and breaking of symmetry, either by classical orbital fixing or by a newly developed polyhedral technique (Sections 2.3),
- strengthened McCormick relaxations for bilinear terms in MINLPs (Section 2.4.4),
- improved and enabled solution polishing when using SCIP with SoPlex (Section 3.1), and
- new interfaces to the NLP solvers FilterSQP and WORHP (Section 2.7.7).

The impact of these improvements is reflected in large gains regarding MIP and MINLP solver performance, detailed in Section 2.1.

In addition, in its latest version, the Steiner tree extension SCIP-Jack shows notable performance gains, both for the overall solver engine and specifically for the class of maximum-weight connected subgraph problems. The mixed-integer semidefinite programming (MISDP) solver SCIP-SDP, released in parallel with the SCIP Optimization Suite, now features a battery of warmstart techniques to reduce the solving time of the underlying SDP solvers.

The Zimpl version has not been updated in the current release and GCG 2.1.3 mainly provides smaller bugfixes and necessary adjustments to the new SCIP API. Last, not least, UG 0.8.5 now comes with a new parallization for SCIP-SDP and a small revision of the parallelization for the Steiner tree solver SCIP-Jack.

Structure of the Paper The paper is mostly organized according to the different components of the SCIP Optimization Suite. Section 2 describes new features and algorithmic advancements in SCIP itself, accompanied by computational results that analyze the performance progress of SCIP as MIP and MINLP solver. Section 3 details improvements in the LP solver SoPlex. Updates to the SCIP extensions SCIP-SDP and SCIP-Jack as well as the newly added SCIP application CycleClustering are presented in Section 4. Developments in the context of the UG framework are described in Section 5.

2 Advances in SCIP

2.1 Overall Performance Improvements for MIP and MINLP

One of the main modes of using SCIP is to employ it as a standalone solver for mixed-integer linear and nonlinear programs out-of-the-box. This section gives a summary of the overall improvements in solving performance that were achieved for this use case.

2.1.1 Computational Setup

Obtaining indicative performance results for MIP and MINLP solvers requires a careful benchmarking methodology. As first observed by Danna [23], small and from a mathematical point of view neutral changes to the problem formulation or the algorithm can have a large impact on the behavior and performance of a MIP or MINLP solver. Lodi and Tramontani [66] provide a recent overview of this phenomenon, which is called *performance variability* and poses a challenge to the accurate evaluation of deterministic, that is, systematic changes in the algorithm.

One important enhancement for robust performance experimentation was already introduced with the previous release SCIP 4: a centralized random seed that influences numerical perturbations in the LP solver as well as tie breaking and heuristic decision rules that necessarily occur in many parts of modern MINLP solvers [68]. Changing the value of this centralized random seed is one possibility to trigger performance variability systematically. The same holds for changing the permutation seed, which affects the order of variables and constraints in the formulation. In the following, we detail our standard computational setup used when comparing different solver versions.

Testsets A reasonably large and diverse testset is the basis for hedging against performance variability. As an example, the 87 instances of the MIPLIB 2010 benchmark selection often are too few in order to obtain robust results. Hence, the starting point for compiling the MIP testset were the publicly available instances of the COR@L testset [22] and the five MIPLIB versions up to MIPLIB 2010 [60], including all 361 instances of MIPLIB 2010. Instances that are marked as "numerically unstable" in the MIPLIB 2010 classification were removed and duplicate instances excluded, leaving in total 666 instances. In order to use available computing resources most efficiently and to increase the amount of experiments that can be performed during development, we decided to reduce this testset further by excluding instances that regularly exceed the time limit. Specifically, SCIP 4.0.0, SCIP 4.0.1, and intermediate development versions of SCIP 5 were run using five different random seeds. From these runs, all "solvable" instances for which at least one solver succeeded on at least one seed were selected. This yielded a final testset of 417 MIPs including 19 instances known to be infeasible.

For MINLP a similar procedure was followed, starting from the publicly available instances in MINLPLib2 [77]. Accounting for the fact that some instance types are overrepresented and some instances are known to be numerically troublesome by construction, 143 instances were selected from the base set, after applying manual inspection and heuristic filtering. Again, SCIP 4.0.0 and an advanced release candidate of SCIP 5 was run and all instances that could be solved within the time limit using at least one of five different random seeds were included. The resulting MINLP testset consists of 105 instances.

Note that, both for MIP and MINLP, the last step may have excluded instances that might be solved by the final release version of SCIP 5. However, all instances that can be solved with SCIP 4 are included. Thus, the performance results reported below provide a conservatively biased evaluation, that is, possibly, a slight underestimation of the improvements achieved by SCIP 5 for instances that can be solved to optimality.

Unless specified otherwise, these testsets are simply referred to as the MIP and the MINLP testset.

Hardware and Software The MIP experiments were performed on a cluster of machines with Intel Xeon E5-2670 v2 CPUs with 2.50 GHz and 128 GB main memory. For the MINLP experiments, a slightly faster cluster was used, which is equipped with Intel Xeon E5-2660 v3 CPUs at 2.60 GHz and 128 GB main memory. For all performance results, each instance was solved exclusively on one machine in order to ensure accurate time measurements. Only for the runs that were used to collect the MIP and MINLP testsets of "solvable" instances described above, non-exclusive runs were employed with a slightly increased time limit in order to account for a potential slowdown due to cache sharing.

SCIP interfaces to a set of external software packages. For the following performance results, SCIP 5.0.0 was built with GCC 5.4 and linked to

- the simultaneously released SoPlex 3.1.0 as underlying LP solver (see Section 3),
- the NLP solver Ipopt 3.12.5 [54] built with linear algebra package MUMPS 4.10 [7],
- the algorithmic differentiation code CppAD 20160000.1 [21], and
- the graph automorphism package bliss 0.73 [55] for detecting symmetry (with a patch, see Section 2.3.1).

SCIP 4.0.0 uses SoPlex 3.0.0, the same Ipopt and CppAD version, and no bliss.

Evaluation In the overall performance experiments, for each solver version each instance was run with five different random seed values in order to account for the effect of performance variability, including the default settings (seed of zero) of each version. For the MIP experiments we changed the random seed, for the MINLP experiments we used the permutation seed because this yielded a higher variability.

We compare the average performance of two solver versions following a similar methodology as described, for example, by Achterberg and Wunderling [4]. It is based on the shifted geometric mean of solving times and number of branch-and-bound nodes using a shift of 1 second and 100 nodes, respectively. The *shifted geometric mean* of values $t_1, \ldots, t_n$ is

$$\left(\prod (t_i + s)\right)^{1/n} - s$$

with shift s. Beforehand, we increase solving times below 0.5 seconds to 0.5 seconds in order to ensure that small inaccuracies in time measurement for these very fast instances don't affect the evaluation. Normalizing solving time and number of nodes with respect to SCIP 5.0 yields speedup and tree size reduction factors. In addition, the number of instances that could not be solved within the given time or memory limit is reported for each solver. (In the latter case the time limit is used when computing the mean solving times.)

As can be seen in Table 1 and 2, these statistics are displayed for different sets of instances. Both tables summarize results that treat every pair (instance,seed) as an individual observation. The subset "all" contains all instances of the "solvable" testsets, only excluding those for which one of the solvers encountered an error, including cases where the primal and dual bounds returned were inconsistent with a known optimal objective value. Additionally, we consider a hierarchy of filtered instance sets of increasing difficulty. The notation $[s, t]$ denotes the set of instances for which the maximum solving time between both solver versions lies between s and t seconds and at least one solver solved the instance. Hence, it excludes instances that cannot be solved by either version and those that can be solved in less than s seconds by both versions. Excluding the first set of instances is motivated by the fact that treating both solvers equally (with the time limit) may arbitrarily over- or underestimate their relative performance and only

Table 1: Performance comparison of SCIP 5 versus SCIP 4 on the MIP testset using five different seeds.

		SCIP 4.0.0+SoPlex 3.0.0			SCIP 5.0.0+SoPlex 3.1.0			relative	
Subset	instances	timeout	time	nodes	timeout	time	nodes	time	nodes
all	2069	403	131.6	4497	217	93.1	1699	1.41	2.65
[0,7200]	1878	212	88.2	3024	26	60.5	1087	1.46	2.78
[1,7200]	1734	212	123.9	3997	26	82.3	1343	1.50	2.98
[10,7200]	1413	212	282.5	7966	26	166.9	2299	1.69	3.47
[100,7200]	972	212	832.8	18715	26	402.4	4325	2.07	4.33
[1000,7200]	494	212	3197.5	71004	26	904.7	8546	3.53	8.31
diff-timeouts	238	212	5930.2	197521	26	854.6	10828	6.94	18.24
MIPLIBs	949	182	148.4	6282	120	116.6	2718	1.27	2.31
COR@L	1195	236	131.8	3910	117	87.0	1311	1.52	2.98

acts as a constant scalar on the relative performance comparison. Excluding the second set of instances selects harder instances in an unbiased way: if they are hard for any of the solvers. Achterberg and Wunderling [4] call these sets "brackets". Furthermore the tables show the degenerate time bracket "diff-timeouts", which contains all instances that can be solved by at least one of the solvers, but not both.

In addition, the MIP results are stated specifically for the instances contained in one of the MIPLIB versions and for the instances contained in the COR@L testset. (Note that some instances are contained in both testsets.) For MINLP, the table distinguishes also between MINLPs with integer variables ("integer") and pure NLPs ("continuous").

2.1.2 MIP Performance

For MIP experiments we use a time limit of 7200 seconds per instance and a zero gap limit. Table 1 provides a comparison of SCIP 4 and SCIP 5 with LP solver SoPlex 3.0 and 3.1, respectively, regarding their performance on the MIP testset described above. It can be seen that in all considered metrics—the number of solved instances, the mean solving time, and the mean number of branch-and-bound nodes explored—quite remarkable improvements have been achieved:

- Overall, SCIP 5 is about 41% faster than SCIP 4 and needs less nodes by a factor of 2.65.
- Most importantly, the number of instances that could not be solved within the time limit of two hours has been reduced almost by half from 403 to 217 instances.
- On the harder instances in the [100,7200] bracket, SCIP 5 is more than twice as fast as SCIP 4. The tree sizes are reduced by a factor of 4.33.
- The improved performance is more pronounced on instances of the COR@L testset, compared to the speedups on MIPLIBs.

Note that it is common to observe node reductions that exceed speedups, since often smaller search trees can only be achieved by an increased computational effort per node.

2.1.3 MINLP Performance

The MINLP experiments are performed with a time limit of 3600 seconds, because the number of instances solved between one and two hours currently is too small in order to justify the increased usage of computing time. In contrast to the MIP runs, we set a relative gap limit of 0.0001. Although we have not quantified this systematically, this choice is motivated by our repeated observation that spatial branch-and-bound solvers

Table 2: Performance comparison of SCIP 5 versus SCIP 4 on the MINLP testset using five different seeds.

		SCIP 4.0.0+SoPlex 3.0.0			SCIP 5.0.0+SoPlex 3.1.0			relative	
Subset	instances	timeout	time	nodes	timeout	time	nodes	time	nodes
all	515	125	177.5	39490	105	152.1	16010	1.17	2.47
[0,3600]	447	57	135.8	35708	37	118.2	15732	1.15	2.27
[1,3600]	442	57	143.0	37849	37	124.2	16634	1.15	2.28
[10,3600]	412	56	180.6	46063	36	153.8	19800	1.17	2.33
[100,3600]	291	55	380.8	99049	34	310.5	36694	1.23	2.70
[1000,3600]	117	48	1097.0	277998	32	631.2	69720	1.74	3.99
diff-timeouts	94	57	949.8	240517	37	257.3	26131	3.69	9.20
continuous	114	39	169.8	46495	38	138.4	23218	1.23	2.00
integer	401	86	179.8	37698	67	156.2	14403	1.15	2.62

suffer even more from performance variability, in particular, during this last solving phase when trying to close remaining gaps below this value.

As can be seen in Table 2, SCIP 5 outperforms SCIP 4 also on the MINLP testset:

- Overall, SCIP 5 is about 17% faster than SCIP 4, which can be observed almost equally on purely continuous and mixed-integer instances. The number of nodes decreased by a factor of 2.47.
- The number of instances that timed out decreased by 20 instances. This was mostly achieved on mixed-integer instances.
- On the harder instances in the [1000,3600] bracket SCIP 5 is even 74% faster and the tree sizes are reduced by a factor of 3.99.

Naturally, these MINLP results need to be taken with more caution since the benchmark sets available to us are significantly smaller and certainly less diverse and representative than for MIP. This is a difficulty that the MINLP solver community faces in general and will hopefully be improved over time. Nevertheless, although the precise quantification of the performance improvements should be read with care, the results give a strong indication that SCIP 5 has qualitatively improved in MINLP performance.

The remainder of Section 2 is dedicated to the description of new and improved features in SCIP 5 that contributed to the observed improvements. Where possible, their performance impact is quantified by similar experiments. However, note that some of these experiments were performed in slightly different setups. Some experiments could only be conducted with intermediate versions of the code. For some methods different testsets may be more indicative. And last, but not least, limited computing resources prohibited us from repeating each experiment with five seeds or permutations before finalizing the release and this report. Hence, not all of the results reported in the following may be reproducible in the most strict sense using the final release version of SCIP 5. Nevertheless, the results were included because they still help to give the reader an indication of how the individual improvements contributed to the whole.

2.2 Presolving

SCIP 5 implements three new presolving methods that are—in concert with the existing methods—applied before starting the branch-and-bound process in order improve the problem formulation: an analysis of the clique table for computing aggregations of variables, a sparsification method to reduce the number of nonzeros in linear constraints, and an improved disaggregation option for quadratic constraints.

2.2.1 Clique Table Analysis

The *clique table* in SCIP represents set packing conditions on binary variables. These may be given explicitly in the model or extracted during presolving. The clique table is another representation of the conflict graph [9] and used for cutting plane separation and propagation of linear constraints [9] as well as primal heuristics, see Section 2.6.1 and the recent technical report by Gamrath et al. [39]. The following description of the analysis of the clique table introduced in SCIP 5.0 uses the implication graph representation of the data stored in the clique table. The *implication graph* captures how the value of a binary variable implies values of other variables. Let B be the set of binary variables and $L = \{x_i : i \in B\} \cup \{\bar{x}_i : i \in B\}$ the set of literals, where $\bar{x}_i$ is the complemented variable $\bar{x}_i = 1 - x_i$, $i \in B$. In the following, literals are referenced by u and v, where $u = x_i$ or $u = \bar{x}_i$ for some $i \in B$, so u may be an original or a complemented variable. It follows that the complemented literal $\bar{u} = 1 - u$ gives $\bar{u} = \bar{x}_i$ if $u = x_i$ and $\bar{u} = x_i$ if $u = \bar{x}_i$. The same applies to v. The implication graph $G = (L, A)$ on the binary variables is a directed graph with a node for each literal and arcs $(u, v) \in A$ that encode the implication $u = 1 \Rightarrow v = 1$. Each clique C of binary literals can be represented in the implication graph by $|C|(|C| - 1)$ arcs $(u, \bar{v})$, $u \neq v \in C$.

SCIP 5.0 uses Tarjan's algorithm [98] to compute strongly connected components (SCCs) in the implication graph G. Recall that an SCC is a subgraph $G_S = (V_S, A_S)$ of a directed graph such that for each pair u, v of vertices in V_S, there exists a directed path in G_S from u to v. In the implication graph, this means that for each pair of variables u, v in an SCC, it holds $u = 1 \Rightarrow v = 1$ and $v = 1 \Rightarrow u = 1$. The latter implication is equivalent to $u = 0 \Rightarrow v = 0$, thus $u = v$. Based on this argument, all variables of an SCC can be aggregated to a single one. Moreover, during the SCC computation, a directed path from a node u to its complement $\bar{u}$ may be found. In this case, u can be fixed to 0.

While the computation of SCCs is independent of the order in which Tarjan's algorithm traverses the graph, the identification of possible fixings depends on it and thus on the start node(s) chosen for Tarjan's algorithm. In order to find more fixings, the SCC computation returns a topological order of the graph as a side product (note that after the aggregation of SCCs, the graph is cycle-free and such an order is guaranteed to exist). SCIP performs a second run of Tarjan's algorithm afterwards, starting from the last node in the topological order, to increase the chance to identify variable fixings.

The effort for Tarjan's algorithm is linear in the number of nodes and arcs of the graph. The explicit representation of each clique by a quadratic number of arcs, however, can turn this into a quadratic running time which is too expensive for large instances with many large cliques. A remedy for this situation is the observation by Achterberg [4] that a careful implementation only enters each clique at most twice. Therefore, SCIP works on the clique level and treats the arcs implicitly to make use of this observation and keep the effort for the clique table analysis reasonable. The clique table analysis is enabled by default in SCIP 5.0. At the time of addition, activating the feature resulted in a speedup of 1.8 % for the MIP testset and 5.7 % for the [100,7200] bracket.

2.2.2 Nonzero Cancellation

The basic idea of nonzero cancellation is to add an appropriately scaled linear equality to other constraints in order to improve the nonzero pattern of the constraint matrix $A \in \mathbb{R}^{\mathcal{M} \times \mathcal{N}}$. This approach to reduce the number of nonzeros in A has already been used by Chang and McCormick [20], Gondzio [44] and, more recently, Achterberg et. al. [5]. More precisely, assume two constraints

$$\begin{array}{rcrcrcrcl} A_{iU}x_U & + & A_{iV}x_V & + & A_{iW}x_W & & & = & b_i, \\ A_{rU}x_U & + & A_{rV}x_V & + & & & A_{rY}x_Y & \leq & b_r, \end{array}$$

with $i, r \in \mathcal{M}$ and disjoint subsets of the column indices U, V, W, $Y \subseteq \mathcal{N}$. Further, assume there exists a scalar $\lambda \in \mathbb{Q}$ such that $\lambda A_{iU} - A_{rU} = 0$ and $\lambda A_{ik} - A_{rk} \neq 0$ for all $k \in V$. Subtracting λ times the equality i from constraint r yields

$$\begin{array}{rcrcrcrcl} A_{iU}x_U & + & A_{iV}x_V & + & A_{iW}x_W & & & = & b_i \\ & & (A_{rV} - \lambda A_{iV})x_V & - & \lambda A_{iW}x_W & + & A_{rY}x_Y & \leq & b_r - \lambda b_i. \end{array} \tag{3}$$

The difference in the number of nonzeros of A is $|U| - |W|$. The case $|U| - |W| \leq 0$ does not seem to offer any advantage. Thus, the remainder of this section assumes $|U| - |W| > 0$, which means that the number of nonzeros actually decreases.

For mixed-integer programming, reducing the number of nonzeros of A has two main advantages. First, many subroutines in a MIP solver depend on this number. Especially the LP-solver benefits from sparse basis matrices. Secondly, nonzero cancellation may open up possibilities for other presolving techniques to perform useful reductions or improvements on the formulation. One special case occurs if $W = \emptyset$, that is, the column indices of the equality are a subset of the column indices of the other constraint. This case is of particular interest because decompositions may take place. The current default parameters restrict the nonzero cancellation to this case.

An exhaustive search for suitable pairs $(i, r) \in \mathcal{M} \times \mathcal{M}$ for A is usually too inefficient, since it is of quadratic complexity in the number of constraints. To achieve a favorable trade-off between search time and effectiveness, a hashing mechanism on variable pairs is used. For each pair of variable indices j, k in each equality i, the quadruple (a_{ij}, j, a_{ik}, k), consisting of the variable indices and their coefficients, is hashed using an open surjective hash function H. The quintuple $(i, a_{ij}, j, a_{ik}, k)$ is then saved using the key $H(a_{ij}, j, a_{ik}, k)$. If $H(a_{ij}, j, a_{ik}, k)$ is already contained in the hashtable, then the entry corresponding to the sparser row is kept. Afterwards, for each pair of variable indices j, k in each inequality r, the hashtable is queried for $H(a_{rj}, j, a_{rk}, k)$. If the conditions $a_{ij} = a_{rj}, a_{ik} = a_{jk}$, $W = \emptyset$ hold for the corresponding entry $(i, a_{ij}, j, a_{ik}, k)$, then reformulation (3) is applied. To further decrease search time, certain limits are used to heuristically prevent unrewarding investigations.

In addition, it seems important to preserve special structures by not applying cancellation if it would destroy one of the following properties in the row r above: integrality of the coefficients; more specifically coefficients $+1$ and -1; setpacking, setcovering, setpartitioning, or logicor constraint types; variables with no or only one lock. Finally, it should be mentioned that adding scaled equations to other constraints needs to be done with care. In particular, too large or too small scaling factors λ can lead to numerical problems. Currently, as in [5], a limit of $|\lambda| \leq 1000$ is used.

The performance impact of this feature on the complete MIP testset was neutral at the time of merging this feature, but two more instances where solved within the timelimit. In fact, only few instances are affected significantly. On the subsets [1000,7200] and MIPLIB, however, a performance improvement between 3% and 6% was observed.

2.2.3 Disaggregation of Quadratic Constraints

In a mixed-integer nonlinear program, quadratic constraints may appear in a block-separable form, that is, the quadratic function can be split into a sum of quadratic functions, such that no two functions share any variable. Disaggregating such constraints can improve performance, since the linear relaxation for each of the disaggregated constraints has to take less variables into account.

Formally, consider a quadratic constraint of the form

$$g(x) = \bar{a}^\top x + \sum_{i \in I} a_i x_i + \sum_{i,j \in I} q_{ij} x_i x_j \leq \beta, \tag{4}$$

with $\bar{a} \in \mathbb{R}^n$, $I \subseteq \mathcal{N}$, $a \in \mathbb{R}^I$, $(q_{ij}) \in \mathbb{R}^{I \times I}$, $\beta \in \mathbb{R}$, and $\bar{a}_i = 0$ for $i \in I$. Further, let $\{I_p\}_{p \in P}$ be a partitioning of I ($\cup_{p \in P} I_p = I$, $I_p \cap I_{p'} = \emptyset$ for $p, p' \in P$, $p \neq p'$) such that $q_{ij} = 0$ for all $i \in I_p$, $j \in I_{p'}$, $p, p' \in P$, $p \neq p'$.

Then the disaggregated formulation

$$\bar{a}^\top x + \sum_{p \in P} z_p \leq \beta, \tag{5}$$

$$\sum_{i \in I_p} a_i x_i + \sum_{i,j \in I_p} q_{ij} x_i x_j \leq z_p, \qquad p \in P, \tag{6}$$

is equivalent to (4). This reformulation can be advantageous when constructing a tight linear relaxation of the feasible set. For example, assume that m_p linear inequalities are used to relax each inequality in (6). To formulate the same relaxation without the additional variables z_p, $p \in P$, that is, based on the original formulation (4), all possible aggregations of (5) with the inequalities that relax equations (6) might be necessary. This may require $\prod_{p \in P} m_p$ inequalities in the worst case. Hijazi, Bonami, and Ouorou [50] provide an example where a simple outer approximation algorithm can profit considerably from disaggregation of quadratic constraints.

Since SCIP version 2.1.0, the parameter `constraints/quadratic/disaggregate` has allowed to enable the reformulation (5)–(6), using the finest possible partitioning of I. With this release, we have revised the disaggregation algorithm. The size of the partition ($|P|$) can now be bounded by the parameter `constraints/quadratic/maxdisaggrsize` (which replaces `constraints/quadratic/disaggregate`) and is set to 127 by default. Further, the constraints (5) and (6) are now scaled up sufficiently to ensure that a solution that is feasible for the reformulated constraints is also feasible for the original constraint.

It would also be possible to consider a disaggregation of (4) in which a variable can appear in more than one of the inequalities (6), thereby dropping the condition $q_{ij} = 0$ for variables i and j from different partition elements. For example, a constraint $x^2 + 2xy + y^2 \leq 1$ could be disaggregated into $z_1 + z_2 \leq 1$, $x^2 + 2xy \leq z_1$, and $y^2 \leq z_2$. However, while the original constraint is convex, this does not hold anymore for $x^2 + 2xy \leq z_1$. Additionally, stronger propagation might be applied to the original constraint, since interaction of variables can better be taken into account, see also Vigerske and Gleixner [100, Section 2.2.1.2].

2.3 Symmetry Handling

Let $\mathcal{S}_n$ be the symmetric group, i.e., the set of all permutations, on $\{1, \ldots, n\}$. For a given a mixed-integer program (MIP)

$$\min \{c^\top x \,:\, Ax \leq b,\ \ell_i \leq x_i \leq u_i \ \forall\, i \in \mathcal{N},\ x_i \in \mathbb{Z} \ \forall\, i \in \mathcal{I}\}, \tag{7}$$

a *symmetry* is a permutation $\gamma \in \mathcal{S}_n$, acting on $x \in \mathbb{R}^n$ via $\gamma(x) := (x_{\gamma^{-1}(1)}, \ldots, x_{\gamma^{-1}(n)})$, that maps feasible solutions onto feasible solutions preserving the objective value. If such symmetries are present in a MIP, this typically has a negative effect on the performance of branch-and-bound procedures, because symmetric solutions are inspected repeatedly during the solution process. For this reason, several symmetry handling techniques were discussed in the literature, see, for example, [34, 51, 58, 57, 62, 71, 72] and the overview by Margot [73].

To be able to handle symmetries in SCIP, a symmetry detection mechanism (Section 2.3.1) as well as two symmetry handling approaches have been implemented for the release of version 5.0. The first approach is based on separating and propagating valid inequalities for certain symmetry breaking polytopes (Section 2.3.2), whereas the second approach is a pure propagation approach to handle symmetries (Section 2.3.3). It is worth to point out that both approaches can exclusively handle symmetries of binary variables. Nevertheless, these approaches can also be used in general MIPs by ignoring symmetries of non-binary variables. Thus, symmetries of non-binary variables are not handled.

2.3.1 Symmetry Detection

To be able to handle symmetries, one first has to detect which symmetries are present in a MIP. However, computing all symmetries of a MIP is NP-hard, see [73]. Therefore, one typically refrains from computing symmetries of the feasible region of (7). Instead, one computes symmetries that keep a specific MIP formulation $Ax \leq b$ and its objective vector c invariant. These permutations form the so-called *formulation group* $\Gamma = \Gamma(A, b, c)$ of a MIP, which can be computed by determining the automorphism group of a suitably defined graph, see [90]. Consequently, Γ can be found by solving a graph automorphism problem, which is not known to be either NP-hard or solvable in polynomial time. However, it can typically be solved efficiently in practice by software like BLISS [55], NAUTY [76], or SAUCY [25]. With SCIP 5.0, users have the opportunity to link SCIP against BLISS to compute symmetries.

2.3.2 Symmetry Breaking Polytopes

Let Γ be the symmetry group of a binary program. A polyhedral approach to handle symmetries of Γ is given by adding for each $\gamma \in \Gamma$ the so-called *fundamental domain (FD) inequality*

$$\bar{c}^\top x \geq \bar{c}^\top \gamma(x), \quad \text{where } \bar{c} \coloneqq (2^{n-1}, 2^{n-2}, \ldots, 2, 1)^\top,$$

since these inequalities force a solution to be lexicographically maximal in its Γ-orbit, see [34]. However, this approach is impractical due to the large coefficients in $\bar{c}$, which may cause numerical instabilities in applications. To avoid this problem, a different approach considers the convex hull of the binary points fulfilling all these inequalities. This leads to so-called *symmetry breaking polytopes (symretopes)*

$$\mathrm{S}(\Gamma) = \operatorname{conv}\left(\left\{x \in \{0,1\}^n \,:\, \bar{c}^\top x \geq \bar{c}^\top \gamma(x),\ \gamma \in \Gamma\right\}\right),$$

which were introduced in [51].

The resulting polyhedral approach either uses a complete linear description of $\mathrm{S}(\Gamma)$ or derives an IP formulation with small coefficients to avoid the exponential coefficients of FD-inequalities. By adding valid inequalities for either formulation, one can handle symmetries in a binary program, because these inequalities enforce lexicographical maximality of a solution. However, complete linear descriptions of symretopes are, in general, not available. For this reason, the focus of the next section is on IP formulations of symretopes. Afterwards, a special symretope, the so-called full orbitope, is discussed in more detail.

IP Formulations with $\{0, \pm 1\}$-coefficients To obtain IP formulations of symretopes $\mathrm{S}(\Gamma)$ with left hand side coefficients in $\{0, \pm 1\}$, so-called *ternary IP formulations*, the approach of [51] is to combine IP formulations for different single FD-inequalities. This

leads to the notion of *symresacks* corresponding to a permutation $\gamma \in \Gamma$:

$$\mathrm{P}_\gamma \coloneqq \mathrm{conv}(\{x \in \{0,1\}^n \,:\, \bar{c}^\top x \geq \bar{c}^\top \gamma(x)\}).$$

FD-inequalities may contain positive and negative coefficients. Thus, flipping variables $x_i \mapsto 1 - x_i$ with a negative coefficient turns P_γ into a classical knapsack polytope $\tilde{\mathrm{P}}_\gamma$. This allows for deriving a ternary IP formulation for $\tilde{\mathrm{P}}_\gamma$ based on minimal cover inequalities.

Given a knapsack polytope $P = \mathrm{conv}(\{x \in \{0,1\}^n \,:\, a^\top x \leq \beta\})$ for some $a \in \mathbb{R}^n_+$, $\beta > 0$, a *cover* is a set $C \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in C} a_i > \beta$. A cover C is called *minimal* if every proper subset of C is not a cover. A classical result in the literature is that *minimal cover inequalities* $\sum_{i \in C} x_i \leq |C| - 1$ for all minimal covers C of P define an IP formulation of P, i.e.,

$$P \cap \{0,1\}^n = \Big\{ x \in \{0,1\}^n \,:\, \sum_{i \in C} x_i \leq |C| - 1 \text{ for all minimal covers } C \Big\}, \tag{8}$$

see Balas and Jeroslow [10]. Hence, a ternary IP formulation $\tilde{\mathrm{P}}_\gamma$ is given by (8). In particular, by reverting the transformation $x_i \mapsto 1 - x_i$, the IP formulation of $\tilde{\mathrm{P}}_\gamma$ turns into a ternary IP formulation of P_γ. In general, this IP formulation consists of exponentially many inequalities. For the particular case of symresacks, the formulation can be separated in $O(n\,\alpha(n))$ time, where α is the inverse Ackermann function, see [51]. Consequently, a ternary IP formulation for $\mathrm{S}(\Gamma)$ is given by combining the ternary IP formulations of P_γ for every $\gamma \in \Gamma$, which can be separated in $O(|\Gamma|\, n\, \alpha(n))$ time.

Of course, the above IP formulation for $\mathrm{S}(\Gamma)$ can be strengthened by replacing minimal cover inequalities for P_γ with tighter cutting planes, for example, facet defining inequalities of symresacks. However, such facets are in general unknown. But for the particular case in which the underlying permutation γ is a composition of m disjoint 2-cycles, a facet description of P_γ is available, see Kaibel and Loos [56]. The facets of the corresponding symresacks, so-called *orbisacks* O_m, can be separated in linear time, see Loos [67], and the separation routine of minimal cover inequalities can implemented to run in linear time as well, see [51].

To be able to handle arbitrary symmetries via minimal cover inequalities in SCIP, version 5.0 contains a constraint handler for symresacks that implements the separation routine of minimal cover inequalities. Moreover, the constraint handler contains a propagation routine for the FD-inequality associated with the symresack P_γ that runs in linear time and which was described in [51]. Furthermore, a constraint handler for orbisacks was included in SCIP. This constraint handler consists of the separation routine for both minimal cover and facet inequalities as well as a propagation routine for the corresponding FD-inequality.

Full Orbitopes The *full orbitope* $\mathrm{O}_{m,n}$ is the convex hull of all binary $(m \times n)$-matrices whose columns are sorted lexicographically non-increasing, see [57]. Thus, it is the symretope for the group Γ that acts on the order of the columns of such matrices. To be able to handle symmetries related to full orbitopes, the existing orbitope constraint handler has been extended by separation and propagation routines for full orbitopes, which will be discussed in turn.

Since a complete linear description of full orbitopes is unknown, the separation routine of the orbitope constraint handler separates an IP formulation of $\mathrm{O}_{m,n}$. However, the separation routine does not separate the IP formulation described in the last section, because this formulation would consist of the intersection of $n!$ symresacks. Instead, it uses the observation that a binary matrix X is contained in $\mathrm{O}_{m,n}$ if and only if the j-th column $X_{\cdot j}$ of X is not lexicographically smaller than the $(j+1)$-st column $X_{\cdot j+1}$. Furthermore, note that $X_{\cdot j}$ is not lexicographically smaller than $X_{\cdot j+1}$

if and only if $(X_{\cdot j}, X_{\cdot j+1})$ is contained in the symresack of the permutation γ_j which swaps X_{ij} and $X_{i,j+1}$ for every $i \in [m]$. Since γ_j is a composition of disjoint 2-cycles, P_{γ_j} is linearly equivalent to the orbisack O_m. As a consequence, the separation routine implemented in the orbitope constraint handler separates the minimal cover inequalities of the $n-1$ orbisacks P_{γ_j}. The running time of this routine is in $O(mn)$.

Let L and $U \in \{0,1\}^{m \times n}$ encode local lower and upper bounds of entries of X, respectively. Then the propagation algorithm for full orbitopes and bounds L and U is based on the observation that a binary matrix $X \in \mathrm{O}_{m,n}$ has the following structure, see [67]. The first row of X consists of $j \in \{0, 1, \ldots, n\}$ 1-entries which are followed by $(n-j)$ 0-entries. Moreover, the same structure holds recursively for all submatrices of rows 2 to m and column index sets $\{1, \ldots, j\}$ and $\{j+1, \ldots, n\}$. Thus, if the first $i-1$ rows were already propagated, the above observation can be used to set

- $L_{ij'} = 1$ for all $j' < j$ with $L_{ij} = 1$ for some $j \in \{1, \ldots, n\}$,
- $U_{ij'} = 0$ for all $j' > j$ with $U_{ij} = 0$ for some $j \in \{1, \ldots, n\}$, and
- declaring the bound pair (L, U) as infeasible if either of the fixings lead to a contradiction $U_{ij'} < L_{ij'}$.

Afterwards, the same routine can be called recursively for row $i+1$ within column ranges

$$\{1, \ldots, \max\{j : L_{ij} = 1\}\} \quad \text{and} \quad \{\min\{j : U_{ij} = 0\}, \ldots, n\}.$$

The running time of this procedure is in $O(mn)$.

Besides these routines for full orbitopes, the orbitope constraint handler is also able to handle orbitopes with additional structure, so-called packing and partitioning orbitopes. These orbitopes are defined as the convex hull of all vertices of $\mathrm{O}_{m,n}$ whose rows contain at most one 1-entry. In contrast to full orbitopes, a complete linear description of packing and partitioning orbitopes is known, and it can be separated and propagated in linear time, see [57, 58]. Both the separation and propagation routine are available in the orbitope constraint handler.

2.3.3 Orbital Fixing

For a binary program that is solved by branch-and-bound, denote for a given node of the branch-and-bound tree the sets of variables that were branched to 0 and 1 by B_0 and B_1, respectively. Moreover, let F_0 and F_1 be the set of variables that were fixed to 0 and 1, respectively, by some reason different from a branching decision. The idea of *orbital fixing* is to find further variables that can be fixed to 0 or 1 due to symmetries of a group Γ. To be able to formulate orbital fixing, the (set-wise) *stabilizer* of a set S, that is, all permutations that keep S invariant, is denoted by $\mathrm{Stab}_\Gamma(S) \coloneqq \{\gamma \in \Gamma : \gamma(S) = S\}$. The *orbit* $\Gamma(i) \coloneqq \{\gamma(i) : \gamma \in \Gamma\}$ of a variable i contains all variables that can be the image of i with respect to permutations in Γ.

Lemma 2.1 (Orbital Fixing [72, 81, 82])**.** *Let F_0 and F_1 be obtained by symmetry independent methods, and let $O = \mathrm{Stab}_\Gamma(B_1)(i)$ for some variable i.*

- *If $O \cap (B_0 \cup F_0) \neq \emptyset$, all variables in $O \setminus (B_0 \cup F_0)$ can be fixed to 0.*
- *If $O \cap F_1 \neq \emptyset$, all variables in $O \setminus F_1$ can be fixed to 1.*

Thus, to make orbital fixing work, in each node of the branch-and-bound tree the stabilizer group $\mathrm{Stab}_\Gamma(B_1)$ has to be determined; alternatively, one may compute the local symmetry group at the corresponding node. However, both approaches are relatively costly. For this reason, the implementation of orbital fixing in SCIP uses the following heuristic to find a subgroup Σ of $\mathrm{Stab}_\Gamma(B_1)$: Let $G = \{\gamma_1, \ldots, \gamma_m\}$ be a set of generators of Γ. In each node of the branch-and-bound tree, the heuristic *filters* out

generators $\gamma \in \Gamma$ that do not keep B_1 invariant. The remaining generators generate the subgroup Σ, which is used as an approximation of $\mathrm{Stab}_\Gamma(B_1)$ within orbital fixing. This approach is valid since $\Sigma(i) \subseteq \mathrm{Stab}_\Gamma(B_1)(i)$ for every variable i.

2.3.4 Using Symmetry Handling in SCIP

Based on the implementation of symmetry handling routines described in [83], both orbital fixing and the polyhedral approach to handle symmetries were implemented in SCIP 5.0 and can be activated by setting the parameter `misc/usesymmetry` to 1 (symretopes) or 2 (orbital fixing, default). To deactivate symmetry handling in SCIP, the parameter has to be set to 0. Moreover, the user has the opportunity to decide whether symmetries are computed before or after the presolving methods of SCIP by setting the parameter `presolving/symmetry/computepresolved` to `FALSE` or `TRUE`, respectively. By default, symmetries are computed after presolving.

If either of the symmetry handling approaches is activated, the symmetry presolver of SCIP is called exactly once to compute symmetries (before or after the remaining presolving steps). Furthermore, if the symretope approach is used, the symbreak presolver is called exactly once at the very end of all presolving steps to determine whether a part of the symmetry group Γ can be handled by full/packing/partitioning orbitopes, and adds the corresponding constraints. For the generators of Γ that do not contribute to symmetries handable by orbitopes, the presolver adds either symresack or orbisack constraints.

After presolving has finished, the actual symmetry handling methods are called at the nodes of the branch-and-bound tree. Orbital fixing uses the symmetry group computed by the symmetry presolver to fix variables. The symretope approach separates inequalities for the different constraint handlers that were added by the symbreak presolver. Moreover, propagation methods of these constraint handlers are called. Note, however, that if either of the symmetry handling methods is activated, all other components of SCIP that might break symmetry have to be deactivated. Otherwise, SCIP may declare a feasible but non-optimal solution as optimal. In particular, if user plugins are added to SCIP that might affect the symmetry group of the problem, the symmetry handling routines of SCIP should be deactivated.

Computational Results To evaluate the impact of symmetry handling on the performance of SCIP, we first conducted experiments on the MIPLIB 2010 benchmark testset and 16 highly symmetric instances taken from Margot [72].[1] In comparison to SCIP without symmetry handling methods, orbital fixing achieves a speedup of about 13.0% and solves 2 more instances (77 instead of 75) on the MIPLIB 2010 benchmark set. The speedup of the polyhedral approach is about 9.5% with the same number of solved instances. On Margot's highly symmetric instances, however, the picture is different: Orbital fixing leads to a speed-up of 72.9% and solves 5 more instances (11 instead of 6). The polyhedral approach achieves a speedup of 91.6%, solving 8 more instances.

In order to investigate the impact on the full MIP testset described in Section 2.1, default SCIP with orbital fixing was compared against SCIP without symmetry handling.[2] This shows that turning off symmetry handling leads to an increase of running time by about 12% on the instances that can be solved by one of the solvers. On harder instances of the [1000,7200] bracket, deactivating symmetry handling even leads to a 57% slowdown of SCIP. Furthermore, if also those instances are taken into account that are

[1] These experiments were run on a Linux cluster with Intel i3 3.2GHz dual core processors and 8GB memory using a time limit of one hour and the default seed zero. Note that using a smaller time limit dampens the influence of unsolved instances and hence slightly favors the weaker solver.

[2] This was performed with a time limit of two hours and the hardware and computational setup described in Section 2.1, using the default seed zero.

solved by neither of these settings, deactivating symmetry handling leads to an overall slowdown of 10% on MIPLIB instances and 15% on COR@L instances.

2.4 Separation and Convexification

SCIP 5 includes numerous improvements for several aspects of the cutting plane separation. This includes algorithmic improvements and bugfixes for the separators for Gomory, StrongCG, CMIR, flowcover, and $\{0, \frac{1}{2}\}$-cuts. In addition, the management of cutting planes, fundamental for a good performance of branch-and-cut codes, has also been revised and improved. For MINLPs, SCIP 5 provides a novel technique to strengthen the convexification of bilinear terms.

In a nutshell, the separation of cutting planes in SCIP works as follows. It is implemented in terms of separation rounds. Within a single separation round the goal is to find cuts that are violated by the current LP solution. All the cuts found in one separation round are added to the *separation storage*. When the separation round is finished, the separation storage selects a subset of its cuts for entering the LP relaxation, because it is usually not a good idea to select all cuts, see, e.g., [1, 8, 3, 101]. Still, the cuts that are not added to the LP should not be discarded immediately, as they might turn out to be violated in future LP solutions. Therefore, globally valid cuts—cuts that are valid in all nodes of the branch-and-bound tree—are kept in a different cut storage, the *cut pool*. The cut pool adds violated cuts again to the separation storage and filters cuts, for instance if they are duplicates of other cuts.

This strategy involves several decisions about individual cuts, therefore, different properties of the cuts are taken into consideration. The *efficacy* of a separating cut $a^\top x \leq b$ is the Euclidean distance between the current LP solution x^* and the halfspace defined by that cut, that is, $\frac{a^\top x^* - \beta}{\|a\|}$. The *parallelism* between two vectors u and v is defined as $\frac{\langle u,v \rangle}{\|u\| \cdot \|v\|}$ and is used to measure the similarity between two cuts, or a cut and the objective function.

2.4.1 Cut Management

The cut management includes the filtering and selection of cuts, when to add or remove cuts from the LP, and how often cutting planes are separated. This sections describes the differences between SCIP 4 and SCIP 5 in this matter.

Cut Filtering and Cut Selection The separation storage aims to select a subset of cuts that are of high quality and dissimilar. To achieve this, the selected subset of cuts must satisfy a maximum parallelism constraint. The selection is performed greedily, the best cut by some quality measure is selected first and all inferior cuts that violate the maximum parallelism constraint are removed from the separation storage. The process is iterated until no cuts are left, or the maximum number of cuts has been selected. SCIP 4 and SCIP 5 differ in their quality measure and parallelism constraint.

SCIP 4 computes the score as a weighted sum of the efficacy, the parallelism to the objective function, and the parallelism between cuts that have already been selected in the current separation round. The latter property scores a cut higher if it has a low parallelism to already selected cuts, but can cause the scores to change after each selected cut. The maximum parallelism in SCIP 4 is 0.5 by default.

Following suggestions of Wesselmann and Suhl [101], the score of a cut in SCIP 5 is computed as a convex combination of its efficacy, objective parallelism, and *integral support*—the percentage of its support that is on integer variables. Moreover, the parallelism to other cuts is not included in the score anymore, instead the cuts are filtered with a stricter default value of 0.1 for the maximum parallelism. For cuts of relatively

high quality, however, there is an exception: If the score of a cut is at least 0.9 times the highest score of all cuts found in that separation round, the maximum allowed parallelism is still 0.5.

Besides, the cut pool in SCIP 5 applies some filtering steps. It detects parallel cuts with a hashing approach and only adds cuts to the separation storage if their efficacy is above a certain threshold η. The value of η is initialized relative to the highest efficacy seen so far. If in several rounds the cut pool contained violated cuts, but none of them, or too many, had an efficacy above η, then η is adjusted as suggested by Andreello et. al. [8].

Adding and Removing Cuts from the LP SCIP 4 separates general MIP cuts only in the root node. Most separators add globally valid cuts to both, the cut pool and the separation storage. The cut pool is also only separated in the root node by default and therefore solely takes care of adding cuts found in earlier separation rounds of the root node again. SCIP 5 uses the cut pool and the separation storage differently. The separators for general MIP cuts now add all the globally valid cuts exclusively to the cut pool. Global cuts that enter the LP relaxation at local nodes are not removed from the LP anymore when switching to a different node. By default, SCIP 5 separates the cut pool at every node whose depth is a multiple of 10 and also calls the separators at local nodes. For separation at nodes that are not dual-bound-defining, the separators are requested to only add globally valid cuts.

2.4.2 General Improvements of Separators

The separation algorithms of the most important MIP cuts have many similarities. Usually LP rows are aggregated according to some rule to obtain a base inequality. Then a relaxation is created from the base inequality for which different kinds of cuts can be separated. In SCIP 4 these steps are implemented as a single routine which leads to inefficiencies, because it is not possible to reuse the same base inequality and try to generate different kinds of cutting planes.

Therefore, SCIP 5 adds an API for computing an aggregation of LP rows, which better exploits sparsity and can be used independently of the cut generation routines. The routines for aggregation and cut generation both use double-double arithmetic internally to achieve roughly twice the precision and mostly avoid numerical rounding errors, see Section 2.7.3. Moreover, a post-processing step is applied after the cut generation. It entails the enforcement of a maximal dynamism by cancelling small coefficients with bound constraints, appropriate scaling, and coefficient strengthening [91], a technique known from presolving. The improved accuracy and numerics due to these changes allow to remove several checks for rejecting cuts because of numerical considerations, such as limiting the rank of cuts or only accepting cuts that can be scaled to integral coefficients with small scaling factors.

Additionally, SCIP 5 includes a new implementation of the separator for $\{0, \frac{1}{2}\}$-cuts [18], which is now enabled by default. At the time of incorporating the new implementation, it gave a modest improvement of the solving time by 3% to 5%. On a few instances, however, it is a key component: SCIP 5 solves the instance `macrophage` from the MIPLIB 2010 benchmark set in under 200 seconds consistently over several random seeds, whereas SCIP 4 does not solve it within 2 hours.

2.4.3 Complemented Mixed-Integer Rounding (CMIR) and Flowcover Separation

The class of complemented mixed-integer rounding (CMIR) cuts [70] is one of the most important classes of cutting planes for solving MIP problems [4]. In the separation

Table 3: Performance comparison of SCIP 4 and SCIP 5 on UFCN [80] and ULSB [61] instances.

		SCIP 5.0.0+SoPlex 3.1.0			SCIP 4.0.1+SoPlex 3.0.1			relative	
Testset	instances	timeout	time	nodes	timeout	time	nodes	time	nodes
UFCN	83	8	42.0	657	22	134.7	13444	3.21	20.45
ULSB	65	0	5.1	6	9	1155.0	1074292	227.36	195.33

procedure, a base inequality is generated by aggregating rows heuristically with the aim of projecting out *active* continuous variables—continuous variables that are strictly between their bounds in the current LP solution. Subsequently, a bound substitution step is performed to obtain a mixed knapsack relaxation from the base inequality, in other words, variables are complemented or shifted with their bounds so that they have a lower bound of zero. Finally, different scaling factors are tested and an MIR cut is generated for the scaling factor that yields the best cut.

For SCIP 5, the aggregation heuristic and the cut generation heuristic have been improved. The scoring of rows is now based on a measure that indicates how much the activity of a row is influenced by the fractionality of the current LP solution x^*. Particularly, let $a^\top x + c^\top y \leq b$ be a row with x integer and y continuous. Then the integral variables contribute $a^\top \min(x^* - \lfloor x^* \rfloor, \lceil x^* \rceil - x^*)$ to the score. Furthermore, all y_i, which appear in a constraint of the form $y_i \leq d_i x_i + b$ with x_i integer such that $y_i^* = d_i x_i^* + b$, contribute $c_i\, d_i\, \min(x_i^* - \lfloor x_i^* \rfloor, \lceil x_i^* \rceil - x_i^*)$ to the score. After a cut was generated from an aggregation of rows, the scores for these rows are decreased based on their parallelism to the generated cut. Additionally, the aggregation heuristic prefers equalities that only contain a single active continuous variable, since such an equality can always be used to project this variable out of other rows without introducing new active continuous variables.

The cut generation heuristic for CMIR cuts now also separates lifted flow cover inequalities [47] for the same base inequality. SCIP 4, on the contrary, uses a CMIR-based approach for separating flowcover inequalities [69, 104] only from single rows. Moreover, the cut generation heuristic runs considerably faster, since the base inequality is only computed once and the testing of different scaling factors improved algorithmically. In particular, by exploiting the observation that the efficacies of MIR cuts obtained from one base inequality with n integer variables and m continuous variables for k different scaling factors can be computed in $O(k \cdot n + m)$ instead of $O(k \cdot (n + m))$ by aggregating the continuous variables. Eventually, only the cut with the highest efficacy is considered to be added to the LP relaxation, namely either the lifted flowcover cut or the MIR cut obtained with the best scaling factor.

The improvements of the CMIR and flowcover separation can have a tremendous impact on problem instances that contain fixed charge network structure, such as the MIP formulations for many network design, lot-sizing, and unit commitment problems: Table 3 lists computational results for instances of the uncapacitated lot-sizing problem with backlogging (ULSB) from Küçükyavuz et. al. [61] and of the uncapacitated fixed charge network flow problem (UFCN) from Ortega et. al. [80]. SCIP 5 achieves a speed-up of 3.2 on the UFCN and 227.3 on the ULSB instances. The latter ones are mostly solved in the root node with SCIP 5 and always faster than SCIP 4. Also note that on UFCN every instance solved by SCIP 4 is solved by SCIP 5 as well, and SCIP 4 is only faster on a few easier instances that are solved quickly by branching and therefore the increased time spent in separation does not pay off.

2.4.4 Stronger Relaxations for Bilinear Terms Using Linear Inequalities

The well-known McCormick relaxation [75, 6] describes the convex hull of the graph of $x_i\,x_j$ on a box $[\ell_i, u_i] \times [\ell_j, u_j]$ and is of major importance for creating linear relaxations for nonconvex constraints involving bilinear terms. Even though the McCormick relaxation is best possible for $[\ell_i, u_i] \times [\ell_j, u_j]$, it does not yield the convex hull when considering additional, for example, linear constraints on x_i and x_j. The tighter convex hull of the graph of $x_i\,x_j$ has been described by Linderoth [64] for triangular domains, by Hijazi [49] on sets defined by a single constraints $x_i \leq x_j$, and by Locatelli [65] on general polytopes $P \subset \mathbb{R}^2$.

However, in order to apply these results, it is necessary to identify valid inequalities containing only the variables of bilinear terms automatically in the solver. These inequalities could be already present in the MINLP (2), but could also be derived from exploiting a linear relaxation $\mathcal{R}$. A simple example is optimization-based bound tightening (OBBT), which yields the best possible variable bounds, i.e., singleton inequalities, over $\mathcal{R}$. (See Gleixner et al. [42] for details on the OBBT implementation in SCIP.)

In order to find other non-axis parallel inequalities, SCIP 5.0 solves for each bilinear term $x_i\,x_j$ that appears in at least one nonconvex quadratic constraint auxiliary LPs of the form

$$\max \left\{ \lambda : \begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} s_i \\ s_j \end{pmatrix} + \lambda \begin{pmatrix} t_i - s_i \\ t_j - s_j \end{pmatrix},\ x \in \mathbb{R},\ \lambda \in [0,1] \right\}. \tag{9}$$

Here $s_i,\ t_i \in \{\ell_i, u_i\}$ and $s_j,\ t_j \in \{\ell_j, u_j\}$ are chosen such that $s_i \neq s_j$ and $t_i \neq t_j$ holds. Let x^* be the optimal solution of the LP (9). If $(x_i^*, x_j^*) \neq (t_i, t_j)$ then there exists an inequality that separates the point t from $\mathcal{R}$. Using LP duality, one can show that aggregating the constraints of the linear relaxation weighted by the optimal dual multipliers results in a valid inequality of the form

$$\alpha_i\,x_i + \alpha_j\,x_j \leq \alpha_0 \tag{10}$$

with $\alpha_i, \alpha_j \neq 0$. These up to four inequalities per bilinear term form a polytope $P_{i,j} \subseteq [\ell_i, u_i] \times [\ell_j, u_j]$, that constitutes a relaxation of the projection of $\mathcal{R}$ onto x_i and x_j. It can be used in order to separate the convex hull of the graph of $x_i\,x_j$ according to the formulas given by Locatelli [65].

In SCIP 5.0, all LPs (9) are solved in the already existing `prop_obbt` propagator. Note that the LPs depend on the current bounds of the variables appearing bilinearly and thus are solved after the classical OBBT-LPs. SCIP uses for each call of the propagator the same iteration limit as for OBBT, but bounds the total number of LP iterations for solving (9) via the parameter `propagating/obbt/itlimitfactorbilin`. Each inequality of the type (10) is passed to `cons_quadratic` through a new interface function `SCIPaddBilinearIneqQuadratic`. The inequalities are only used during the term-wise separation of nonconvex quadratic constraints whenever a bilinear term needs to over- or underestimated.

The new feature is enabled per default and operates on bilinear terms that appear in at least one nonconvex quadratic constraint. Overall, 398 instances of the MINLPLib2 are potentially affected by this technique in the sense that at least one nontrivial inequality for at least one bilinear term could be derived in an experiment without working limits on the solution ofthe auxiliary LPs. On this set of instances, the root gap could be reduced by 40% when setting the separation emphasis to aggressive. A thorough evaluation of the overall performance impact is still to be conducted. When activating the feature, the number of instances solved over five permutations on the MINLP testset increased by two.

2.5 Conflict and Dual Proof Analysis

During branch-and-bound, SCIP analyzes subproblems that are infeasible due to contradicting bound changes, infeasible LP relaxations, or LP relaxations that exceed the objective cutoff bound provided by the incumbent solution. In SCIP 4.0, *dual ray analysis* was introduced to extend the analysis of infeasible LP relaxations, see [103]. Vaguely speaking, dual ray analysis amounts to converting the Farkas proof that is valid under local bounds into a globally valid inequality that is used for propagation during the subsequent tree search. Prior to SCIP 4.0, only *conflict graph analysis* was available, which is a generalization of SAT techniques [74] to MIP developed by Achterberg [3]. For SCIP 4.0, the addition of dual ray analysis yielded a speedup of 1.05 on the MIPLIB 2010 benchmark set and 1.19 on hard instances, i.e., those instances that take more than 1000 seconds solving time [68].

In SCIP 5.0 the conflict graph analysis for *bound exceeding LPs* is now complemented by the so-called *dual solution analysis*. Consider a subproblem of form

$$\min \{c^\top x \ : \ Ax \leq b,\, \ell' \leq x \leq u',\, x_i \in \mathbb{Z}\ \forall i \in \mathcal{I}\} \tag{11}$$

with local bound vectors $\ell \leq \ell' \leq u' \leq u$. The subproblem can be pruned if the LP relaxation value of (11) exceeds the current objective cutoff bound $z^\star$. This holds if and only if there exists a feasible solution $(y', \underline{r}', \overline{r}')$ of the LP dual

$$\max \{\underline{r}^\top \ell' - \overline{r}^\top u' - b^\top y \ : \ \underline{r} - \overline{r} - A^\top y = c,\ y,\, \underline{r},\, \overline{r} \geq 0\} \tag{12}$$

such that $(\underline{r}')^\top \ell' - (\overline{r}')^\top u' - b^\top y' > z^\star$. From this dual solution the inequality

$$(c + A^\top y')^\top x \leq b^\top y' + z^\star \tag{13}$$

can be constructed, which is globally valid, but violated for all $x \in [\ell', u']$. SCIP 5.0 collects and uses inequalities of type (13) for propagation in order to avoid exploring suboptimal subproblems. By default, both conflict graph and dual solution analysis are enabled for bound exceeding LPs. Modifying the parameter `conflict/useboundlp`, the analysis for this kind of infeasibility can be disabled completely or changed to use only one of conflict graph or dual solution analysis.

Note that dual ray analysis may be considered a special case of dual solution analysis when setting $c = 0$ and $z^\star = 0$ in (13). Indeed, while both cases are distinguished, SCIP internally uses the same implementation for filtering, storing, propagating, and dynamically removing useless inequalities.

In SCIP 4.0 and prior versions, the analysis of bound exceeding LPs was disabled by default. In SCIP 5.0, extended by dual solution analysis, its activation yielded a speedup of 1.02 on the MIP testset and 1.06 on the [1000,7200] bracket. On MIPLIB 2010, dual solution analysis improved the solving time by a factor of 1.07 and reduced the tree size by a factor of 1.12.

2.6 Primal Heuristics

The current SCIP release contains two new primal heuristics as well as improvements to two existing pre-root heuristics. The first new addition is a dedicated method to construct integer-feasible solutions to mixed-binary MINLPs, which is based on solving a sequence of mathematical programs with equilibrium constraints and goes back to Schewe and Schmidt [92]. The second new heuristic implements a novel scheme to coordinate many of the existing large neighborhood search heuristics in a more dynamic manner.

2.6.1 Improved Structure-driven Fix-and-Propagate Heuristics

After the addition of the variable locks heuristic in the last release [68], SCIP 5.0 features a rework of the other two structure-based pre-root heuristics *clique* and *variable bound* [38].

Both heuristics employ a fix-and-propagate scheme. In a first step, they fix variables based on structures in the problem and apply domain propagation after each fixing to identify implied bound changes. In SCIP 5.0, infeasible fixings in this phase are now undone by a backtracking step and the fixing phase is continued. By this, higher final fixing rates are reached. Consequently, the solution of the reduced LP solved after the fixing phase can be rounded to an integer feasible solution more often. This allows to skip the more expensive sub-MIP solve in many cases and even if the sub-MIP is solved, the problem is smaller and therefore faster to solve.

The clique heuristic uses a different fixing scheme now. Instead of computing a clique partition at the beginning, it selects a clique with the highest number of unfixed variables in each step. It then selects the cheapest among these variables and fixes it to 1; all others are automatically fixed to 0 by domain propagation. The variable bound heuristic now takes into account cliques when sorting the variable bound graph topologically, since they represent a special form of variable bounds. Additionally, it was extended by two more fixing scheme variants, which allow applying fixings only if they correspond to the better or worse bound of the variable with respect to the objective function, respectively. More details and extensive computational experiments for all three structure-driven heuristics are provided in the technical report [39].

The updated clique and variable bounds heuristic are both enabled by default in SCIP 5.0. Although the average solving time stayed unchanged, other measures showed improvements in computational experiments: four more instances of out MIP test set were solved within the time limit of two hours, the primal integral was reduced by 3 % and the time to the first solution by 11 %. For instances with more inherent structure, as is the case for the instances in the MIP testset that originate from the COR@L testset, larger improvements can be observed: solving time and primal integral are reduced by 4 % and 10 %, respectively.

2.6.2 The MPEC Heuristic

A mathematical program with equilibrium constraints (MPEC) is an optimization problem with complementarity constraints or variational inequalities. For example, the constraint $x \in \{0, 1\}$ can be modeled as a complementarity constraint $x \perp 1 - x$, i.e., $x\,(1 - x) = 0$, and $x,\, 1 - x \geq 0$.

The basic idea of the MPEC heuristic, developed by Schewe and Schmidt [92], is as follows. Given a mixed-binary nonlinear problem (MBNLP), reformulate it as an MPEC and solve the MPEC to local optimality. The MPEC reformulation can itself be reformulated to an NLP by writing $x \perp 1-x$ as $x\,(1-x) = 0$. However, solving this NLP reformulation with a generic NLP solver will often fail. One issue is that the reformulated complementarity constraints will not, in general, satisfy constraint qualifications such as the Linear Independence or Mangasarian-Fromovitz constraint qualifications.

Therefore, in order to increase the chances of solving the NLP reformulation of the MPEC successfully, the heuristic solves regularized versions of the NLP by relaxing $x(1 - x) = 0$ to $x(1 - x) \leq \theta$, for different, ever smaller $\theta > 0$. Specifically, the MPEC heuristic, as implemented in SCIP 5.0, proceeds as described in Algorithm 1. By default, the starting values are $\theta = \frac{1}{8}$ and $\sigma = \frac{1}{2}$. These values can be modified with the parameters `heuristics/mpec/inittheta` and `heuristics/mpec/sigma`, respectively.

Activating the MPEC heuristics did not have a major effect on the overall performance of SCIP on the MINLP testset. By default, the heuristic is currently only applied

Algorithm 1: MPEC heuristic as implemented in SCIP 5.0

Input: MBNLP, $\theta \in (0, \frac{1}{4})$, $\sigma \in (0, 1)$, current LP solution x_{LP}
Output: feasible solution x^* or "no solution found"

Build regularized NLP (rNLP);
$\hat{x} \leftarrow x_{LP}$, re-initialized $\leftarrow$ false, fixed $\leftarrow$ false;
while *true* **do**
  solve (rNLP) using $\hat{x}$ as initial point;
  if *(rNLP) is feasible* **then**
    let x be the feasible solution of (rNLP);
    if $x_i \approx 0 \vee x_i \approx 1$ *for all* $i \in \mathcal{I}$ **then**
      call sub-NLP heuristic with x as initial solution;
      if *sub-NLP heuristic succeeded* **then**
        return solution found by sub-NLP heuristic;
    else
      $\theta \leftarrow \theta \cdot \sigma$, $\hat{x} \leftarrow x$, re-initialized $\leftarrow$ false, fixed $\leftarrow$ false
  else
    if *(rNLP) is infeasible* **then**
      let x be the returned infeasible point of (rNLP);
      if $x_i \approx 0 \vee x_i \approx 1$ *for all* $i \in \mathcal{I}$ **then**
        return no solution found
      if *not re-initialized* **then**
        for all $i \in \mathcal{I}$ set $\hat{x}_i$ to 1 if $\hat{x}_i < 0.5$, otherwise to 0;
        re-initialized $\leftarrow$ true;
        continue;
      if *not fixed* **then**
        for all $i \in \mathcal{I}$ fix $\hat{x}_i$ to 1 if $\hat{x}_i < 0.5$, otherwise to 0;
        fixed $\leftarrow$ true;
        continue;
      return solution found by sub-NLP heuristic;
  update (rNLP) with new θ;

with conservative working limits. However, out of 42 calls on the whole testset it found improving solutions in 6 cases, two of which were an optimal solution.

2.6.3 Adaptive Large Neighborhood Search

Large Neighborhood Search (LNS) heuristics for MIP and MINLP explore an auxiliary problem around a (set of) reference points or solutions. For MIP, auxiliary problems are usually defined by fixing a subset of the integer variables directly to some reference solution values such as in RINS [24], or by adding additional inequalities as in Local Branching [29]. Sometimes the original objective function is replaced by an artificial objective function. Adaptive Large Neighborhood Search (ALNS) is a novel framework to coordinate eight LNS heuristics: Crossover and Mutation [87], DINS [41], Local Branching, Proximity [30], RENS [14], RINS, and Zero Objective.

Inside ALNS, all these LNS heuristics are subject to a single computational budget, which makes it easier to restrict or emphasize the use of LNS techniques in SCIP altogether. In addition, the selection of the next LNS technique to run is centralized. The selection is inspired by algorithms for multi armed bandit problems (see [16] and

the reference therein). By default, ALNS selects the next LNS heuristic based on upper confidence bounds [16] around an LNS specific reward function, which weighs the success of a run against its computational cost.

The difficulty of the auxiliary problem is used by the ALNS framework to adapt the minimum improvement, the node budget, and the target fixing rate between runs. If the auxiliary problem is solved to optimality or proven to be infeasible, the target fixing rate is reduced. If the solution process terminated without a solution, the target fixing rate is increased because the auxiliary problem was too hard. Adjusting the target fixing rate has been originally proposed by [87].

On the algorithmic side, ALNS combines the variable fixings of an LNS heuristic with a generic variable fixing priorization to (un)fix additional integer variables if the heuristic misses (exceeds) its individual target fixing rate. Fixings are prioritized to keep the auxiliary problem as connected as possible by considering proximity in the variable constraint graph. In case of a tie, reduced costs and pseudo costs of the fixings are compared to prioritize between two variables. It is noteworthy that this generic variable priorization allows to run LNS heuristics such as Local Branching that originally do not fix variables on their own. Such heuristics have been previously deactivated in SCIP because of their expensive runtime behavior. Inside ALNS, these heuristics are started conservatively, requiring about 45 % of the variables fixed. Only if they are successful, the target fixing rate is gradually reduced to 0 %.

ALNS is now activated as one of the default primal heuristics of SCIP. By default, it uses all LNS heuristics whose auxiliary problem definitions modify the source MIP at hand. The first 8 runs call all neighborhoods in a randomized order once. After that, the selection process is determined by upper confidence bounds based on the observed rewards.

An overall speedup on the MIP testset of 2 % has been obtained, in particular a 4 % speedup on MIPLIB 2010. The benefit of ALNS increases with the instance difficulty. On harder instances that take 1000 seconds or more, the speedup is even 8 %. The performance on MIPLIB 2010 is better if only the already active primal heuristics RENS, RINS, and Crossover were used for ALNS. However, the performance on other instances improved more by the help of additional, previously deactivated neighborhoods. In the future, the selection of algorithmic solver strategies via bandit selection may extend to other components of SCIP. As a first step, all three bandit routines have therefore been made available through the public API, see also Section 2.7.1.

2.7 Technical Improvements and New User Features

In this section, a list of smaller algorithmic enhancements, new data structures, and user features are described that come with the current release of SCIP:

- central data structures for bandit algorithms as used, for example, by the ALNS heuristics described above (Section 2.7.1),
- a new "union find" data structure for maintaining connected components information (Section 2.7.2),
- a fast software-side implementation for quadruple precision (Section 2.7.3),
- a new plugin type that allows users to add tables to SCIP's statistics output (Section 2.7.4),
- the possibility to query maximum violations of constraints, variable bounds, and integrality requirements (Section 2.7.5),
- the functionality to display a classification of linear constraints (Section 2.7.6), and last, not least,
- two new interfaces to the NLP solvers FilterSQP and WORHP (Section 2.7.7).

2.7.1 Bandit Algorithms

The introduction of the ALNS heuristic (Section 2.6.3) required an algorithmic component that selects among several LNS heuristics under the uncertainty which of them will perform best. Such a selection problem is referred to as *multi armed bandit problem* [16]. In its basic version, the multi armed bandit problem is often presented as a game, in which the player has to select between a finite set of actions and receives a reward for the selected action only. The player's goal is to maximize their total reward. Clearly, the challenge lies in a careful balance between exploration over the entire set of available actions, and the greedy exploitation of the action that performed well on average.

The SCIP API has been extended by three selection algorithms for multi armed bandit problems, namely upper confidence bounds, ϵ-greedy, and Exp.3, see [16] for details. Except for their creation, which requires a set of one or two individual parameters to control their selection behavior, the algorithms use the same interface and can be exchanged easily. In every round, a selection is made by the bandit algorithm using the method `SCIPbanditSelect`. Second, the reward is returned via `SCIPbanditUpdate`. In the future, the bandit routines might also be applied in other parts of SCIP that repeatedly require a coordinated choice among a class of similar algorithms.

2.7.2 Disjoint Set Data Structure

The core data structures of SCIP have been extended by a disjoint set data structure for a fast update of graph connectedness information. Disjoint sets (also known as "union find") maintain a representative of the connected component of every node of a graph. Whenever an edge is inserted, only the representatives of the two end nodes have to be merged.

Connectedness information is used by several default plugins and core technologies of SCIP. The connected components of the clique table are now maintained with a disjoint set data structure, which provides memory and performance benefits compared to the previous implementation using a directed graph. Most notably, it allows a fast update of connectedness information when a new clique enters the clique table.

2.7.3 Double-double Arithmetic

SCIP uses double-precision (64bit) floating-point arithmetics and therefore can solve problem instances only up to certain numerical tolerances. Moreover, the potential accumulation of round-off errors can compromise the correctness of results regarding feasibility and optimality. Therefore it is important to perform critical computations defensively and to discard results if there are indications that the desired accuracy cannot be reached. Another solution in such cases would be to switch to a higher level of precision, for example by using arbitrary precision libraries such as GMP [46]. These libraries, however, are usually orders of magnitudes slower than computations in double precision, which would affect performance.

Hence, starting with version 5, SCIP provides routines that use so-called *double-double arithmetic* due to Rump [89], which achieves roughly twice the precision, that is, quadruple precision. These routines can be implemented in terms of standard double-precision operations where each number is represented as an unevaluated sum of two double-precision floating-point values. The operations are implemented without branches and use only ordinary arithmetic instructions. As a result, they are well optimized by modern compilers and very fast in practice. The exact sum of two double-precision numbers requires only 6 additions. In SCIP 5 double-double arithmetic is used during the separation of MIP cutting planes.

2.7.4 Statistics Tables

For the current release a further plugin type has been added to SCIP: the statistics table. This plugin type controls the output of statistics within SCIP. Whenever the statistics output is requested from SCIP through the console or some other filestream, the `TABLEOUTPUT` callback of all active statistics tables will be called in order of increasing position parameters. With the addition of the statistics table plugin type, also all default SCIP statistics have been changed to statistics tables, so that they can now be enabled or disabled directly through their corresponding `active` parameter. Additional sections can also be easily added to the SCIP statistics by users through writing their own `SCIP_TABLE`. For details on the implementation of new statistics tables, see the corresponding how-to article in the online documentation.

2.7.5 Numerical Violations

Every constraint handler of SCIP must implement a callback to check the feasibility of a solution, which reports either "feasible" or "infeasible" back to SCIP. Even if a solution is reported feasible, slight violations below the tolerances may be present. With version 5.0, SCIP allows to query this information for every solution. Numerical violations of LP rows, constraints, variable bounds and integrality are now computed when a solution is checked. The maximum absolute and relative violations of the incumbent solution are displayed upon calling `checksol` in the interactive shell. Even for feasible solutions, displaying the violations may give an insight into possible numerical problems.

The implementation of the extended check is optional for constraint handlers. The displayed violations should therefore be considered a lower bound to the actual maximum violations. This bound is tight if the solution was checked completely and all involved constraint handlers implement the update. All default constraint handlers support the computation of numerical violations. Every constraint handler should use the following methods if it detects a violation.

- `SCIPsolUpdateIntegralityViolation`,
- `SCIPsolUpdateBoundViolation`,
- `SCIPsolUpdateLPRowViolation`, and
- `SCIPsolUpdateConsViolation`.

As a convenience method, `SCIPsolUpdateLPConsViolation` can be used to update the numerical violation of a constraint that is represented as an LP row.

2.7.6 Classification of Linear Constraint Types

Most MIP models feature linear constraints with a certain structure. With version 5.0, SCIP can classify the linear part of a given MIP or MINLP. The classification recognizes the different linear constraint types of the MIPLIB 2010 benchmark library, which it extends by constraints of type `PRECEDENCE`. The resulting 17 types are given in Table 4.

After reading a problem, the linear classification is invoked from the SCIP interactive shell with the command `display linclass`. This method applies the above classification hierarchy (from top to bottom) to every linear constraint of the problem. In particular, neither nonlinear constraints are considered, nor linear constraints that have been upgraded to more special types during presolving. In order to obtain a complete constraint classification after presolving, all linear upgrades should be disabled.

It is important to emphasize that there is a difference between the definitions of cardinality constraints in the classification above and the corresponding constraint handler of SCIP. The latter considers inequalities instead of equations and is not restricted to binary variables.

Table 4: Classification of linear constraint types

Type	Description
`EMPTY`	linear constraint with no variables
`FREE`	linear constraint with no finite side
`SINGLETON`	linear constraint with a single variable
`AGGREGATION`	equation with two variables
`PRECEDENCE`	inequality where both variables have the same type and coefficient with opposite sign ($ax - ay \leq b$)
`VARBOUND`	general inequality for two variables including a binary variable ($ax + by \leq c$, $x \in \{0,1\}$)
`SETPARTITION`	set partition constraint ($\sum_i x_i = 1$, $x_i \in \{0,1\}\ \forall i$)
`SETPACKING`	set packing constraint ($\sum_i x_i \leq 1$, $x_i \in \{0,1\}\ \forall i$)
`SETCOVERING`	set covering constraint ($\sum_i x_i \geq 1$, $x_i \in \{0,1\}\ \forall i$)
`CARDINALITY`	generalized set partition constraint ($\sum_i x_i = k$, $x_i \in \{0,1\}\ \forall i$, $k \geq 2$)
`INVKNAPSACK`	generalized set packing constraint ($\sum_i x_i \leq b$, $x_i \in \{0,1\}\ \forall i$, $b \in \mathbb{N}_{\geq 2}$)
`EQKNAPSACK`	knapsack equation ($\sum_i a_i x_i = b$, $x_i \in \{0,1\}, a_i \in \mathbb{N}\ \forall i$, $b \in \mathbb{N}_{\geq 2}$)
`BINPACKING`	special knapsack constraint of the form ($\sum_i a_i x_i + ax \leq a$ with $x, x_i \in \{0,1\}\ \forall i$, $a_i, a \in \mathbb{N}_{\geq 2}$)
`KNAPSACK`	general 0/1 knapsack inequality ($\sum_i a_i x_i \leq b$, $x_i \in \{0,1\}\ \forall i$, $b \in \mathbb{N}_{\geq 2}$)
`INTKNAPSACK`	general integer knapsack ($\sum_i a_i x_i \leq b$, $x_i \in \mathbb{Z}\ \forall i$, $b \in \mathbb{N}$)
`MIXEDBINARY`	inequality or equation for binary and continuous variables ($\sum_i a_i x_i + \sum_i p_i s_i \{\leq, =\} b$, $x_i \in \{0,1\}\ \forall i$)
`GENERAL`	general linear constraint that matches none of the above types

In rare cases, a single linear constraint can be classified as two different types. An example of this is a combination of set covering and and invariant knapsack inequalities in a single ranged row. In such a case, the number of original constraints may be smaller than the number of classified constraints.

2.7.7 New Interfaces to NLP Solvers FilterSQP and WORHP

New interfaces to the Nonlinear Programming solvers FilterSQP and WORHP have been added to SCIP. FilterSQP is an implementation of the Sequential Quadratic Programming (SQP) method by Fletcher and Leyffer [31, 32]. It is not publicly available, but may be obtained from Sven Leyffer upon request. WORHP by Büskens and Wassel [17] implements the SQP method and a Penalty-Interior-Point (IP) algorithm. It is developed at the University of Bremen and is free for academic purposes [105].

SCIP can be compiled with multiple NLP solvers and selects the solver with the highest priority at the beginning of the solving process. The priorities of FilterSQP and WORHP's IP and SQP implementations can be adjusted via the three parameters `nlpi/{filtersqp,worhp-ip,worhp-sqp}/priority`. By default, the priorities of all available NLP solvers are, in descending order: Ipopt, WORHP-IP, FilterSQP, WORHP-SQP. If more than one solver is available, then it is possible to solve all NLPs during the solving process with all available NLP solvers by setting the parameter `nlpi/all/priority` to the highest value. In this case, SCIP uses the solution from the solver that provides the best objective value. Other possible use cases for the availability of multiple solvers have not been implemented yet.

An extensive performance comparison between Ipopt, WORHP, and FilterSQP in SCIP is available in Müller et al. [79]. The results show that Ipopt is more successful in finding local optimal points than the other NLP solvers, but SCIP is performing fastest with FilterSQP.

3 SoPlex

SoPlex 3.1 is a minor update on the previous version and mainly includes internal improvements on already existing features. Most notable are the improved solution polishing implementation and a new aggressive scaling method.

3.1 LP Solution Polishing

LP solution polishing exploits degeneracy in the problem in order to choose an alternative LP optimum, for instance one with less fractional variables. This technique was already implemented in SoPlex 3.0 [68]. In this version the implementation is improved by not simply looping over all nonbasic variables until no pivot step can be performed anymore. Instead, we keep a candidate list of slack or continuous variables that may be pivoted into the basis. This reduces the involved overhead while maintaining the original algorithmic behavior.

Most importantly, LP solution polishing has been activated inside SCIP 5.0 during probing and diving mode in order to decrease fractionality. This can positively affect the success of primal heuristics that iteratively solve LPs, possibly interleaved with rounding and propagation steps, in order to arrive at an integer feasible solution. Prominent examples are the feasibility pump heuristic and many diving heuristics. Polishing remains deactivated during strong branching and OBBT propagation, which are not focused at finding primal solutions.

At the time of activating this feature, an experiment on the MIP testset with random seed zero showed a SCIP speedup of 6%. Especially harder instances in the [100,7200] bracket were positively affected with a speedup of over 10%.

3.2 A New Aggressive Scaling Method

Matrix scaling is a widely used means to improve the conditioning of linear programs, see, e.g., [27, 84]. SoPlex 3.1 features a new method that combines two existing scaling variants, geometric and equilibrium scaling. While equilibrium scaling divides all coefficients in each nonzero row and column of the constraint matrix by the absolute largest entry within this vector, geometric scaling uses a simplified geometric mean of the absolute vector entries as divisor: For each column A_j of the constraint matrix A the divisor is $\sqrt{\max_{i:a_{ij}\neq 0} |a_{ij}| \cdot \min_{i:a_{ij}\neq 0} |a_{ij}|}$, for each row $a_{i\cdot}$ it is $\sqrt{\max_{j:a_{ij}\neq 0} |a_{ij}| \cdot \min_{j:a_{ij}\neq 0} |a_{ij}|}$.

Geometric mean scaling is computationally more expensive than equilibrium scaling, since it is applied iteratively (up to eight times in SoPlex); equilibrium scaling on the other hand always converges in one step. While geometric scaling attempts to reduce the maximum absolute coefficient ratio within each column and row, equilibrium scaling yields a matrix such that the largest entry in each nonzero row and column is of magnitude one. The new scaling method performs geometric followed by equilibrium scaling. Like least-squares scaling it is mostly recommended for numerically difficult instances and can be activated with the command line parameter `-g6`. More information about the scaling procedure implemented in SoPlex can be found in Maher et al. [68].

3.3 Technical Improvements

The stability and reliability of the solver has been improved by fixing several bugs and numerical issues. Furthermore, SoPlex 3.1 comes with two smaller modifications. First, the maximum number of updates to the LU factorization before can now be set via a

parameter from within SCIP (`lp/refactorinterval`). This allows to tune the solver more easily to problems where a larger or smaller refactorization interval is desirable.

Second, in the sparse data structures entries with value zero are now automatically removed when creating or modifying a vector. This gave a minimal performance improvement and complies with newly added checks in the LP solver interface of SCIP, which forbid the introduction of structural nonzero entries with value zero. This is motivated by the desire to make behavior of different LP solvers as consistent as possible.

4 Applications and Extensions

The SCIP Optimization Suite comes with a series of applications and extensions that have been developed to solve various classes of mathematical programming problems. The current release contains a new application CycleClustering for a graph partitioning problem motivated by the analysis of Markov processes [102, 26] and improvements to the Steiner tree solver SCIP-Jack [40] and the MISDP extension SCIP-SDP [36].

4.1 CycleClustering

The CycleClustering application is an implementation of a clustering problem that detects cyclic behavior in Markov processes [102]. Given a discrete set of states $\mathcal{S} = \{1, \ldots, n\}$ and a set of clusters $\mathcal{K} = \{1, \ldots, m\}$, a clustering is defined as a partitioning of the set of states into m pairwise disjoint clusters

$$\mathcal{S} = \bigcup_{t=1}^{m} C_t, \; C_t \cap C_{t'} = \emptyset \text{ for all } t \neq t'.$$

This can be modeled by introducing binary decision variables x_{it} for each state $i \in \mathcal{S}$ and each cluster $t \in \mathcal{K}$ with

$$x_{it} = 1 \iff i \in C_t \iff \text{state } i \text{ is assigned to cluster } t.$$

In cycle clustering, the goal is to identify an ordered set of clusters $(C_1, \ldots, C_n)$ with high *net flow* f_t between consecutive clusters and high *coherence* g_t within clusters. A scaling parameter $\alpha > 0$ is used to control the emphasis on coherence in the objective function. The cycle clustering problem can be formulated as the MINLP

$$\begin{aligned}
\max \; & \sum_{t \in \mathcal{K}} f_t \; + \; \alpha \cdot \sum_{t \in \mathcal{K}} g_t & & \\
\text{s.t.} \; & \sum_{t \in \mathcal{K}} x_{it} = 1 & & \text{for all } i \in \mathcal{S}, \\
& \sum_{i \in \mathcal{S}} x_{it} \geq 1 & & \text{for all } t \in \mathcal{K}, \\
& g_t = \sum_{\substack{i,j \in \mathcal{S} \\ i<j}} (q_{ij} + q_{ji}) x_{it} x_{jt} & & \text{for all } t \in \mathcal{K}, \\
& f_t = \sum_{\substack{i,j \in \mathcal{S}, \\ i \neq j}} (q_{ij} - q_{ji}) x_{it} x_{j\phi(t)} & & \text{for all } t \in \mathcal{K}, \\
& x_{it} \in \{0, 1\} & & \text{for all } t \in \mathcal{K}, i \in \mathcal{S}, \\
& f_t, g_t \geq 0 & & \text{for all } t \in \mathcal{K},
\end{aligned}$$

where $\phi(t) = t + 1$, if $t < m$ and $\phi(m) = 1$ otherwise.

The CycleClustering application solves the above MINLP by using a branch-and-cut approach on a problem-specific, compact linearization. Problem-specific heuristics, valid inequalities, and specific branching rules were implemented to improve the performance on these challenging MIP models. Further details can be found in [26].

4.2 SCIP-Jack: Steiner Tree and Related Problems

Given an undirected, connected graph $G = (V, E)$, costs (or weights) $c : E \to \mathbb{R}_+$ and a set $T \subseteq V$ of *terminals*, the Steiner tree problem in graphs (SPG) asks for a tree $S = (V_S, E_S) \subseteq G$ such that $T \subseteq V_S$ holds and $\sum_{e \in E_S} c(e)$ is minimized.

The SPG is a classical optimization problem, being the subject of hundreds of research articles, see [52] for an overview. It also has real-world applications, albeit rarely in pristine form. However, there are many applications that involve problems closely related to the SPG. To handle such problems, the SCIP Optimization Suite contains SCIP-Jack, an exact solver for the SPG and 11 of its variants.

The latest version SCIP-Jack 1.2 comes with a number of generic improvements, such as its own solution pool and cache-optimized graph routines. Compared to SCIP-Jack 1.1, which was released with the SCIP Optimization Suite 4.0, SCIP-Jack 1.2 delivers notable performance gains. Many instances can be solved more than twice as fast. A striking example of the improved performance is the degree-constrained Steiner tree problem: Although SCIP-Jack 1.2 does not include any new problem-specific features for this variant, it solves almost twice as many instances in two hours. With this it even outperforms other, more specialized solvers such as the one described by Liers et al. [63].

In addition to the significant speedup provided by generic implementation improvements, the main advances have been problem-specific, most notably for the maximum-weight connected subgraph problem (MWCSP). These improvements include new preprocessing techniques, new heuristics, and a new IP formulation, see [86] for a detailed description. SCIP-Jack 1.2 is for many MWCSP instances more than three orders of magnitude faster than other solvers and is able to solve formerly unsolved problems in less than a minute.

While the focus of the recent developments has been on variants of the SPG such as the maximum-weight connected subgraph problem, the next year will see a return to the roots: the classical SPG. The aim is to improve performance not only by enhancing existing implementations, but by implementing a number of new techniques—a few having been published for instance in Rehfeldt and Koch [85].

4.3 SCIP-SDP

SCIP-SDP is an external plugin for solving mixed-integer semidefinite programming (MISDP) in SCIP. It uses either a cutting plane approach, similarly to how SCIP solves MINLPs, or a nonlinear branch-and-bound approach using interfaces to interior-point SDP solvers. SCIP-SDP solves MISDPs in the (dual) form

$$\begin{aligned} \inf \quad & b^\top y \\ \text{s.t.} \quad & \sum_{i=1}^{m} A_i\, y_i - C \succeq 0, \\ & y \in \mathbb{R}^p \times \mathbb{Z}^{m-p}, \end{aligned} \tag{MISDP}$$

with symmetric matrices C, $A_i \in \mathbb{R}^{n \times n}$ for all $i = 1, \ldots, m$. For a general description of SCIP-SDP see [36].

For Version 3.1 the possibility to warmstart interior-point solvers within the nonlinear branch-and-bound approach was added. The main effort of the nonlinear branch-and-bound approach consists of solving the primal-dual pair

$$\begin{array}{lll} \inf & b^\top y & \\ \text{s.t.} & \sum_{i=1}^{m} A_i\, y_i - C \succeq 0, & \text{(SDP-D)} \\ & y \in \mathbb{R}^m & \end{array} \qquad \begin{array}{lll} \sup & C \bullet X & \\ \text{s.t.} & A_i \bullet X = b_i & \text{for all } i = 1, \ldots, m, \\ & X \succeq 0 & \\ & & \text{(SDP-P)} \end{array}$$

in each node of the branch-and-bound tree. For the simplex algorithm for linear problems many iterations can be saved by starting the solving process of each node after the root from the optimal basis of the parent node, since the optimal basis of the parent node is always dual feasible for the child and therefore a valid starting point for the dual simplex algorithm. For interior-point algorithms warmstarting is much harder. Feasible interior-point algorithms require the initial point to lie in the relative interior of the feasible region, preferably close to the central path, but the final iterates of the parent node will usually be close to the boundary, since a linear function is optimized over a convex set. Infeasible primal-dual interior-point-methods can compensate for the need of the solution to be feasible for the new variable bounds after branching, but they still require the solution to be sufficiently centered, which is not the case for optimal solutions.

Warmstarting techniques for interior-point algorithms can be grouped into three main categories. Either the interior-point methods themselves are adjusted, or earlier iterates of the solving process of the parent node are used, or the optimal solutions of the parent node are adjusted in an additional preprocessing step. While adjusted interior-point methods using penalty techniques have already been successfully implemented in solvers for mixed-integer nonlinear programs, for example by Benson and Shanno [12] in the MINLP solver MILANO [11] using the interior-point solver LOQO [99], they are not really usable for SCIP-SDP, which interfaces existing SDP solvers. Therefore these will not be discussed in the following and the reader is referred to the overview article by Engau [28] for an extension of those methods to mixed-integer semidefinite programming.

Starting from Earlier Iterates One possibility to find a solution in the relative interior, first proposed by Gondzio [45], is to use an earlier iterate of the interior-point algorithm, which may still have a larger duality gap, but is still strictly feasible. In this approach the interior-point solver is stopped once the duality gap has reached a predetermined value ε_1, at which point the primal and dual solutions for the current iterate are stored, before continuing until the desired final gap tolerance ε_2 has been reached.

Convex Combination with Interior-Point For restoring strict feasibility for the optimal solution, one possibility is to take a convex combination with a point in the relative interior. Since the optimal solution has to be primal and dual feasible, taking the convex combination with a primal and dual strictly feasible point will lead to a solution that is again primal and dual strictly feasible, but also close to the optimal solution. This kind of technique was already used by Helmberg and Rendl in 1998 [48], but has recently been further investigated by Skajaa et al. [97] in the context of the homogeneous self-dual embedding.

The strictly feasible solution can be chosen as a scaled identity matrix, similar to the usual default initial point of most interior-point solvers, with scaling factors depending on the largest entries of either both primal and dual matrix, or with different factors for the primal and dual matrix, respectively. A more involved choice for the solution in the

relative interior would be to compute the analytic center of the feasible set once in the root node and use this throughout the tree (the true analytic center might change, but since the computation may be as costly as solving the SDP-relaxation, it is not feasible to compute it anew in each node).

Projection onto Positive Definite Cone Instead of moving the solution of the parent node towards some more or less arbitrary point within the cone of positive definite matrices, it is also possible to project it, to find a positive definite matrix that is as close as possible to the original solution. Of course such a projection onto the cone of positive definite matrices cannot exist, since the cone is not closed. From a computational point of view, however, it is also not sufficient for the matrix to be positive definite, it needs to be sufficiently far away from the boundary. Therefore, SCIP-SDP uses the projection $P_{\underline{\lambda}}$ onto the set of all symmetric matrices with all eigenvalues larger than or equal to some $\underline{\lambda} > 0$. Given an eigenvector decomposition $V\text{Diag}(\lambda)V^\top = X$ with $V \in \mathbb{R}^{n\times n}$ (which exists because X is real, symmetric and positive semidefinite), this projection can be computed explicitly as

$$P_{\underline{\lambda}}(X) = V\text{Diag}((\max\{\lambda_i, \underline{\lambda}\})_{i\leq n})V^\top, \tag{14}$$

which is a generalization of the well-known projection onto the set of positive semidefinite matrices proposed by Schwertman and Allen [93].

Rounding Problems Çay et al. [19] proposed a warmstarting approach for mixed-integer second-order cone programming based on Jordan frames, which are an extension of eigenvector decompositions to general Jordan algebras. Applied to MISDPs, the general idea of the approach is to fix the eigenvector decomposition of the parent node and optimize over the corresponding eigenvalues, which becomes a linear program, to restore feasibility for the adjusted bounds and afterwards use a convex combination again to move the solution to the interior. In the first step, feasibility of the primal solution will be restored. For this, given an eigenvector decomposition $V\text{Diag}(\hat{\lambda})V^\top = \hat{X}$ of the optimal primal solution of the parent node, the following linear so-called "rounding problem" is solved:

$$\begin{aligned} \sup \quad & C \bullet (V\text{Diag}(\lambda)V^\top) \\ \text{s.t.} \quad & A_i \bullet (V\text{Diag}(\lambda)V^\top) = b_i \quad \text{for all} \quad i = 1, \dots, m, \\ & \lambda_i \geq 0 \quad \text{for all} \quad i \leq n. \end{aligned} \tag{P-R}$$

Since (P-R) is a restriction of (SDP-P) to matrices with the same eigenvectors as $\hat{X}$, the optimal objective value of (P-R) gives a lower bound on the optimal objective value of (SDP-P) and therefore, by weak duality, also on the optimal objective of (SDP-D). This implies that if (P-R) is unbounded (or more generally dual infeasible), so is (SDP-P) and therefore (SDP-D) has to be infeasible and can be cut off without having to solve an SDP. Furthermore, as mentioned in the conclusion of [19], by the same arguments it can be shown that if the optimal objective of (P-R) is larger than the cutoff bound of (MISDP), e.g., the best known integer solution, the node can also be cut off via bounding.

If the node could not be cut off via (P-R), as a second step the corresponding dual rounding-problem

$$\begin{aligned} \inf \quad & b^T y \\ \text{s.t.} \quad & \sum_{i\leq m} A_i y_i - W\text{Diag}(\mu)W^T = C, \\ & \mu_i \geq 0 \quad \text{for all} \quad i \leq n, \quad y \in \mathbb{R}^m, \end{aligned} \tag{D-R}$$

Algorithm 2: Warmstart via rounding problems

Input: (bounded) SDP-relaxation (SDP-D), parentnode solution $(\hat{X}, \hat{y}, \hat{Z})$, cutoff bound U, strictly feasible solution (X^0, y^0, Z^0), IP-weight γ

Output: optimal solution $(X^\dagger, y^\dagger, Z^\dagger)$ or "cutoff" if (SDP-D) is infeasible or the optimal objective is no better than the cutoff bound

```
compute eigenvector decomposition V Diag(λ̂) V^⊤ of X̂;
solve (P-R) for V;
if (P-R) is primal feasible then
    if (P-R) is dual infeasible then
        return cutoff;
    else
        let p be the optimal objective value and λ̄ a solution of (P-R);
        if p ≥ U then
            return cutoff;
        else
            compute eigenvector decomposition W Diag(μ̂) W^⊤ of Ẑ;
            solve (D-R) for W;
            if (D-R) is primal feasible then
                let d be the optimal objective value and (ȳ, μ̄) a solution of (D-R);
                if d = p then
                    return (X̄ := V Diag(λ̄) V^⊤, ȳ, Z̄ := W Diag(μ̄) W^⊤);
                else
                    (X̃, ỹ, Z̃) ← (1 − γ)(V Diag(λ̄) V^⊤, ȳ, W Diag(μ̄) W^⊤)
                    +γ(X^0, y^0, Z^0);
                    solve (SDP-D) with initial point (X̃, ỹ, Z̃);
                    return solution of (SDP-D);
            else
                solve (SDP-D) with coldstart;
else
    solve (SDP-D) with coldstart;
```

has to be solved, where $W\mathrm{Diag}(\hat{\mu})W^\top = \hat{Z} := \sum_{i \le m} A_i \hat{y}_i - C$ is an eigenvector decomposition of the optimal slack matrix of the parent node, which again is a linear program. By the same arguments as before, (D-R) gives an upper bound on the optimal objective value of (SDP-D), so

$$\mathrm{optval}(\text{P-R}) \le \mathrm{optval}(\text{SDP-P}) \le \mathrm{optval}(\text{SDP-D}) \le \mathrm{optval}(\text{D-R}).$$

Therefore, if optval(P-R) = optval(D-R), then also optval(SDP-D) = optval(D-R), and since every feasible solution of (D-R) is also feasible for (SDP-D), an optimal solution to (SDP-D) could be found by solving linear problems only. Otherwise, the optimal solutions $(\bar{\lambda}, \bar{y}, \bar{\mu})$ form primal and dual feasible solutions $(\bar{X} := V\mathrm{Diag}(\bar{\lambda})V^\top, \bar{y}, \bar{Z} := W\mathrm{Diag}(\bar{\mu})W^\top)$ of (SDP-P) and (SDP-D). To get a solution in the relative interior, a convex combination with either a default initial point or an earlier iterate of the parent node can be taken.

Note that since (P-R) and (D-R) are restrictions of (SDP-P) and (SDP-D), they may be infeasible even though (SDP-P) and (SDP-D) are feasible, in which case no information can be extracted from them. In this case the warmstarting procedure has to be aborted and (SDP-P) and (SDP-D) have to be solved with a coldstart. The whole procedure is given in Algorithm 2.

Table 5: Solving times for different warmstarting techniques for the complete testset

settings	solved	time	SDP-iter	penalty	unsolved
no warmstart	290	117.85	22 827.93	6.72 %	8.46 %
unadjusted warmstart	126	821.82	–	18.78 %	22.50 %
earlier iterate: gap 0.01	172	396.93	–	4.50 %	3.08 %
earlier iterate: gap 0.5	252	213.88	26 923.91	8.01 %	10.66 %
convcomb: 0.01 scaled (pdsame) id	288	113.60	19 697.25	8.65 %	10.90 %
convcomb: 0.5 scaled (pddiff) id	289	108.60	18 307.29	9.29 %	9.16 %
convcomb: 0.5 scaled (pdsame) id	290	109.92	19 684.70	6.05 %	8.60 %
convcomb: 0.5 analcent	288	140.21	25 351.48	13.55 %	10.65 %
projection	289	112.87	20 195.03	6.13 %	9.27 %
roundingprob 0.5 id	281	180.95	16 955.37	6.73 %	8.76 %
roundingprob inf only	289	159.66	18 521.50	6.76 %	8.50 %

Implementation and Numerical Results All of the described techniques have been implemented for the SDPA [107, 108] interface of SCIP-SDP. The earlier iterate and convex combination techniques have also been implemented for DSDP [13], while the other techniques could not be implemented for DSDP, since it only allows to specify the dual vector y for the initial point, but not the primal and dual matrices X and Z. Since MOSEK 8 [78] does not allow to specify an initial point for its conic interior-point solver, none of the techniques can currently be used in conjunction with MOSEK.

Numerical experiments have been conducted on the testset of all 194 MISDP instances of the conic benchmark library [33], which were originally proposed in [36], and an additional 126 instances from a compressed sensing application proposed in [35]. The tests were carried out on a cluster of 64-bit Intel Xeon E5-1620 CPUs running at 3.50GHz with 32GB RAM using SDPA 7.4.0 together with preliminary developer versions of SCIP 5.0 and SCIP-SDP 3.1. We omit results for DSDP, since only some of the techniques could be compared and the effects of changing the initial point are greatly diminished for DSDP because of its internal penalty formulation. The results for SDPA are given in Table 5 with the number of solved instances, the time as shifted geometric mean, the number of SDP-iterations as the shifted geometric mean (shift 1000) over all instances solved to optimality by all settings except for the unadjusted warmstart and the warmstart with a preoptimal solution of gap at most 10%, the average percentage of SDP relaxations that could only be solved using the penalty formulation explained in [36] and the amount of SDP relaxations that could not be solved at all.

The results show that a direct warmstart with the unadjusted solution of the parent node, like for the dual simplex, is not a good idea. Also restarting with earlier iterations did not work too well within SCIP-SDP, although using 50% gap solutions at least led to a small speedup for the minimum-k-partitioning instances. Taking the convex combination with a scaled identity matrix (with the scaling factors chosen dynamically based on the specific instance and with either the same or different scaling factors for the primal and dual problem) seemed to be the best approach for SCIP-SDP, with all three variants given in Table 5 leading to small speedups overall. Looking at the specific instance sets, there are even some quite significant speedups. Taking only 1% of the interior-point is the fastest approach for both cardinality-constrained least-squares and minimum-k-partioning, with speedups of 11% and 16%, respectively, but is 41% slower than the coldstart for the truss topology instances. The 50% combination with different scaling factors in the primal and dual leads to the largest speedup on any testset, with 28% on the compressed sensing instances, and is also the fastest version overall with a speedup of 8%. The version with equal scaling factors for the primal and dual is a bit more stable, however, leading only to a 15% slowdown for truss topology and being significantly faster than the cold start on all other instance sets. Using the analytic center of the root node relaxation instead of a scaled identity matrix did not seem to work to well within SCIP-SDP, with a significantly increased percentage of relaxations

needing the penalty formulation. Projecting onto positive definite matrices again led to a speedup, although a bit less than the convex combinations, but of all warmstarting techniques, this was the one that led to the smallest slowdown on the truss topology testset with only 8% in comparison to coldstarts.

The rounding problems led to the smallest amount of SDP iterations, showing that they can indeed be used to reduce the amount of work spent in interior-point solvers, but the additional LP solves turned out to be too expensive, even if only the primal rounding problems were solved to detect infeasibility or suboptimality. For the cardinality-constrained least-squares instances, when solving both primal and dual rounding problems, more time was spent in the rounding problems than it takes to solve the MISDP via coldstarts, and also for the compressed sensing instances more than 50% of the time was spent in the rounding problems. For the other two instance sets with sparser matrices the amount of time was much less, but the results were also quite disappointing, with the solution of the dual rounding problem failing regularly, so that the problems had to be coldstarted in the end. For the first two applications, however, the approach actually succeeded cutting off a significant number of SDP-relaxations. For the cardinality-constrained least-squares testset over 25% of all infeasible relaxations could be cut off via the dual rounding problem, by solving a linear program only. For the compressed sensing application over 50% of all relaxations could be cut off via bounding through the primal rounding problem, and the dual rounding problem could be solved for a valid initial point in almost all of the remaining cases. It should be noted, however, that these numbers include all solved SDP-relaxations, including easier problems solved in heuristics. Actually almost all of the infeasible problems cut off for the cardinality-constrained least-squares applications came from heuristics, which would have been easier to solve anyways, and also for the compressed sensing application, suboptimality was detected much more often for the problems stemming from heuristics, but in this case still almost 30% of all nodes could be cut off via rounding problems. This shows that the rounding problems could in theory lead to a speedup if the solving times for the rounding problems could be decreased in the implementation, for example by warmstarting them using the dual simplex or by eliminating the smallest eigenvalues of the parent node from the formulation.

Overall it can be seen that significant speedups can be gained from warmstarting, but no approach seems to work well for all applications. Therefore, warmstarts are disabled by default in SCIP-SDP 3.1, but can be activated by the corresponding parameters of the SDP relaxation handler, for more details see the documentation of SCIP-SDP.

5 The UG Framework

The *Ubiquity Generator* framework UG is a generic framework to parallelize state-of-the-art branch-and-bound solvers, which are referred to as the *base solvers*, from "outside." UG is composed of a collection of C++ base classes, which define interfaces that can be customized for any base solver and communication library. Shinano [94] gives a general overview of the UG framework. For SCIP, a UG parallelization has been implemented both for shared and distributed memory computing environments, called FiberSCIP [96] and ParaSCIP [95], respectively.

UG 0.8.5 provides minor modifications necessary to comply with changes in SCIP 5.0, a small revision of the parallelization for the Steiner tree solver SCIP-Jack, and most notably a first parallelization of the MISDP solver SCIP-SDP.

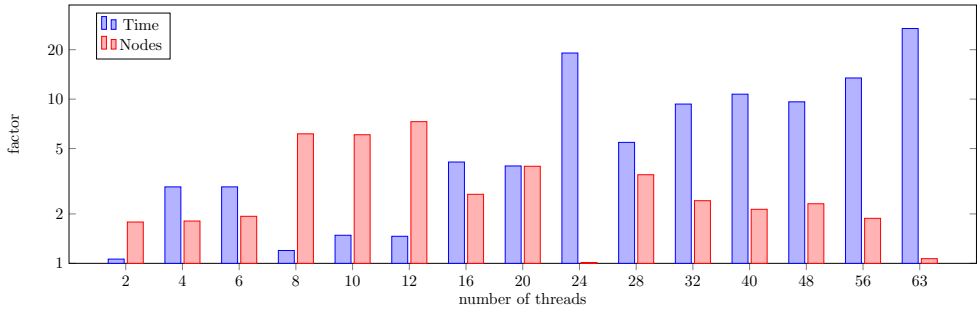

Figure 1: Speedup factors and relative changes in the number of nodes for increasing numbers of slave threads used by UG-MISDP on CBLIB instance `5x5_1bar`.

5.1 Revisions to the Steiner Tree Parallelization

A parallel version of the Steiner tree problem solver SCIP-Jack (see Section 4.2) is included as the UG application "STP". After revising the interface with SCIP, the current release provides improved stability of this parallelization. While SCIP-Jack can handle a variety of different Steiner tree problem variants, the UG parallelization is currently only supported for the classical Steiner tree problem in graphs.

In order to use parallel SCIP-Jack most efficiently, users are advised to tune the UG parameters depending on the problem type they want to solve. The default parameter settings in this release package set `racing ramp-up` and disable presolving inside the LoadCoordinator. These settings were selected for instances that can be solved already by sequential SCIP-Jack in at most one hour. In contrast, Gamrath et al. [40] chose special settings targeted towards solving open instances. Here, each instance was presolved once and solved at its root node in the LoadCoordinator. Then root node cuts were added to the presolved instance and the parallel solving phase was initiated using `normal ramp-up`.

5.2 A New Parallel MISDP Solver

UG 0.8.5 comes with a preliminary beta version of a new application to parallelize the tree search of the MISDP solver SCIP-SDP described in Section 4.3 on both shared memory and distributed memory computing environments. The parallelized tree search can additionally be combined with the internal parallelization of some of the underlying SDP solvers.

At this stage, a systematic performance analysis of this implementation, especially in comparison to a parallelization of the SDP solving, is still pending. However, in order to exemplify the potential gains, preliminary computational results on a single instance of the conic benchmark library [33] with a single-threaded solving time of 4931 seconds and 264 092 nodes are presented in Figure 1. The experiment was conducted on a shared memory cluster of 64 Intel Xeon E3-4650 CPUs running at 2.7GHz with 1 TB RAM. The underlying SDP solver used was MOSEK 8.1.0.25 [78]. As can be seen, the parallelization can lead to speedup factors of up to 27 when employing 64 threads in total (one master and 63 slave threads). However, as is to be expected when looking at the behavior for just a single instance, a rather large variability can be observed in the number of branch-and-bound nodes. This again leads to a large variability of speedup factors.

6 Final Remarks

The goal of this article was to provide an overview of the SCIP Optimization Suite 5.0. In particular, we highlighted the performance improvements and how they have been achieved. Moreover, we presented changes in the framework that might be helpful for users, and we hope that these are inspiring to the readers.

Of course, there are many ideas that we will work on in the future—not only with respect to performance improvements, but also for extending the scope of the SCIP Optimization Suite and its related software packages. We are always happy for suggestions, collaborations, and contributions.

Acknowledgements

The authors want to thank all contributors to the SCIP Optimization Suite. Special thanks go to Roland Wunderling, the creator of SoPlex, to Tobias Achterberg, the creator of SCIP, to Marco Lübbecke, who drives the GCG development, to Alexander Martin who developed SIP, the predecessor of SCIP, and to all former and present developers of SCIP, GCG, and SCIP-SDP.

We wish to thank Katsuki Fujisawa at IMI, Kyushu University, for allowing us to use his computing factilities for the computations in Section 5.2, Chuen Teck See for creating the first parallelized version of SCIP-SDP using UG, Renke Kuhlmann for supporting us in writing and analyzing the NLP interface to WORHP, Stefan Heinz for maintaining the LP interface to Xpress, and Tobias Achterberg and Michael Winkler for helping with the LP interface to Gurobi. We are grateful to the HLRN III supercomputer staff, especially Matthias Läuter and Guido Laubender and to the ISM supercomputer staff in Tokyo, especially Tomonori Hiruta.

Code Contributions of the Authors

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

The updates to the presolving in SCIP presented in Section 2.2 have been implemented by GG (clique table analysis), by PG, AG, RG, and DW (nonzero cancellation), and by BM and SV (quadratic constraint disaggregation). The improvements in Section 2.4 have been contributed by RG (cut management and MIP separation together with FeS) and BM, FeS, and AG (bilinear term relaxations). The symmetry handling extensions of SCIP (Section 2.3) have been implemented by MP and CH. Conflict and dual proof analysis (Section 2.5) has been extended by JaW. Primal heuristics (Section 2.6) have been extended by GH (Adaptive Large Neighborhood Search) and BM and FeS (MPEC heuristic for MINLP). GG has improved the structure-based heuristics. Additional contributions to the code infrastructure from Section 2.7 have been made by RG (double-double arithmetic), GH (disjoint set data structure together with DR, bandit selection algorithms, linear constraint classification together with AG), TG (new table plugin type together with GH), MV (numerical violation computation for solutions together with GH), and by SV and BM (new NLP solver interfaces to WORHP and FilterSQP).

The improvements to SoPlex described in Section 3 have been chiefly performed by MM together with DR (for scaling) and AG (for solution polishing inside SCIP). The work on the applications in Section 4 has been conducted by LE (CycleClustering application together with JaW), DR (updates in SCIP-Jack), and TG (warmstarting procedures in SCIP-SDP). The adjustments to UG explained in Section 5 have been

implemented by YS, TG (for MISDP), and DR (for SCIP-Jack). Last, not least, GH and FrS have supported the infrastructure for performing and evaluating computational tests with their work on the tools Ipet [53] and Rubberband [88].

References

[1] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Opt.*, 4(1):4–20, 2007.

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

[3] T. Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[4] T. Achterberg and R. Wunderling. Mixed integer programming: Analyzing 12 years of progress. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, pages 449–481. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-38189-8_18.

[5] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. Technical Report 16-44, ZIB, Takustr. 7, 14195 Berlin, 2016.

[6] F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.

[7] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[8] G. Andreello, A. Caprara, and M. Fischetti. Embedding $\{0, \frac{1}{2}\}$-cuts in a branch-and-cut framework: A computational study. *INFORMS Journal on Computing*, 19(2):229–238, 2007. doi:10.1287/ijoc.1050.0162.

[9] A. Atamtürk, G. L. Nemhauser, and M. W. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40–55, 2000.

[10] E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972. doi:10.1137/0123007.

[11] H. Y. Benson. Mixed integer nonlinear programming using interior-point methods. *Optimization Methods and Software*, 26(6):911–931, 2011.

[12] H. Y. Benson and D. F. Shanno. An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming. *Computational Optimization and Applications*, 38(8):371–399, 2007.

[13] S. J. Benson and Y. Ye. Algorithm 875: DSDP5–software for semidefinite programming. *ACM Transactions on Mathematical Software*, 34(4):16:1–16:20, 2008.

[14] T. Berthold. RENS–The optimal rounding. *Mathematical Programming Computation*, 6 (1):33–54, 2014. doi:10.1007/s12532-013-0060-9.

[15] R. Borndörfer, S. Schenker, M. Skutella, and T. Strunk. PolySCIP. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016, 5th International Congress, Proceedings*, volume 9725 of *LNCS*, Berlin, Germany, 2016. Springer. doi:10.1007/978-3-319-42432-3.

[16] S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *CoRR*, abs/1204.5721, 2012. URL `http://arxiv.org/abs/1204.5721`.

[17] C. Büskens and D. Wassel. The ESA NLP solver WORHP. In G. Fasano and J. D. Pintér, editors, *Modeling and optimization in space engineering*, pages 85–110. Springer, 2012.

[18] A. Caprara and M. Fischetti. $\{0, 1/2\}$-chvátal-gomory cuts. *Mathematical Programming*, 74(3):221–235, 1996. doi:10.1007/BF02592196.

[19] S. B. Çay, I. Pólik, and T. Terlaky. Warm-start of interior point methods for second order cone optimization via rounding over optimal Jordan frames. ISE technical report 17T-

006, Lehigh University, 2017. URL http://www.optimization-online.org/DB_HTML/2017/05/5998.html.

[20] S. F. Chang and S. T. McCormick. Implementation and computational results for the hierarchical algorithm for making sparse matrices sparser. *ACM Transactions on Mathematical Software*, 19(3):419–441, 1993. doi:10.1145/155743.152620.

[21] COIN-OR. CppAD, a package for differentiation of C++ algorithms. http://www.coin-or.org/CppAD.

[22] Computational Optimization Research at Lehigh Laboratory (CORAL). MIP instances. https://coral.ise.lehigh.edu/data-sets/mixed-integer-instances/. Visited 12/2017.

[23] E. Danna. Performance variability in mixed integer programming, 2008. Presentation at Workshop on Mixed Integer Programming.

[24] E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005. doi:10.1007/s10107-004-0518-7.

[25] P. T. Darga, H. Katebi, M. Liffiton, I. L. Markov, and K. Sakallah. Saucy. http://vlsicad.eecs.umich.edu/BK/SAUCY/, 2012.

[26] L. Eifler. Mixed-integer programming for clustering in non-reversible Markov processes. Master's thesis, Technische Universität Berlin, 2017.

[27] J. M. Elble and N. V. Sahinidis. Scaling linear optimization problems prior to application of the simplex method. *Computational Optimization and Applications*, 52(2):345–371, 2012. doi:10.1007/s10589-011-9420-4.

[28] A. Engau. Recent progress in interior-point methods: Cutting-plane algorithms and warm starts. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, volume 166 of *International Series in Operations Research & Management Science*, pages 471–498. Springer Science+Business Media, 2012.

[29] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[30] M. Fischetti and M. Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, 2014. doi:10.1007/s10732-014-9266-x.

[31] R. Fletcher and S. Leyffer. User manual for filterSQP. Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Scotland, 1998.

[32] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239–269, 2002. doi:10.1007/s101070100244.

[33] H. A. Friberg. CBLIB 2014: A benchmark library for conic mixed-integer and continuous optimization. *Mathematical Programming Computation*, 8(2):191–214, 2016.

[34] E. J. Friedman. Fundamental domains for integer programs with symmetries. In A. Dress, Y. Xu, and B. Zhu, editors, *Combinatorial Optimization and Applications*, volume 4616 of *Lecture Notes in Computer Science*, pages 146–153. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-73556-4_17.

[35] T. Gally and M. E. Pfetsch. Computing restricted isometry constants via mixed-integer semidefinite programming. Technical report, Optimization Online, 2016. URL http://www.optimization-online.org/DB_HTML/2016/04/5395.html.

[36] T. Gally, M. E. Pfetsch, and S. Ulbrich. A framework for solving mixed-integer semidefinite programs. *Optimization Methods and Software*, 2017. To Appear.

[37] 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.

[38] G. Gamrath, T. Berthold, S. Heinz, and M. Winkler. Structure-based primal heuristics for mixed integer programming. In K. Fujisawa, Y. Shinano, and H. Waki, editors, *Optimization in the Real World*, volume 13 of *Mathematics for Industry*, pages 37–53. Springer Japan, 2015. doi:10.1007/978-4-431-55420-2_3.

[39] G. Gamrath, T. Berthold, S. Heinz, and M. Winkler. Structure-driven fix-and-propagate heuristics for mixed integer programming. Technical Report 17-56, ZIB, Takustr. 7, 14195 Berlin, 2017. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-65387.

[40] 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.

[41] S. Ghosh. DINS, a MIP Improvement Heuristic. In M. Fischetti and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization: 12th International IPCO Conference, Ithaca, NY, USA*, pages 310–323. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-72792-7_24.

[42] A. M. Gleixner, T. Berthold, B. Müller, and S. Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, pages 1–27, 2016. doi:10.1007/s10898-016-0450-4.

[43] A. M. 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.

[44] J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[45] J. Gondzio. Warm start of the primal-dual method applied in the cutting plane scheme. *Mathematical Programming*, 83(1):125–143, 1998.

[46] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.2 edition, 2016. `http://gmplib.org/`.

[47] Z. Gu, G. L. Nemhauser, and M. W. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85(3):439–467, 1999.

[48] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82:291–315, 1998.

[49] H. Hijazi. Perspective envelopes for bilinear functions. Technical report, Optimization Online, 2015. URL `http://www.optimization-online.org/DB_HTML/2015/03/4841.html`.

[50] H. Hijazi, P. Bonami, and A. Ouorou. An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 26(1):31–44, 2014. doi:10.1287/ijoc.1120.0545.

[51] C. Hojny and M. E. Pfetsch. Polytopes associated with symmetry handling. Technical report, Technische Universität Darmstadt, 2017.

[52] F. Hwang, D. Richards, and P. Winter. The Steiner tree problem. *Annals of Discrete Mathematics*, 53, 1992.

[53] Ipet. Interactive Performance Evaluation Tools for Optimization Software. `http://www.github.com/gregorch/ipet`.

[54] Ipopt. Interior Point OPTimizer. `http://www.coin-or.org/Ipopt/`.

[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] T. Koch. *Rapid Mathematical Prototyping*. PhD thesis, Technische Universität Berlin, 2004.

[60] 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.

[61] S. Küçükyavuz and Y. Pochet. Uncapacitated lot sizing with backlogging: the convex hull. *Mathematical Programming*, 118(1):151–175, 2009. doi:10.1007/s10107-007-0186-5.

[62] L. Liberti. Reformulations in mathematical programming: Automatic symmetry detection and exploitation. *Mathematical Programming*, 131(1-2):273–304, 2012. doi:10.1007/s10107-010-0351-0.

[63] F. Liers, A. Martin, and S. Pape. Binary Steiner trees: Structural results and an exact solution approach. *Discrete Optimization*, 21:85–117, 2016. doi:10.1016/j.disopt.2016.05.006.

[64] J. Linderoth. A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. *Mathematical Programming*, 103(2):251–282, 2005.

[65] M. Locatelli. Convex envelopes of bivariate functions through the solution of KKT systems. Technical report, Optimization Online, 2016. URL `http://www.optimization-online.org/DB_HTML/2016/01/5280.html`.

[66] A. Lodi and A. Tramontani. Performance variability in mixed-integer programming. *Tutorials in Operations Research*, pages 1–12, 2013. doi:10.1287/educ.2013.0112.

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

[68] S. J. Maher, T. Fischer, T. Gally, G. Gamrath, A. Gleixner, R. L. Gottwald, G. Hendel, T. Koch, M. E. Lübbecke, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, D. Weninger, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr. 7, 14195 Berlin, 2017.

[69] H. Marchand. *A polyhedral study of the mixed knapsack set and its use to solve mixed integer programs*. PhD thesis, Université catholique de Louvain, 1998.

[70] H. Marchand and L. A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001. doi:10.1287/opre.49.3.363.11211.

[71] F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94 (1):71–90, 2002. doi:10.1007/s10107-002-0358-2.

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

[73] F. Margot. Symmetry in integer linear programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming*, pages 647–686. Springer, 2010.

[74] J. P. Marques-Silva and K. Sakallah. Grasp: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 1999.

[75] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[76] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

[77] MINLPLIB2. MINLP library 2. `http://www.gamsworld.org/minlp/minlplib2/html/`.

[78] MOSEK ApS. *The MOSEK C optimizer API manual. Version 8.1 (Revision 25)*, 2017. URL `http://docs.mosek.com/8.1/capi/index.html`.

[79] B. Müller, R. Kuhlmann, and S. Vigerske. On the performance of NLP solvers within global MINLP solvers. In *Operations Research Proceedings*, 2017. To appear.

[80] F. Ortega and L. A. Wolsey. A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, 41(3):143–158, 2003. doi:10.1002/net.10068.

[81] J. Ostrowski. *Symmetry in Integer Programming*. PhD thesis, Lehigh University, 2008.

[82] 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.

[83] M. E. Pfetsch and T. Rehn. A computational comparison of symmetry handling methods for mixed integer programs. Technical report, Optimization Online, 2015. URL `http://www.optimization-online.org/DB_HTML/2015/11/5209.html`.

[84] N. Ploskas and N. Samaras. The impact of scaling on simplex type algorithms. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 17–22, New York, NY, USA, 2013. ACM. doi:10.1145/2490257.2490283.

[85] D. Rehfeldt and T. Koch. Generalized preprocessing techniques for Steiner tree and maximum-weight connected subgraph problems. Technical Report 17-57, ZIB, Takustr. 7, 14195 Berlin, 2017.

[86] D. Rehfeldt and T. Koch. Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem. Technical Report 17-45, ZIB, Takustr. 7, 14195 Berlin, 2017.

[87] E. Rothberg. An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007. doi:10.1287/ijoc.1060.0189.

[88] Rubberband. A flexible archiving platform for optimization benchmarks. `http://www.github.com/ambros-gleixner/rubberband`.

[89] S. M. Rump. High precision evaluation of nonlinear functions. In *Proceedings of 2005 International Symposium on Nonlinear Theory and its Applications*, pages 733–736. The Institute of Electronics, Information and Communication Engineers (IEICE), 2005.

[90] D. Salvagnin. A dominance procedure for integer programming. Master's thesis, University of Padua, 2005.

[91] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA J. Comput.*, 6(4):445–454, 1994.

[92] L. Schewe and M. Schmidt. Computing feasible points for MINLPs with MPECs. Technical report, Optimization Online, 2016. URL `http://www.optimization-online.org/DB_HTML/2016/12/5778.html`.

[93] N. C. Schwertman and D. M. Allen. Smoothing an indefinite variance-covariance matrix. *Journal of Statistical Computation and Simulation*, 9(3):183–194, 1979.

[94] Y. Shinano. The Ubiquity Generator framework: 7 years of progress in parallelizing branch-and-bound. ZIB-Report 17-60, Zuse Institute Berlin, 2017. Accepted for the Operations Research Proceedings 2017.

[95] 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. Springer, 2012. doi:10.1007/978-3-642-24025-6_12.

[96] 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.

[97] A. Skajaa, E. D. Andersen, and Y. Ye. Warmstarting the homogeneous and self-dual interior point method for linear and conic quadratic problems. *Mathematical Programming Computation*, 5(1):1–25, 2013.

[98] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

[99] R. J. Vanderbei and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13(1–3):231–252, 1999.

[100] S. Vigerske and A. Gleixner. SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods & Software*, to appear. doi:10.1080/10556788.2017.1335312.

[101] F. Wesselmann and U. H. Suhl. Implementing cutting plane management and selection techniques. Technical report, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany, 2012.

[102] J. Witzig, I. Beckenbach, L. Eifler, K. Fackeldey, A. Gleixner, A. Grever, and M. Weber. Mixed-integer programming for cycle detection in non-reversible Markov processes. *Multiscale Modeling and Simulation*, 2016. Accepted for publication.

[103] J. Witzig, T. Berthold, and S. Heinz. Experiments with conflict analysis in mixed integer programming. In D. Salvagnin and M. Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, pages 211–220. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-59776-8_17.

[104] K. Wolter. Implementation of cutting plane separators for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006.

[105] WORHP. We Optimize Really Huge Problems. `https://worhp.de/`.

[106] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.

[107] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0). *Optimization Methods and Software*, 18: 491–505, 2003.

[108] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto. A high-performance software package for semidefinite programs: SDPA 7. Technical Report Research Report B-460, Dept. of Mathematical and Computing Science, Tokyo Institute of Technology, 2010.