Recent Developments in SCIP

Ksenia Bestuzheva, bestuzheva@zib.de

5th ZIB-RIKEN-IMI-ISM MODAL Workshop on Optimization,
Data Analysis and HPC in AI

September 28, 2021

SPONSORED BY THE
Federal Ministry
of Education
and Research

The SCIP Optimization Suite

A toolbox for generating and solving MINLPs and CIPs:

- **SCIP**: MINLP solver and constraint programming framework,
- **SoPlex**: LP solver,
- **PaPILO**: parallel presolver for integer and linear optimization,
- **ZIMPL**: mathematical programming language,
- **UG**: parallel framework for MINLPs,
- **GCG**: generic branch-cut-and-price solver,
- **SCIP-Jack**: solver for Steiner tree problems.

Overview of Changes

SCIP:

- New framework for handling nonlinear constraints
- Improved symmetry handling
- Mixing/conflict cuts
- Decomposition and PADM heuristics

Interface improvements:

- PySCIPOpt
- Julia
- MATLAB
- C wrapper for SoPlex

PaPILO:

- Dual postsolving and integration into SoPlex
- Conflict analysis

SCIP-SDP:

- Revised handling of relaxations
- New heuristic
- New presolving methods

SCIP-Jack:

- Major performance improvements
- Better than state-of-the-art for Euclidian STP and almost all benchmark sets for STP on graphs

Exact SCIP:

- Substantial revision and extension of the original framework
- Average speedup of 10.7x
- Available at https://github.com/leoneifler/exact-SCIP

Performance Comparison with SCIP 7.0: MILP

Table: Performance comparison

		SCIP 7.0			SCIP 8.0			relative	
Subset	instances	solved	time	nodes	solved	time	nodes	time	nodes
all	1423	1191	355.4	3714	1208	300.8	3341	0.85	0.90
affected	1222	1159	255.3	2993	1176	212.0	2705	0.83	0.90
[0,tilim]	1254	1191	236.7	2735	1208	195.8	2477	0.83	0.91
[1,tilim]	1215	1152	278.1	3050	1169	228.8	2753	0.82	0.90
[10,tilim]	1142	1079	362.8	3767	1096	294.9	3384	0.81	0.90
[100,tilim]	881	818	752.8	6952	835	592.0	5941	0.79	0.85
[1000,tilim]	413	350	2325.8	23993	367	1674.0	18051	0.72	0.75
diff-timeout	109	46	5094.2	33875	63	2563.7	16770	0.50	0.50
both-solved	1145	1145	176.5	2138	1145	153.1	2055	0.87	0.96
MIPLIB 2010	435	380	374.0	5679	392	319.4	5554	0.85	0.98
MIPLIB 2017	639	493	601.8	5592	490	541.2	5544	0.90	0.99

Performance Comparison with SCIP 7.0: MINLP

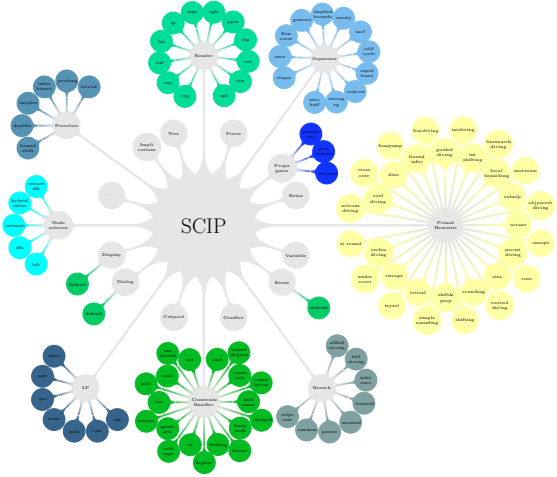
Table: Performance comparison

		SCIP 7.0			SCIP 8.0			relative	
Subset	instances	solved	time	nodes	solved	time	nodes	time	nodes
all	473	434	19.0	1433	419	19.9	1055	1.04	0.74
affected	424	407	16.9	1693	392	17.6	1235	1.04	0.73
[0,tilim]	451	434	14.6	1409	419	15.2	1044	1.05	0.74
[1,tilim]	338	321	32.8	3311	306	34.8	2287	1.06	0.69
[10,tilim]	237	220	83.1	7350	205	93.7	4937	1.13	0.67
[100,tilim]	142	125	183.0	12592	110	257.9	8460	1.41	0.67
[1000,tilim]	66	49	179.1	5818	34	899.1	20131	5.02	3.46
diff-timeout	49	32	115.4	2202	17	1131.6	19016	9.81	8.64
both-solved	402	402	11.2	1333	402	8.7	711	0.78	0.53
convex	152	140	19.0	1012	121	47.9	3181	2.52	3.14
nonconvex	321	294	19.1	1685	298	13.0	604	0.68	0.36

Number of failures descreased from 25 to 5.

SCIP (Solving Constraint Integer Programs)

- Provides a **full-scale MILP and MINLP solver**,
- is **constraint based**,
- incorporates
  - **MIP features** (cutting planes, LP relaxation),
  - **MINLP features** (spatial branch-and-bound, OBBT)
  - **CP features** (domain propagation),
  - **SAT-solving features** (conflict analysis, restarts),
- is a **branch-cut-and-price framework**,
- has a modular structure via **plugins**,
- is **free for academic purposes**,
- and is **available in source-code** under http://scip.zib.de !

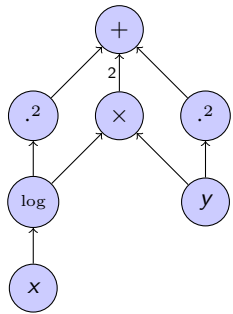


Extended Formulations of Nonlinear Constraints

- Expressions are represented as **expression graphs**,
- **Auxiliary variables** are introduced for subexpressions, used in **relaxations only**
- The original formulation is kept
- This avoids wrong feasibility checks

Example: $\log(x)^2 + 2\log(x)y + y^2 \rightarrow$

$$
\begin{aligned}
& w_1, \\
& w_2 + 2w_3 + w_4 = w_1, \\
& w_5^2 = w_2, \\
& w_5 y = w_3, \\
& y^2 = w_4, \\
& \log(x) = w_5
\end{aligned}
$$

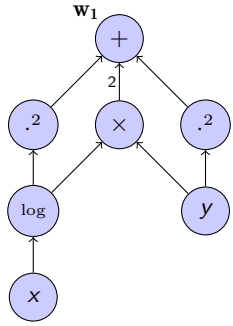


Extended Formulations of Nonlinear Constraints

- Expressions are represented as **expression graphs**,
- **Auxiliary variables** are introduced for subexpressions, used in **relaxations only**
- The original formulation is kept
- This avoids wrong feasibility checks

Example: $\log(x)^2 + 2\log(x)y + y^2 \rightarrow$

$$
\begin{aligned}
& w_1, \\
& w_2 + 2w_3 + w_4 = w_1, \\
& w_5^2 = w_2, \\
& w_5 y = w_3, \\
& y^2 = w_4, \\
& \log(x) = w_5
\end{aligned}
$$

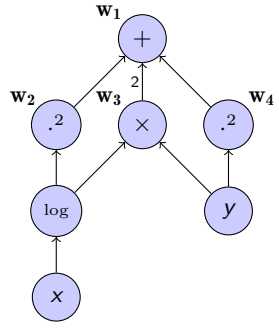


Extended Formulations of Nonlinear Constraints

- Expressions are represented as **expression graphs**,
- **Auxiliary variables** are introduced for subexpressions, used in **relaxations only**
- The original formulation is kept
- This avoids wrong feasibility checks

Example: $\log(x)^2 + 2\log(x)y + y^2 \rightarrow$

$$
\begin{aligned}
& w_1, \\
& w_2 + 2w_3 + w_4 = w_1, \\
& w_5^2 = w_2, \\
& w_5 y = w_3, \\
& y^2 = w_4, \\
& \log(x) = w_5
\end{aligned}
$$

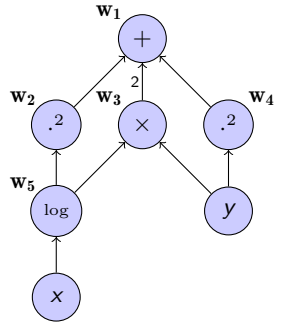


Extended Formulations of Nonlinear Constraints

- Expressions are represented as **expression graphs**,
- **Auxiliary variables** are introduced for subexpressions, used in **relaxations only**
- The original formulation is kept
- This avoids wrong feasibility checks

Example: $\log(x)^2 + 2\log(x)y + y^2 \rightarrow$

$$
\begin{aligned}
& w_1, \\
& w_2 + 2w_3 + w_4 = w_1, \\
& w_5^2 = w_2, \\
& w_5 y = w_3, \\
& y^2 = w_4, \\
& \log(x) = w_5
\end{aligned}
$$



Expression and Nonlinearity Handlers

- **Separate expression operators** ($+$, $\times$) and **high-level structures** (quadratic, semi-continuous, second order cone, etc.)
- **Expression handlers** implement functionality for **expression operators**: evaluation, differentiation, interval evaluation and bound tightening, etc.
- **Nonlinearity handlers** implement functionality for **high-level structures**: separation, propagation, etc.
- **Avoid redundancy / ambiguity** of expression types

MINLP Features

- Improved bound propagation for **quadratic** expressions
- **Intersection** cuts
- Separation for 2×2 **principal minors** for constraints $X = xx^T$
- Tight linear relaxations for **second order cones**
- Tight convex relaxations for **bilinear** products
- Reformulation Linearisation Technique cuts for implicit and explicit **bilinear** products
- Tight linear relaxations for **convex** and **concave** expressions
- Generalised **perspective** cuts for functions of semi-continuous variables
- **Symmetry** detection
- Linearization of **products of binary variables**

Symmetry Handling Techniques in SCIP

Existing Features in SCIP 7

- Handling of symmetries of binary variables via
  - symretope-based constraint handlers, or
  - orbital fixing
- Detection and handling of certain actions of symmetric groups

New Features in SCIP 8

- Handling of symmetries of general variables via cuts from the Schreier-Sims table (Salvagnin 2018)
- Refined detection routine of symmetric group actions
- Adapted strategy to select symmetry handling routines for an individual instance
- Handling symmetries in MINLPs

Algorithmic Enhancements of Symmetry Constraint Handlers

- Improved running time of separation routine for symresack constraints
- Improved propagation routines for symresack and orbisack constraints find all possible local variable fixings for these constraints
- Parser for symresack and orbisack constraints for .cip files

Mixing/conflict cuts

Normalized variable lower/upper bounds of $y \in [\ell, u]$

$$y \geq \ell + a_i x_i,\ x_i \in \{0,1\},\ i \in \mathcal{N}, \tag{1}$$
$$y \leq u - a_j x_j,\ x_j \in \{0,1\},\ j \in \mathcal{M}. \tag{2}$$

Mixing set

$$\mathcal{X} = \left\{ (x,y) \in \{0,1\}^{|\mathcal{N} \cup \mathcal{M}|} \times \mathbb{R} : (1),\ (2) \right\}.$$

Mixing (Atamtürk et al. (2001)) and **conflict cut separator**: generate cuts based on $\mathcal{X}$.

Performance impact

- $1.2\times$ speed-up on the testset studied in Zhao et al. (2017).
- Neutral on testset mipdev-solvable.

Decomposition Heuristic: Dynamic Partition Search

MIP with linking constraints:

$$\begin{aligned} \min_{x_q} \quad & \sum_{q \in \mathcal{K}} c_q x_q \\ \text{s.t.} \quad & x_q \in P_q \qquad \forall\, q \in \mathcal{K} \\ & \sum_{q \in \mathcal{K}} A_q x_q \leq b \end{aligned}$$

Reformulation:

$$\begin{aligned} \min_{x_q, p_q} \quad & \sum_{q \in \mathcal{K}} c_q x_q \\ \text{s.t.} \quad & x_q \in P_q \qquad \forall\, q \in \mathcal{K} \\ & A_q x_q \leq p_q \qquad \forall\, q \in \mathcal{K} \\ & \sum_{q \in \mathcal{K}} p_q = b \end{aligned}$$

p_q describe partition of right-hand side between blocks.

Goal:
Search partition of feasible solution.

Step 1: Guess initial partition p^0 satisfying $\sum_{q \in \mathcal{K}} p_q = b$.

Step 2: Check if all blocks have a feasible solution.

Step 3: All blocks feasible
$\Rightarrow$ feasible solution found

At least one block infeasible
$\Rightarrow$ update partition depending on violations,
go to Step 2

Reoptimization in the Penalty Alternating Direction Method Heuristic

MIP with linking variables:

$$\min_{x_q, z} \sum_{q \in \mathcal{K}} c_q x_q + dz$$
$$\text{s.t. } (x_q, z) \in P_q \qquad \forall\, q \in \mathcal{K}$$

Reformulation:

- Copy linking variables z
- Penalize difference
- Blockproblem q:

$$\min_{x_q, z_q} \sum_{i \in \mathcal{K} \setminus q} \lambda |z_q - \bar{z}_i|$$
$$\text{s.t. } (x_q, z_q) \in P_q \qquad \forall\, q \in \mathcal{K}$$

Algorithm:
Solve blockproblems on alternating basis.
If the linking variables don't reconcile after a couple of iterations, the penalty parameters λ are increased.
Repeat.

Reoptimization:
If PADM found a solution, fix linking variables and reoptimize with original objective function to improve solution quality.
Fixed problem is smaller and easier to solve.
In addition, use small solving limits.

Interfaces

- **PySCIPOpt**: added interface for the cut selector plugin (more in M. Turner's talk)
- **Julia interface** SCIP.jl:
  - Direct SCIP: low-level interface following the SCIP C interface, automatically generated from SCIP header files
  - MathOptInterface.jl: unified API to interact with solvers for structured constrained optimization
  - Optional precompiled SCIP binaries shipped with the Julia package, removing the need for compilation by users
- A new **MATLAB** interface:
  - Also runs under Linux and MacOS
  - Works for Octave (but at the moment a bug in Octave blocks the usage of the nonlinear part)
  - Now also fully works for SCIP-SDP
- **C wrapper for SoPlex**: shared library and header file

PaPILO

- **Dual postsolve**: ability to postsolve the dual solution → include in SoPlex
- **Conflict analysis**: reduces conflicts by analysing internal conflicts and conflicts between the presolvers → reduces the number of rounds
- **Conflict analysis**: reorder presolvers to reduce conflicts

		PaPILO 1.0.3		PaPILO 2.0.0			
subset	instances	time [s]	rounds	time [s]	relative	rounds	relative
[0,tilim]	240	0.294557	19.57	0.281663	0.956	18.50	0.945
[0.01,tilim]	206	0.349799	22.15	0.334085	0.955	20.91	0.944
[0.1,tilim]	111	0.693586	33.02	0.660184	0.952	30.88	0.935
[1,tilim]	26	2.633462	43.69	2.477353	0.941	40.73	0.932

SCIP-SDP

A framework for solving MISDPs.

- Revised **handling of SDP relaxations**
- **Decreased the memory footprint** of SCIP-SDP
- A **new heuristic** that rounds integer variables based on fractional values in the last SDP relaxation
- **New presolving** techniques
- Allow **solving LP relaxations** instead of SDP relaxations
- Handling of **rank 1** constraints
- ...and more

SCIP-Jack

SCIP-Jack: Solver for the classic **Steiner tree problem** in graphs (SPG) and 14 related problems.

In SCIP Optimization Suite 8:

- Major improvements on several problem classes, including SPG
- Better results on almost all SPG benchmark sets than state-of-the-art solver by Polzin and Vahdati Daneshmand (had been unchallenged for almost 20 years)
- Even better results for the Euclidean Steiner tree problem than state-of-the-art geometric Steiner tree solver GeoSteiner 5.1 (Juhl et al., 2018). On largest benchmark set of 15 instances with $100\,000$ terminals in the plane:
  - GeoSteiner solves 3 within **one week**
  - SCIP-Jack solves all 15 in **11 minutes**