Recent Developments in SCIP

Mathieu Besançon, besancon@zib.de

ROADEF 2022

February 25, 2022

SPONSORED BY THE
Federal Ministry of Education and Research

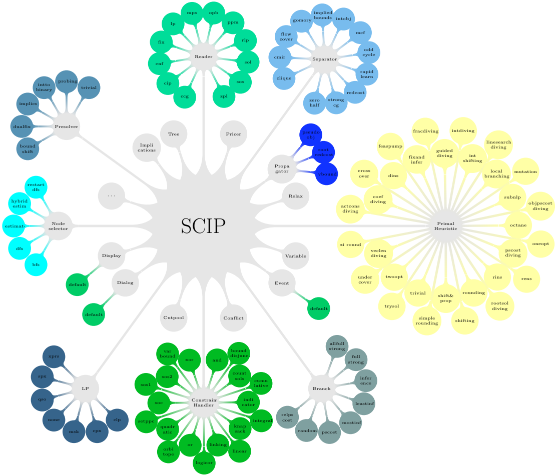
The SCIP Optimization Suite

A toolbox for generating and solving MI(N)LPs and Constraint Integer Programs (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,
- **SCIP-SDP**: for mixed-integer semidefinite problems.

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 https://www.scipopt.org/ and https://github.com/scipopt.



Overview of Changes

More details in the SCIP 8 release report [Bestuzheva et al., 2021].

SCIP:

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

PaPILO:

- Dual postsolving and integration into SoPlex
- Conflict analysis

SCIP-SDP:

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

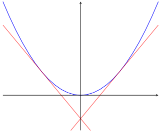
SCIP-Jack:

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

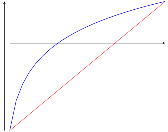
Solve: Bounding and Branching

LP relaxation via convexification and linearization:

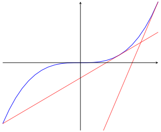
convex functions



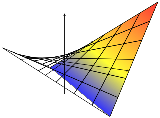
concave functions



$x^k \quad (k \in 2\mathbb{Z}+1)$



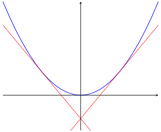
$x \cdot y$



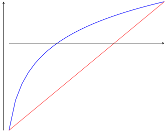
Solve: Bounding and Branching

LP relaxation via convexification and linearization:

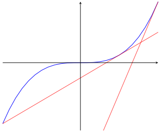
convex functions



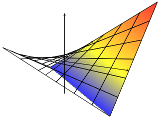
concave functions



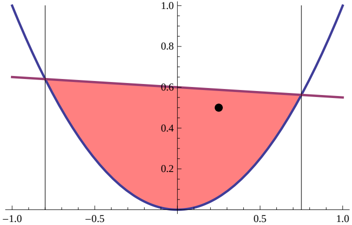
$x^k \quad (k \in 2\mathbb{Z}+1)$

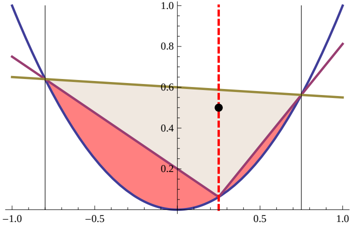


$x \cdot y$



Branching on variables in violated nonconvex constraints:

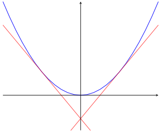




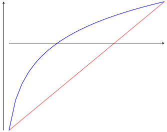
Solve: Bounding and Branching

LP relaxation via convexification and linearization:

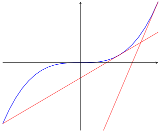
convex functions



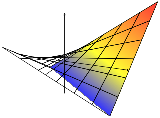
concave functions



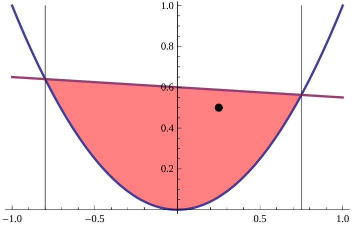
$x^k \quad (k \in 2\mathbb{Z}+1)$

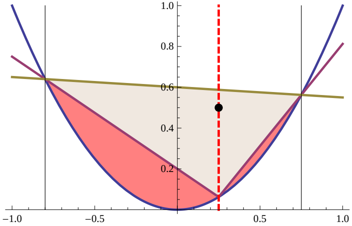


$x \cdot y$



Branching on variables in violated nonconvex constraints:





...and **bound-tightening** (FBBT, OBBT), **primal heuristics** (e.g., sub-NLP/MIP/MINLP), other **special techniques**

Problem with former (<= SCIP 7) implementation

Consider

$$\begin{aligned} &\min z \\ &\text{s.t. } \exp(\ln(1000) + 1 + xy) \leq z \\ &\quad\quad x^2 + y^2 \leq 2 \end{aligned}$$

An optimal solution:
$x = -1$
$y = 1$
$z = 1000$

Reformulated problem

Reformulation **takes apart** $\exp(\ln(1000) + 1 + x\,y)$, thus **SCIP actually solves**

$$\begin{aligned} \min\ & z \\ \text{s.t. } & \exp(w) \leq z \\ & \ln(1000) + 1 + x\,y = w \\ & x^2 + y^2 \leq 2 \end{aligned}$$

Reformulated problem

Reformulation **takes apart** $\exp(\ln(1000) + 1 + xy)$, thus **SCIP actually solves**

	Violation
$\min z$	
s.t. $\exp(w) \leq z$	$0.4659 \cdot 10^{-6} \leq$ `numerics/feastol` ✓
$\ln(1000) + 1 + xy = w$	$0.6731 \cdot 10^{-6} \leq$ `numerics/feastol` ✓
$x^2 + y^2 \leq 2$	$0.6602 \cdot 10^{-6} \leq$ `numerics/feastol` ✓

Solution (`found by relaxation`):

x = -1.000574549
y = 0.999425451
z = 999.999656552
w = 6.907754936

Error on original constraint: $6.7 \cdot 10^{-4}$: not acceptable with tol 10^{-6}!

Reformulated problem

Reformulation **takes apart** $\exp(\ln(1000) + 1 + xy)$, thus **SCIP actually solves**

	Violation
$\min z$	
s.t. $\exp(w) \leq z$	$0.4659 \cdot 10^{-6} \leq$ `numerics/feastol` ✓
$\ln(1000) + 1 + xy = w$	$0.6731 \cdot 10^{-6} \leq$ `numerics/feastol` ✓
$x^2 + y^2 \leq 2$	$0.6602 \cdot 10^{-6} \leq$ `numerics/feastol` ✓

Solution (`found by relaxation`):

x = -1.000574549
y = 0.999425451
z = 999.999656552
w = 6.907754936

Error on original constraint: $6.7 \cdot 10^{-4}$: not acceptable with tol 10^{-6}!

⇒ **Explicit reformulation** of constraints ...

- ... **loses the connection to the original problem**.

Reformulated problem

Reformulation **takes apart** $\exp(\ln(1000) + 1 + xy)$, thus **SCIP actually solves**

	Violation
$\min z$	
s.t. $\exp(w) \leq z$	$0.4659 \cdot 10^{-6} \leq$ `numerics/feastol` ✓
$\ln(1000) + 1 + xy = w$	$0.6731 \cdot 10^{-6} \leq$ `numerics/feastol` ✓
$x^2 + y^2 \leq 2$	$0.6602 \cdot 10^{-6} \leq$ `numerics/feastol` ✓

Solution (`found by relaxation`):

x = -1.000574549
y = 0.999425451
z = 999.999656552
w = 6.907754936

Error on original constraint: $6.7 \cdot 10^{-4}$: not acceptable with tol 10^{-6}!

$\Rightarrow$ **Explicit reformulation** of constraints ...

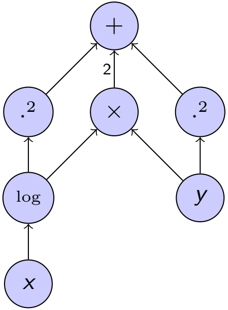
- ... **loses the connection to the original problem**.
- ... **loses distinction between original and auxiliary variables**. Thus, we may branch on auxiliary variables.
- ... **prevents simultaneous exploitation of overlapping structures**.

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

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

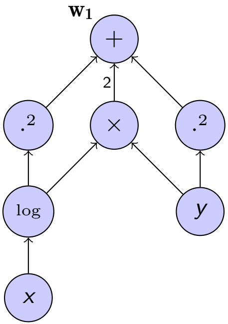


Extended Formulations of Nonlinear Constraints

- 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: $\log(x)^2 + 2\log(x)y + y^2 \to$

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

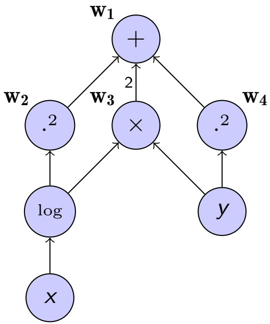


Extended Formulations of Nonlinear Constraints

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

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

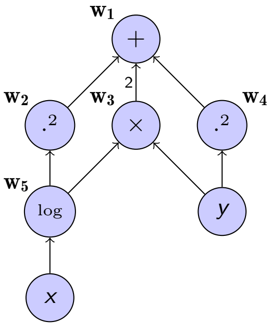


Extended Formulations of Nonlinear Constraints

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

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



Exploiting structure

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$
Smarter reformulation:

- Recognize that $\log(x)^2 + 2\log(x)y + y^2$ is **convex in** $(\log(x), y)$.

Exploiting structure

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$
Smarter reformulation:

- Recognize that $\log(x)^2 + 2\log(x)y + y^2$ is **convex in** $(\log(x), y)$.

$\Rightarrow$ Introduce auxiliary variable for $\log(x)$ only.

$$\begin{aligned} w^2 + 2wy + y^2 &\leq 4 \\ \log(x) &= w \end{aligned}$$

Handle $w^2 + 2wy + y^2 \leq 4$ as convex constraint ("gradient-cuts").

Exploiting structure

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$
Smarter reformulation:

- Recognize that $\log(x)^2 + 2\log(x)y + y^2$ is **convex in** $(\log(x), y)$.

$\Rightarrow$ Introduce auxiliary variable for $\log(x)$ only.

$$\begin{aligned} w^2 + 2wy + y^2 &\leq 4 \\ \log(x) &= w \end{aligned}$$

Handle $w^2 + 2wy + y^2 \leq 4$ as convex constraint ("gradient-cuts").

Nonlinearity Handler (nlhdlrs):

- Adds **additional separation and/or propagation** algorithms for structures that can be identified in the expression graph.
- **Attached to nodes in expression graph**, but **does not define expressions** or constraints.
- Examples: quadratics, convex subexpressions, vertex-polyhedral

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

Most are derived from the new constraint expression framework.

- 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:

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

Reformulation:

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

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:

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

Reformulation:

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

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

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.
- **Julia interface** SCIP.jl:
  - Direct SCIP: low-level interface following the SCIP C interface, automatically generated.
  - MathOptInterface.jl & JuMP: unified API to interact with solvers for constrained optimization.
  - Optional precompiled SCIP binaries shipped with the Julia package, no 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
- **SoPlex**:
  - New C shared library and header file
  - Julia interface coming soon.

PaPILO

Parallel presolving library for MILPs. Can act as a front-end for SCIP or HiGHS.
Standalone binary or integrable within other solvers.

- **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

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**

Conclusion

- Progress on several fronts, SCIP 8, a good and dense release!
- Documentation, help, download: `https://scipopt.org/`
- View source, open issue, discuss: `https://github.com/scipopt`

References I

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M. E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., and Witzig, J. (2021).
The SCIP Optimization Suite 8.0.
ZIB-Report 21-41, Zuse Institute Berlin.

Salvagnin, D. (2018).
Symmetry breaking inequalities from the Schreier-Sims table.
In van Hoeve, W.-J., editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 521–529. Springer.

Zhao, M., Huang, K., and Zeng, B. (2017).
A polyhedral study on chance constrained program with random right-hand side.
Mathematical Programming, 166:19–64.