Zuse Institute Berlin

Takustrasse 7
D-14195 Berlin-Dahlem
Germany

STEFAN VIGERSKE AND AMBROS GLEIXNER

SCIP: Global Optimization of Mixed-Integer Nonlinear Programs in a Branch-and-Cut Framework

This report has been published in the journal Optimization Methods & Software. Please cite as:

S. Vigerske and A. Gleixner. SCIP: global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software*, 33(3):536–539, 2018, DOI:10.1080/10556788.2017.1335312

ZIB Report 16-24 (Revised version May 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

SCIP: Global Optimization of Mixed-Integer Nonlinear Programs in a Branch-and-Cut Framework

Stefan Vigerske* and Ambros Gleixner†

May 22, 2017

Abstract

This paper describes the extensions that were added to the constraint integer programming framework SCIP in order to enable it to solve convex and nonconvex mixed-integer nonlinear programs (MINLPs) to global optimality. SCIP implements a spatial branch-and-bound algorithm based on a linear outer-approximation, which is computed by convex over- and underestimation of nonconvex functions. An expression graph representation of nonlinear constraints allows for bound tightening, structure analysis, and reformulation. Primal heuristics are employed throughout the solving process to find feasible solutions early. We provide insights into the performance impact of individual MINLP solver components via a detailed computational study over a large and heterogeneous test set.

1 Introduction

Nonlinear optimization problems in finite dimension containing both discrete and continuous variables are called *mixed-integer nonlinear programs* (MINLPs). Such problems arise in many fields, like energy production and distribution, logistics, engineering design, manufacturing, and the chemical and biological sciences [8, 39, 50, 75, 84]. State-of-the-art solvers for MINLP harness a variety of algorithmic techniques and the overall computational performance of a solver crucially depends on its single constituents *and* their mutual interplay.

In this paper, we first present the extensions of the constraint integer programming framework SCIP (Solving Constraint Integer Programs) [1, 3] that enable it to solve factorable nonconvex MINLPs to ε-global optimality. SCIP is freely available in source code for academic research uses. Second, we aim to provide insights into the impact of single MINLP solver components via a detailed computational study comparing SCIP's performance with varying algorithmic features deactivated.

Formally, a general MINLP reads

$$\min\{\langle c,x\rangle : x \in X\}, \tag{1a}$$

$$X := \{x \in [\underline{x},\bar{x}] : Ax \leq b,\ g(x) \leq 0,\ x_i \in \mathbb{Z}\ \forall i \in I\}, \tag{1b}$$

where $\underline{x},\bar{x} \in \bar{\mathbb{R}}^n$ are the *lower and upper bounds* on the variables ($\overline{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$), the matrix $A \in \mathbb{R}^{m' \times n}$ and the vector $b \in \mathbb{R}^{m'}$ specify the *linear constraints*, $I \subseteq \{1,\dots,n\}$ denotes the set of variables with *integrality requirements*, $c \in \mathbb{R}^n$ determines the *objective function*, $g : [\underline{x},\bar{x}] \to \mathbb{R}^m$ are the (nonlinear) *constraint functions*, and $\langle\cdot,\cdot\rangle$ denotes the scalar product of two vectors. Here and in the following, we denote by $[\underline{x},\bar{x}] := \{x \in \mathbb{R}^n : \underline{x}_i \leq x_i \leq \bar{x}_i, i = 1,\dots,n\}$ the *box* for the variables and

*GAMS Software GmbH, c/o Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, svigerske@gams.com.

†Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin, Germany, gleixner@zib.de.

by $x_J := (x_j)_{j \in J}$ the subvector of x for some index set $J \subseteq \{1, \ldots, n\}$. The set X is called *feasible set* of (1). The restriction to a linear objective function and inequality constraints is only for notational simplicity. A problem of the form (1) without nonlinear constraints ($m = 0$) is called a *mixed-integer linear program* (MIP). If only integrality requirements are absent ($I = \emptyset$), (1) is called a *nonlinear program* (NLP). The case of only linear constraints and continuous variables ($m = 0$, $I = \emptyset$) is denoted as *linear program* (LP).

The combination of discrete decisions, nonlinearity, and possible nonconvexity of the nonlinear functions in MINLP combines the areas of mixed-integer linear programming, nonlinear programming, and global optimization into a single problem class. While linear and convex smooth nonlinear programs are solvable in polynomial time in theory [56, 86] and very efficiently in practice [28, 71], nonconvexities as imposed by discrete variables or nonconvex nonlinear functions easily lead to problems that are NP-hard in theory and computationally demanding in practice.

The most common method to solve nonconvex MINLPs to ε-global optimality is spatial branch-and-bound [55, 58, 63]: recursively divide the original problem into subproblems on smaller domains until the individual subproblems are easy to solve. Bounding is used to decide early whether improving solutions can be found in a subtree. These bounds are computed from a convex relaxation of the problem, which is obtained by dropping the integrality requirements and relaxing nonlinear constraints by a convex or even polyhedral outer approximation. Branching, i.e., the division into subproblems, is typically performed on discrete variables that take a fractional value in the relaxation solution and on variables that are involved in nonconvex terms of violated nonlinear constraints. The restricted domains allow for tighter relaxations in the generated subproblems. As for spatial branch-and-bound in general, also SCIP requires finite bounds on the variables that are involved in nonconvex nonlinear constraints in order to guarantee ε-global optimality.

Over the last decades, substantial progress has been made in the solvability of both mixed-integer linear programs [29, 57] and nonconvex nonlinear programs [40, 49, 61]. Since its beginning in the mid 1970's [10, 41], also the integration of MIP and global optimization of NLPs and the development of new algorithms unique to MINLP has made remarkable progress [14, 30, 59]. Even though not competitive with MIP, yet, today there exists a variety of general purpose software packages for the solution of medium-size (nonconvex) MINLPs [31]. One of the first of this kind and still actively maintained and improved is BARON [85], which implements a branch-and-bound algorithm employing LP relaxations. Later, Lindo, Inc., added global solving capabilities to their Lindo API solver suite [44]. An open-source implementation of a global optimization solver is available with COUENNE [11]. A branch-and-bound algorithm based on a *mixed-integer* linear relaxation is implemented in the solver ANTIGONE [66]. The extension of the constraint integer programming framework SCIP by capabilities to handle nonlinear constraints is the topic of this paper. Thereby, we focus on extensions that have been available with SCIP 3.1 (released in 2014).

The article is organized as follows. In Section 2, we describe the design and algorithmic features of SCIP concerning MINLPs. In Section 3, we measure the impact of the individual components of SCIP on its overall computational performance using the public benchmark set MINLPLib2. To this end, we compare SCIP's performance with default settings to its performance with one feature disabled or switched to a different strategy. Section 4 contains our conclusions.

2 A Constraint Integer Programming Approach to MINLP

The paradigm of *constraint integer programming (CIP)* [1, 5] combines modeling and solving techniques from the fields of *constraint programming (CP)*, mixed-integer linear programming, and *satisfiability testing (SAT)*. Several authors have shown that this integration can help to solve optimization problems that are intractable with any of the constituting techniques alone. For an overview see [53]. The strength of CP is the strong modeling potential. While a MIP formulation of a constraint may require a large set of linear constraints and additional variables, in CP very expressive constraints that contain a lot of structure can be used. The latter can often be exploited directly by the domain propagation routines. The concept of constraint integer programming aims at

restricting the generality of CP modeling as little as needed while still retaining the full performance of MIP and SAT solving techniques.

Definition 1 (Constraint Integer Program)**.** A *constraint integer program* (CIP) is a tuple $(\mathscr{C}, I, c)$ that encodes the task of solving

$$\min\{\langle c,x\rangle : \mathscr{C}(x) = \mathbf{1},\ x \in \mathbb{R}^n,\ x_I \in \mathbb{Z}^{|I|}\},$$

where $c \in \mathbb{R}^n$ is the objective function vector, $\mathscr{C} : \mathbb{R}^n \to \{0,1\}^m$ specifies the constraints, $\mathbf{1} = \{1\}^m$, and $I \subseteq \{1,\ldots,n\}$ specifies the set of variables that have to take integral values. Furthermore, a CIP has to fulfill the condition that for each fixed integer assignment $\hat{x}_I \in \mathbb{Z}^{|I|}$, the remaining feasible set must be polyhedral, i.e., there exist $A' \in \mathbb{R}^{k\times|C|}$ and $b' \in \mathbb{R}^k$ such that

$$\{x_C \in \mathbb{R}^{|C|} : \mathscr{C}(x_C, \hat{x}_I) = \mathbf{1}\} = \{y \in \mathbb{R}^{|C|} : A'y \leq b'\},$$

where $C := \{1,\ldots,n\} \setminus I$ and $k \in \mathbb{N}$.

Thus, if the discrete variables are bounded, a CIP can be solved, in principle, by enumerating all values of the integral variables and solving the corresponding LPs. In Proposition 1.7 of [1] it is shown that CIP includes MIP and CP over finite domains as special cases.

Note, that while MINLP is a special case of CP, it is in general not a special case of CIP, since the nonlinear constraint $g(x) \leq 0$ may forbid a linear representation of the MINLP after fixing the integer variables. However, the main purpose of this requirement is to ensure that the problem that remains after fixing all integer variables in the CIP is efficiently solvable. Thus, for practical applications, an algorithm that can solve the remaining NLP up to a given precision within finite time is sufficient.

The idea of CIP has been implemented in the branch-cut-and-price framework SCIP [1, 3], which also implements state-of-the-art techniques for solving MIPs. Due to its plugin-based design, it can be easily customized and extended, e.g., by adding problem specific separation, presolving, domain propagation, or branching algorithms. In [24] it has been shown how quadratic constraints can be incorporated into a CIP framework. In this section, we discuss an extension of this work to the handling of general nonlinear constraints.

Figure 1 shows the main solving loop of SCIP, into which the handling of nonlinear constraints has been integrated seamlessly. During presolving, SCIP reformulates factorable nonlinear constraints into elementary nonlinearities by introducing auxiliary variables. It tries to detect special mathematical structures and convexity and applies bound tightening procedures. The bounding step solves an LP relaxation, which is updated repeatedly using local bounds of the variables. This cut loop is iterated with bound tightening as long as there is sufficient progress in tightening the LP relaxation. Branching is first performed on integer variables with fractional value in the LP solution, subsequently on variables contained in violated nonconvex constraints. Primal heuristics that attempt to find high-quality solutions early during the search are applied at various points during the solution process. If an integer feasible solution candidate is available, the integer variables are fixed and the remaining NLP on the space of continuous variables is solved to local optimality.

The construction of the LP relaxation is explained in more detail in Section 2.4. To summarize, SCIP uses convex envelopes for well-known univariate functions, linearization cuts for convex constraints, and the classical McCormick relaxation of bilinear terms. All of these are dynamically separated, for pure NLPs also at a solution of the NLP relaxation.

The strength of SCIP lies in the deep integration of these nonlinear components into the main solving loop and hence with the state-of-the-art techniques available for constraint integer programs in general and for MIPs in particular. Examples are cutting planes, which tighten the MIP relaxation, or conflict analysis, which tries to obtain short certificates of infeasibility from infeasible nodes and propagates them to reduce the size of the search tree. This benefits the performance on problems where the discrete part poses the main challenge. Furthermore, it enables the user to combine classical models for MINLPs with more general types of constraints found in constraint programming, see [23, 25] for examples. In the following, we describe the nonlinear extensions of SCIP summarized above in more detail.

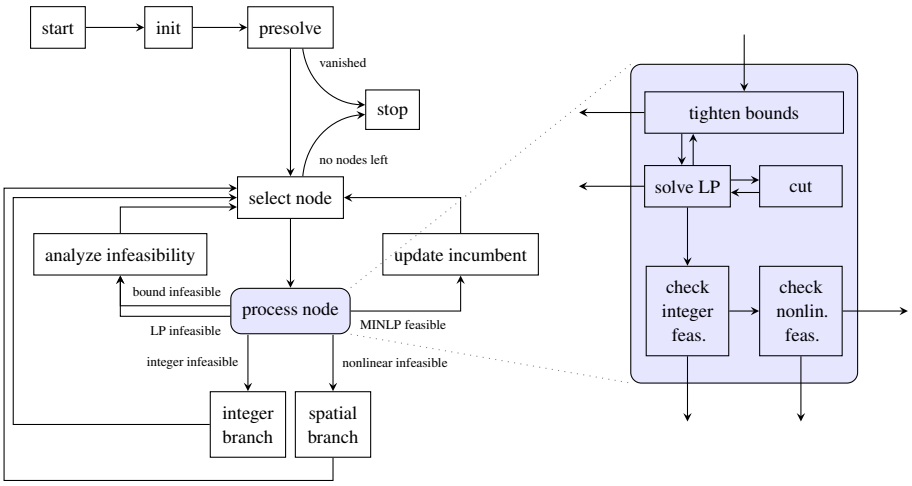

Figure 1: Flowchart of the main MINLP solving loop of SCIP.

2.1 The Expression Graph

The most general form of MINLPs handled by SCIP are those where each nonlinear function $g_j(x)$ can be expressed by the root of an *expression tree*, where leaf nodes correspond to variables and non-leaf nodes to algebraic operations on the children given by either a signomial function $y \mapsto \sum_{j=1}^{k} \alpha_j y_1^{\beta_{j,1}} \cdots y_m^{\beta_{j,m}}$ or any of the univariate functions $y \mapsto \text{sign}(y)|y|^a$, $a > 1$, $y \mapsto \exp(y)$, $y \mapsto \log(y)$, and $y \mapsto |y|$. For expression trees, SCIP can compute first and second derivatives of the associated function by use of the algorithmic differentiation code CppAD[1].

The most general nonlinear constraints in SCIP have the form

$$\ell \leq \langle a, x \rangle + h(x) \leq u \tag{2}$$

with $\ell, u \in \bar{\mathbb{R}}$, $a \in \mathbb{R}^n$, and $h : [\underline{x}, \overline{x}] \to \mathbb{R}$. Functions $h(x)$ that occur in nonlinear constraints are stored in a single so-called *expression graph*, which is a directed acyclic graph that has variables and constants as sources and the functions $h(x)$ as sinks, see also [82, 88]. An example expression graph is depicted in Figure 2. When an expression tree, specifying a single nonlinear function, is added to an existing expression graph, SCIP tries to recognize subexpressions that are already stored in the graph and, thus, avoids adding new nodes to the graph for such subexpressions. During presolve, SCIP flattens the expression graph by applying some simple transformations, e.g., replacing nodes which children are all constants by a corresponding constant and flattening nested sums and products, see also Section 7.3.1 in [87] for details.

2.2 Bound Tightening

The tightness of the relaxation for nonconvex nonlinear constraints depends on the tightness of the bounds of the variables involved in these constraints. Hence bound tightening, also known as propagation, forms a crucial component of nonconvex MINLP solvers. SCIP's standard propagation engine includes, e.g., reduced cost tightening [80] and probing on binary variables [81]. Specifically for MINLP, SCIP implements the constraint-based and optimization-based reduction procedures explained below.

2.2.1 Constraint-Based Bound Tightening

A constraint-based bound tightening method, also known as feasibility-based bound tightening (FBBT, see [12] for a formal definition), is available in SCIP. For a linear constraint, $\ell \leq \langle a, x \rangle \leq u$,

[1] `http://www.coin-or.org/CppAD`

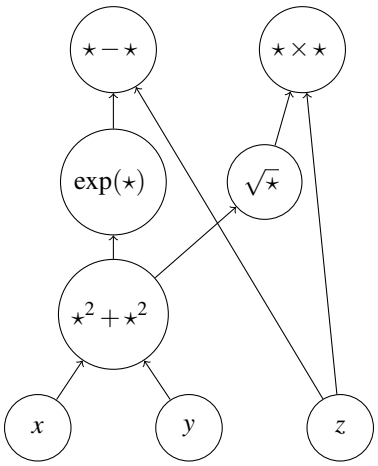

Figure 2: Expression graph for the two expressions $\exp(x^2+y^2)-z$ and $z\sqrt{x^2+y^2}$.

the method computes possibly tighter bounds on each variable from the constraint sides $[l,u]$ and existing variable bounds $[\underline{x},\overline{x}]$ by means of the formulas

$$\frac{1}{a_i}\left(\ell - \sum_{j\neq i} \max\{a_i\underline{x}_i, a_i\overline{x}_i\}\right) \leq x_i \leq \frac{1}{a_i}\left(u - \sum_{j\neq i} \min\{a_i\underline{x}_i, a_i\overline{x}_i\}\right), \qquad \text{if } a_i > 0,$$

$$\frac{1}{a_i}\left(u - \sum_{j\neq i} \min\{a_i\underline{x}_i, a_i\overline{x}_i\}\right) \leq x_i \leq \frac{1}{a_i}\left(\ell - \sum_{j\neq i} \max\{a_i\underline{x}_i, a_i\overline{x}_i\}\right), \qquad \text{if } a_i < 0,$$

see also Section 7.1 in [1] for more details.

For nonlinear constraints, a constraint-based bound tightening method that is based on a rounding-safe extended *interval arithmetic*, an extension of arithmetic on $\mathbb{R}$ to the set of intervals on $\mathbb{R}$ [68, 69], has been implemented in SCIP. Given bounds on the variables, intervals for the linear term $\langle a,x\rangle$ in (2) and all nodes in the expression graph can be computed. Bounds on a function $h(x)$ in a nonlinear constraint (2) are then obtained from the interval of the corresponding sink in the expression graph.

Using the bounds $[\ell,u]$ on $\langle a,x\rangle + h(x)$ from (2), possible tighter bounds on the variables involved in both $\langle a,x\rangle$ and $h(x)$ are computed by applying the interval evaluation in a backward manner, thereby propagating intervals from the sinks of an expression graph to the sources. Figure 3 shows one iteration of propagating intervals on the expression graph from Figure 2.

Alternating propagation of variable bounds and constraint sides may be repeated as long as bound tightenings are found. However, to avoid an endless loop with very small improvements in the bounds, interval tightenings are only propagated if the relative amount of tightening is above a certain threshold, see also Section 7.1 in [1].

Univariate quadratic functions For univariate and bivariate quadratic functions, special interval arithmetic routines that ensure tightest possible bounds are used. The univariate case has been discussed in [36]. In summary, bounds on a quadratic form ax^2+bx, $a\neq 0$, are given by

$$\{ax^2+bx : x\in[\underline{x},\overline{x}]\} = \begin{cases} \operatorname{conv}\{a\underline{x}^2+b\underline{x}, a\overline{x}^2+b\overline{x}, -\frac{b^2}{4a}\}, & \text{if } -\frac{b}{2a}\in[\underline{x},\overline{x}], \\ \operatorname{conv}\{a\underline{x}^2+b\underline{x}, a\overline{x}^2+b\overline{x}\}, & \text{otherwise,} \end{cases} \tag{3}$$

where $\operatorname{conv} S$ denotes the convex hull of a set S. To find the smallest interval containing all solutions of $ax^2+bx\geq c$, $a\neq 0$, we rewrite this inequality as

$$a\left(x+\frac{b}{2a}\right)^2 \geq c+\frac{b^2}{4a}$$

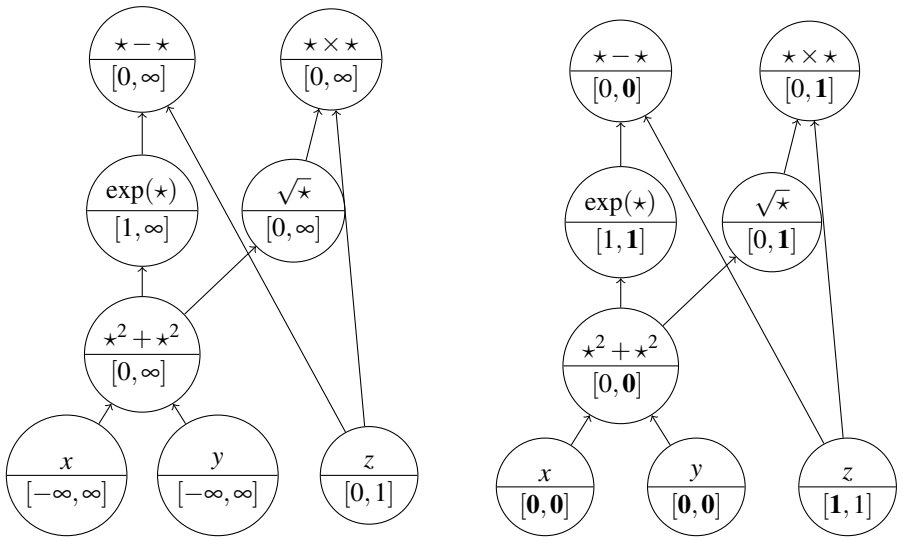

Figure 3: One round of constraint-based bound tightening on expression graph for the two constraints $\exp(x^2+y^2)-z \le 0$ and $z\sqrt{x^2+y^2} \le 1$ with $x \in \mathbb{R}$, $y \in \mathbb{R}$, $z \in [0,1]$.
Left: The bounds on the variable are propagated upwards through the graph, yielding $x^2+y^2 \in [0,\infty]$, $\exp(x^2+y^2) \in [1,\infty]$, $\sqrt{x^2+y^2} \in [0,1]$, $\exp(x^2+y^2)-z \in [0,\infty]$, and $z\sqrt{x^2+y^2} \in [0,\infty]$.
Right: The constraints right-hand-sides are enforced at the top nodes and intervals are propagated downwards through the graph (tightened interval ends are depicted in bold font), yielding $\exp(x^2+y^2) \in [0,0]+[0,1] = [0,1]$, $z \in [1,1]-[0,0] = [1,1]$, $x^2+y^2 \in \ln([1,1]) = [0,0]$, $x = \{x \mid \exists y : x^2+y^2 \in [0,0]\} = [0,0]$, $y = \{y \mid \exists x : x^2+y^2 \in [0,0]\} = [0,0]$, and $\sqrt{x^2+y^2} \in \frac{[0,1]}{[1,1]} = [0,1]$.

If $c+\frac{b^2}{4a} \le 0$ and $a > 0$, then obviously $[\underline{x},\overline{x}] = \mathbb{R}$. If $c+\frac{b^2}{4a} > 0$ and $a < 0$, then $[\underline{x},\overline{x}] = \emptyset$. Otherwise, i.e., $\frac{1}{a}(c+\frac{b^2}{4a}) \ge 0$, we obtain

$$x \in \left[-\infty, -\sqrt{\frac{c}{a}+\frac{b^2}{4a^2}} - \frac{b}{2a}\right] \cup \left[\sqrt{\frac{c}{a}+\frac{b^2}{4a^2}} - \frac{b}{2a}, \infty\right] \qquad \text{for } a > 0, \tag{4a}$$

$$x \in \left[-\sqrt{\frac{c}{a}+\frac{b^2}{4a^2}} - \frac{b}{2a}, \sqrt{\frac{c}{a}+\frac{b^2}{4a^2}} - \frac{b}{2a}\right] \qquad \text{for } a < 0. \tag{4b}$$

Bivariate quadratic functions Next, consider a bivariate quadratic form

$$q(x,y) := a_x x^2 + a_y y^2 + a_{xy} xy + b_x x + b_y y \tag{5}$$

with $a_{xy} \neq 0$. To find bounds on $q(x,y)$ for $(x,y) \in [\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$, we compute the minima and maxima of $q(x,y)$ in the interior and on the boundary of $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$. The (unrestricted) minima/maxima of $q(x,y)$ with $4a_x a_y \neq a_{xy}^2$ are

$$\hat{x} = \frac{a_{xy}b_y - 2a_y b_x}{4a_x a_y - a_{xy}^2}, \quad \hat{y} = \frac{a_{xy}b_x - 2a_x b_y}{4a_x a_y - a_{xy}^2} \quad \Rightarrow \quad q(\hat{x},\hat{y}) = \frac{a_{xy}b_x b_y - a_y b_x^2 - a_x b_y^2}{4a_x a_y - a_{xy}^2}.$$

If $4a_x a_y = a_{xy}^2$ and $2a_y b_x = a_{xy} b_y$, then

$$\hat{x} \in \mathbb{R}, \quad \hat{y} = -\frac{b_x}{a_{xy}} - \frac{a_{xy}}{2a_y}\hat{x} \quad \Rightarrow \quad q(\hat{x},\hat{y}) = -\frac{a_y b_x^2}{a_{xy}^2}.$$

Otherwise ($4a_x a_y = a_{xy}^2$ and $2a_y b_x \neq a_{xy} b_y$), there is no unrestricted minimum/maximum. For x or y fixed to $\underline{x}$, $\overline{x}$, or $\underline{y}$, $\overline{y}$, respectively, bounds on $q(x,y)$ can be derived from (3). Bounds on $q(x,y)$

are then obtained by comparing the bounds at the boundary of $[\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$ with the value at $(\hat{x},\hat{y})$, if $(\hat{x},\hat{y}) \in [\underline{x},\overline{x}] \times [\underline{y},\overline{y}]$. The latter need to be computed only if $4a_x a_y \neq a_{xy}^2$, since otherwise the unrestricted minimum/maximum, if any, is also attained for $x \in \{\underline{x},\overline{x}\}$.

Finally, we aim to compute the interval

$$\operatorname{conv}\{x \in [\underline{x},\overline{x}] : q(x,y) \in [\underline{c},\overline{c}] \text{ for some } y \in [\underline{y},\overline{y}]\} \tag{6}$$

for given $[\underline{x},\overline{x}]$, $[\underline{y},\overline{y}]$, and $[\underline{c},\overline{c}]$ (analogously for bounds on y). First, consider the case $a_x \neq 0$. Thus, w.l.o.g., assume $a_x > 0$. We can rewrite the equation $q(x,y) \in [\underline{c},\overline{c}]$ as

$$\left(\sqrt{a_x}x + b(y)\right)^2 \in r([\underline{c},\overline{c}],y), \tag{7}$$

where

$$b(y) := \frac{b_x + a_{xy}y}{2\sqrt{a_x}} \qquad \text{and} \qquad r(c,y) := c - a_y y^2 - b_y y + b(y)^2.$$

Using (3), we can compute the interval $r([\underline{c},\overline{c}],[\underline{y},\overline{y}])$. If $r([\underline{c},\overline{c}],[\underline{y},\overline{y}]) \cap \mathbb{R}_+ = \emptyset$, then (6) is empty. Otherwise, the set of x that satisfies (7) is

$$\left\{x : \exists y \in [\underline{y},\overline{y}] : \sqrt{a_x}x + b(y) \in -\sqrt{r([\underline{c},\overline{c}],y)} \cup \sqrt{r([\underline{c},\overline{c}],y)}\right\}. \tag{8}$$

It thus remains to compute minimal and maximal values for $\pm\sqrt{r([\underline{c},\overline{c}],y)} - b(y)$. Using a lengthy case distinction, these values can be computed analytically, see [87, Section 7.4.2] for details.

In the case $a_x = 0$, (6) reduces to finding $x \in [\underline{x},\overline{x}]$ such that there exists an $y \in [\underline{y},\overline{y}]$ with $(b_x + a_{xy}y)x \in [\underline{c},\overline{c}] - a_y y^2 - b_y y$. Thus, an interval that encloses the set

$$\left\{\frac{[\underline{c},\overline{c}] - a_y y^2 - b_y y}{b_x + a_{xy}y} : y \in [\underline{y},\overline{y}]\right\}. \tag{9}$$

has to be found. In the case $-\frac{b_x}{a_{xy}} \in (\underline{y},\overline{y})$, this enclosure is $\mathbb{R}$. If $-\frac{b_x}{a_{xy}} \notin [\underline{y},\overline{y}]$, then it is sufficient to find the minima and maxima of $\frac{c - a_y y^2 - b_y y}{b_x + a_{xy}y}$ for $c \in \{\underline{c},\overline{c}\}$ and $y \in [\underline{y},\overline{y}]$. This can be done analytically, see again [87, Section 7.4.2] for details. Finally, in the case $-\frac{b_x}{a_{xy}} \in \{\underline{y},\overline{y}\}$, the SCIP implementation assumes that no bound tightening for x is possible.

Example 1. Consider the quadratic inequality

$$2x^2 - y^2 + xy - y \leq 0 \tag{10}$$

and bounds $x \in [0,1]$ and $y \in [-1,1]$, see also Figure 4. We are interested in deriving a tighter upper bound for x.

First, let us relax (10) into a univariate form by rewriting as

$$2x^2 + [-1,1]x \leq \max\{y^2 + y : y \in [-1,1]\}.$$

Applying (3) to the right-hand side and (4) to the left-hand side (considering the two boundary cases separately), we obtain the bounds $[1/4(1-\sqrt{17}), 1/4(1+\sqrt{17})] \approx [-0.780776, 1.28078]$ for x, which do not improve on the existing ones.

Next, we write (10) in the form of (7), which gives $(\sqrt{2}x + b(y))^2 \in r(\mathbb{R}_-,y)$ with $r(c,y) = c + y^2 + y + b(y)^2$ and $b(y) = \frac{y}{2\sqrt{2}}$. Due to (8), an upper bound on $\sqrt{2}x$ is

$$\max\{\sqrt{r(0,y)} - b(y) : y \in [-1,1]\} = \max\left\{\sqrt{y + \frac{9y^2}{8}} - \frac{y}{2\sqrt{2}} : y \in [-1,1]\right\} = \frac{\sqrt{17}-1}{2\sqrt{2}},$$

which yields $(\sqrt{17}-1)/4 \approx 0.780776$ as new (and tight) upper bound for x.

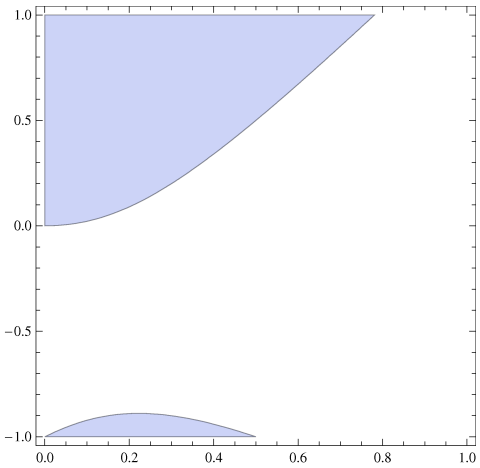

Figure 4: Feasible region of inequality (10).

2.2.2 Optimization-Based Bound Tightening

Compared to the propagation techniques above, OBBT – short for *optimization-based bound tightening* – is a rather expensive procedure that minimizes and maximizes each variable over the feasible set of the problem or a relaxation thereof. In SCIP, the current LP relaxation is used. Whereas FBBT propagates the nonlinearities individually, OBBT considers (the relaxations of) all constraints together, and may hence compute tighter bounds. See [33, 62, 77, 78, 83, 91] for appearances of OBBT in the literature.

OBBT solves for each variable x_k the two auxiliary LPs

$$\min / \max \{x_k : Dx \leq d, \langle c, x \rangle \leq U, x \in [\underline{x}, \overline{x}]\} \tag{11}$$

where $Dx \leq d$, $D \in \mathbb{R}^{\ell \times n}$, $d \in \mathbb{R}^{\ell}$ is a linear relaxation of the feasible region that can be constructed as described in Section 2.4, and $\langle c, x \rangle \leq U$ is an objective cutoff constraint that excludes solutions with objective value worse than the current incumbent. The optimal value of (11) may then be used to tighten the lower / upper bound of variable x_k.

OBBT is a standard feature of many MINLP solvers [6, 7, 11, 65, 73]. SCIP, by default, applies OBBT at the root node to tighten bounds globally. It restricts the computational effort by limiting the amount of LP iterations spent for solving the auxiliary LPs and interrupting for cheaper propagation techniques to be called between LP solves. As a special feature, SCIP does not only use the optimal objective values of (11) to tighten the bounds on x_k, but it also exploits the dual solution in order to learn valid inequalities useful for propagation during the subsequent search.

Suppose the maximization LP is solved and feasible dual multipliers $\lambda_1, \ldots, \lambda_\ell, \mu \geq 0$ for $Dx \leq d$, $\langle c, x \rangle \leq U$ and the corresponding reduced cost vector r are obtained. Then

$$x_k \leq \sum_j r_j x_j + \langle \lambda, d \rangle + \mu U \tag{12}$$

is a valid inequality, which we call *Lagrangian variable bound* (LVB), and

$$\sum_{j: r_j < 0} r_j \underline{x}_j + \sum_{j: r_j > 0} r_j \overline{x}_j + \langle \lambda, d \rangle + \mu U \tag{13}$$

is a valid upper bound for x_k that equals the OBBT bound if the dual multipliers are optimal. Figure 5 gives an illustrative example.

SCIP learns LVBs at the root node and propagates them at local nodes of the search tree whenever the bounds of variables on the right-hand side of (12) become tighter and globally whenever an improved primal solution is found. This gives a computationally cheap approximation of OBBT during the branch-and-bound search and can notably reduce solution times and number of branch-and-bound nodes. For further details, see [47, 48].

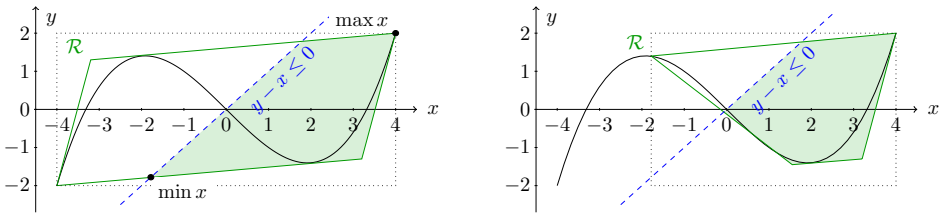

Figure 5: Example $\min\{y - x : y = 0.1x^3 - 1.1x, x \in [-4,4], y \in [-2,2]\}$. On the left, the shaded region over which OBBT is performed is defined by the LP relaxation $\mathscr{R}$ and the dashed objective cutoff resulting from the zero solution. Minimizing x gives a lower bound of $-\frac{16}{9}$ and the LVB $x \geq -\frac{10}{9}U - \frac{16}{9}$. Maximizing x does not tighten its upper bound, still the LVB $x \leq \frac{10}{37}y + \frac{128}{37}$ can be learnt. In this example, this is merely a facet of $\mathscr{R}$, but in higher dimensions it may be nontrivial. On the right is the resulting, tighter relaxation.

2.3 Convexity Detection

Recognizing convexity of a function $h(x)$ in (2) (w.r.t. $x \in [\underline{x}, \overline{x}]$) is advantageous, as it considerably reduces the effort to construct a linear relaxation of (2). Further, if a function $h(x)$ is concave, its convex envelope is a polyhedral function given by the values of $h(x)$ in all vertices of $[\underline{x}, \overline{x}]$ [37].

Existing deterministic methods for proving or disproving the convexity of a function (given as composition of elementary expressions) with respect to bounds on its variables are based on walking an expression tree and applying convexity rules for function compositions [9, 42] or estimating the spectra of the Hessian matrix [67, 70]. Both methods may give inconclusive results, i.e., check only sufficient criteria for convexity. Nevertheless, SCIP implements the former method to recognize "evidently" convex/concave expressions in an expression graph.

Consider a composition $f(g(x))$ of twice-differentiable functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$. Then

$$(f \circ g)''(x) = f''(g(x))\nabla g(x)(\nabla g(x))^\top + f'(g(x))\nabla^2 g(x). \tag{14}$$

Thus, for convexity of $f(g(x))$, it is sufficient that $f(x)$ is convex and monotonically increasing on $[\underline{g}, \overline{g}]$ and $g(x)$ is convex on $[\underline{x}, \overline{x}]$, where we denote by $[\underline{g}, \overline{g}]$ an interval that contains $\{g(x) : x \in [\underline{x}, \overline{x}]\}$. Analogously, if $f(x)$ is concave and monotonically increasing on $[\underline{g}, \overline{g}]$ and $g(x)$ is concave on $[\underline{g}, \overline{g}]$, then $f(g(x))$ is concave on $[\underline{x}, \overline{x}]$. Similar conclusions can be drawn if $f(x)$ is monotonically decreasing. These observations allow to propagate convexity and concavity properties through the expression graph, see [87, Table 7.2] for a summary of the applied rules.

Additionally to the convexity detection for expressions in an expression graph, SCIP detects convexity of quadratic constraints $\langle x, Qx \rangle + \langle q, x \rangle + \bar{q} \leq 0$ by checking whether the minimal eigenvalue of the matrix Q is nonnegative.

2.4 Linear Relaxation

SCIP constructs and strengthens the LP relaxation by adding valid linear inequalities during the cut loop, see Figure 1. In the initial separation rounds, cuts derived from nonlinear constraints are added only if they violate the relaxations solution by a sufficient amount (10^{-4} is the default tolerance). Before branching, when the LP relaxation solution is integer feasible, also weaker cuts can be added as long as they cut off the LP solution w.r.t. the feasibility tolerance (10^{-6} in default settings). In the following, we discuss the cut generation routines for the various types of nonlinear functions that are implemented in SCIP. Let $\tilde{x}$ be a solution of the LP relaxation that we aim to separate.

2.4.1 General Nonlinear Functions

Assume that the right-hand-side of a constraint (2) is violated by $\tilde{x}$. Further, assume that function $h(x)$ in (2) is represented as

$$h(x) = \sum_{j=1}^{k} h_j(x_{J_j}), \tag{15}$$

where $J_j \subseteq \{1,\dots,n\}$, $j = 1,\dots,k$. Due to the convexity check from Section 2.3, SCIP may know that some $h_j : \mathbb{R}^{|J_j|} \to \mathbb{R}$ are convex or concave on $[\underline{x},\overline{x}]$. SCIP tries to compute a cut by adding up linear underestimators for each $h_j(\cdot)$. If this fails, the constraint needs to be enforced by other means (bound tightening, spatial branching).

Convex $h_j(\cdot)$ If $h_j(\cdot)$ is convex, a linear underestimator is obtained by linearization of $h_j(\cdot)$ at $\tilde{x}_{J_j}$, i.e.,

$$h_j(x_{J_j}) \geq h_j(\tilde{x}_{J_j}) + \langle \nabla h_j(\tilde{x}_{J_j}), x_{J_j} - \tilde{x}_{J_j} \rangle. \tag{16}$$

If $h_j(\cdot)$ or $\nabla h_j(\cdot)$ cannot be evaluated at $\tilde{x}_{J_j}$, (16) is retried with a slightly perturbed $\tilde{x}_{J_j}$.

Concave $h_j(\cdot)$ If $h_j(\cdot)$ is concave, its convex envelope w.r.t. $[\underline{x}_{J_j},\overline{x}_{J_j}]$ (if bounded) is given by

$$\inf \left\{ \sum_{i=1}^{2^{|J_j|}} \lambda_i h(x^i) : \sum_{i=1}^{2^{|J_j|}} \lambda_i x^i = x_{J_j},\ \sum_{i=1}^{2^{|J_j|}} \lambda_i = 1,\ \lambda \geq 0 \right\}, \tag{17}$$

where $\{x^i\}_{i=1,\dots,2^{|J_j|}}$ are the extreme points of $[\underline{x}_{J_j},\overline{x}_{J_j}]$ [37]. By duality, (17) is equivalent to

$$\sup \left\{ \langle \mu, x \rangle + \sigma : \langle \mu, x^i \rangle + \sigma \leq f(x^i),\ i = 1,\dots,2^{|J_j|} \right\}. \tag{18}$$

Thus, solving the linear program (18) for $x = \tilde{x}_{J_j}$ yields a linear function $\langle \mu, x \rangle + \sigma$ that underestimates $h_j(\cdot)$ on $[\underline{x}_{J_j},\overline{x}_{J_j}]$ and that takes the value of the convex envelope at $x = \tilde{x}_{J_j}$. No cut is generated if $[\underline{x},\overline{x}]$ is unbounded or $h_j(\cdot)$ cannot be evaluated for some x^i, $i = 1,\dots,2^{|J_j|}$.

If $h_j(\cdot)$ is also univariate ($|J_j| = 1$), the convex envelope is explicitly given as

$$h_j(x_{J_j}) \geq h_j(\underline{x}_{J_j}) + \frac{h_j(\overline{x}_{J_j}) - h_j(\underline{x}_{J_j})}{\overline{x}_{J_j} - \underline{x}_{J_j}} (x_{J_j} - \underline{x}_{J_j}). \tag{19}$$

Indefinite $h_j(\cdot)$ If $h_j(\cdot)$ is neither convex nor concave, but continuously differentiable, and $[\underline{x},\overline{x}]$ is bounded, then SCIP can compute a linear underestimator of $h_j(\cdot)$ by using interval arithmetic on the gradient of $h_j(\cdot)$.

Note, that by Taylor's theorem,

$$h_j(x_{J_j}) \geq h_j(\tilde{x}_{J_j}) + \min_{y \in [\underline{x}_{J_j},\overline{x}_{J_j}]} \langle \nabla h_j(y), x_{J_j} - \tilde{x}_{J_j} \rangle \qquad (x_{J_j} \in [\underline{x}_{J_j},\overline{x}_{J_j}]). \tag{20}$$

Let $[\underline{d},\overline{d}]$ be such that $\nabla h_j(x_{J_j}) \in [\underline{d}_{J_j},\overline{d}_{J_j}]$ for all $x_{J_j} \in [\underline{x}_{J_j},\overline{x}_{J_j}]$. Then (20) yields

$$h_j(x_{J_j}) \geq h_j(\tilde{x}_{J_j}) + \sum_{i \in J_j : x_i \geq \tilde{x}_i} \underline{d}_i (x_i - \tilde{x}_i) + \sum_{i \in J_j : x_i < \tilde{x}_i} \overline{d}_i (x_i - \tilde{x}_i)$$

By moving the reference point $\tilde{x}_{J_j}$ to a closest extreme point of the box $[\underline{x}_{J_j},\overline{x}_{J_j}]$, we can derive a linear underestimator. Thus, let $\hat{x}_{J_j}$ and d_{J_j} be defined by

$$\hat{x}_i = \begin{cases} \underline{x}_i, & \text{if } \tilde{x}_i \leq (\underline{x}_i + \overline{x}_i)/2, \\ \overline{x}_i, & \text{otherwise,} \end{cases} \qquad d_i = \begin{cases} \underline{d}_i, & \text{if } \hat{x}_i = \underline{x}_i, \\ \overline{d}_i, & \text{if } \hat{x}_i = \overline{x}_i, \end{cases} \qquad (i \in J_j). \tag{21}$$

Then

$$h_j(x_{J_j}) \geq h_j(\hat{x}_{J_j}) + \langle d_{J_j}, x_{J_j} - \hat{x}_{J_j} \rangle \tag{22}$$

is a valid underestimator, which is used to derive *interval gradient cuts* [72, Section 7.1.3]. A generalization of (22) by using interval slopes instead of interval gradients is discussed by [45, 82].

If $[\underline{x}_{J_j}, \overline{x}_{J_j}]$ or $[\underline{d}_{J_j}, \overline{d}_{J_j}]$ is unbounded and this results in infinite values in $\hat{x}_{J_j}$ or d_{J_j}, then no cut is generated. The box $[\underline{d}_{J_j}, \overline{d}_{J_j}]$ is computed in SCIP by calling the automatic differentiation methods for the computation of gradients in CPPAD with the base data type changed from usual floating-point numbers to intervals.

Note, that the underestimator (22) can be very weak, even though it usually improves when the box $[\underline{x}_{J_j}, \overline{x}_{J_j}]$ shrinks (by bound tightening or branching). Thus, SCIP avoids general indefinite functions $h_j(\cdot)$ by reformulating the expression graph (see Section 2.5.1) during presolve.

2.4.2 Odd and Signed Power Functions

Consider the constraint

$$\ell \leq \operatorname{sign}(x+b)|x+b|^a + cz \leq u, \tag{23}$$

where $a > 1$, $b, c \in \mathbb{R}$, $\ell, u \in \bar{\mathbb{R}}$. For $a \in 2\mathbb{Z}+1$ (odd power), (23) is equivalent to $\ell \leq (x+b)^a + cz \leq u$. Assume $\tilde{x}$ violates the right-hand-side of constraint (23).

For the interesting case $\underline{x}+b < 0 < \overline{x}+b$, the convex envelope of $x \mapsto \operatorname{sign}(x+b)|x+b|^a$ on $[\underline{x}, \overline{x}]$ is given by the secant between $(\underline{x}, -|\underline{x}+b|^a)$ and $(x^*, (x^*+b)^a)$ for $x \in [\underline{x}, x^*]$ and by the function itself for $x \geq x^*$, where $x^* > -b$ is such that the slope of the secant coincides with the gradient of the function at x^*, i.e.,

$$\frac{(x^*+b)^a + |\underline{x}+b|^a}{x^* - \underline{x}} = a(x^*+b)^{a-1} \tag{24}$$

As in [60, 87], it can be shown that (24) has exactly one solution, which can easily be found numerically via Newton's method.

If $\underline{x} \neq -\infty$, a linear underestimator of $x \mapsto \operatorname{sign}(x+b)|x+b|^a$ is obtained by linearization of the convex envelope in $\tilde{x}$, which yields the cut

$$\begin{aligned} -|\underline{x}+b|^a + \frac{(x^*+b)^a + |\underline{x}+b|^a}{x^* - \underline{x}}(x - \underline{x}) + cz \leq u \qquad & \text{for} \quad \tilde{x} \leq x^*, \\ (\tilde{x}+b)^a + a(\tilde{x}+b)^{a-1}(x - \tilde{x}) + cz \leq u \qquad & \text{for} \quad \tilde{x} > x^*. \end{aligned}$$

If the left-hand-side of (23) is violated, then a similar method is used to derive a cut.

2.4.3 Quadratic Functions

Consider a quadratic constraint

$$\langle x, Qx \rangle + \langle q, x \rangle + \bar{q} \leq 0 \tag{25}$$

with $Q \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$, and $\bar{q} \in \mathbb{R}$.

Convex quadratic functions If the quadratic function in (25) is convex ($Q \succeq 0$), SCIP generates the cut

$$\langle \tilde{x}, Q\tilde{x} \rangle + \langle 2Q\tilde{x}, x - \tilde{x} \rangle + \langle q, x \rangle + \bar{q} \leq 0, \tag{26}$$

which is obtained by linearization in $\tilde{x}$. In the special case that $\langle x, Qx \rangle \equiv ax_i^2$ for some $a > 0$ and $i \in I$ with $\tilde{x}_i \notin \mathbb{Z}$, SCIP generates the cut

$$\bar{q} + \langle q, x \rangle + a(2\lfloor \tilde{x}_i \rfloor + 1)x_i - a\lfloor \tilde{x}_i \rfloor \lceil \tilde{x}_i \rceil \leq 0, \tag{27}$$

which is obtained by underestimating $x_i \in \mathbb{Z} \mapsto x_i^2$ by the secant defined by the points $(\lfloor \tilde{x}_i \rfloor, \lfloor \tilde{x}_i \rfloor^2)$ and $(\lceil \tilde{x}_i \rceil, \lceil \tilde{x}_i \rceil^2)$. Note, that the violation of (27) by $\tilde{x}$ is larger than that of (26).

Nonconvex quadratic functions For a violated nonconvex constraint, SCIP underestimates each term of $\langle x, Qx\rangle$ separately, if none of the special structures discussed below are recognized. A convex term ax_i^2 with $a > 0$, $i \in \{1,\dots,n\}$, is underestimated as discussed above. For the concave case $a < 0$, the secant underestimator $a(\underline{x}_i + \bar{x}_i)x_i - a\underline{x}_i\bar{x}_i$ is used (cf. (19)), if both $\underline{x}_i$ and $\bar{x}_i$ are finite. Otherwise, if $\underline{x}_i = -\infty$ or $\bar{x}_i = \infty$, SCIP does not generate a cut. For a bilinear term ax_ix_j with $a > 0$, McCormick underestimators [63] are utilized,

$$ax_ix_j \geq a\underline{x}_ix_j + a\underline{x}_jx_i - a\underline{x}_i\underline{x}_j,$$
$$ax_ix_j \geq a\bar{x}_ix_j + a\bar{x}_jx_i - a\bar{x}_i\bar{x}_j.$$

If $(\bar{x}_i - \underline{x}_i)\tilde{x}_j + (\bar{x}_j - \underline{x}_j)\tilde{x}_i \leq \bar{x}_i\bar{x}_j - \underline{x}_i\underline{x}_j$ and the bounds $\underline{x}_i$ and $\underline{x}_j$ are finite, the former is used for cut generation, otherwise the latter is used. If both $\underline{x}_i$ or $\underline{x}_j$ and $\bar{x}_i$ or $\bar{x}_j$ are infinite, SCIP does not generate a cut. Similar, for a bilinear term ax_ix_j with $a < 0$, the McCormick underestimators are

$$ax_ix_j \geq a\bar{x}_ix_j + a\underline{x}_jx_i - a\bar{x}_i\underline{x}_j,$$
$$ax_ix_j \geq a\underline{x}_ix_j + a\bar{x}_jx_j - a\underline{x}_i\bar{x}_j.$$

If $(\bar{x}_i - \underline{x}_i)\tilde{x}_j - (\bar{x}_j - \underline{x}_j)\tilde{x}_i \leq \bar{x}_i\underline{x}_j - \underline{x}_i\bar{x}_j$ and the bounds $\bar{x}_i$ and $\underline{x}_j$ are finite, the former is used for cut generation, otherwise the latter is used. Finally, the underestimators are added up to form one linear cut for the given quadratic constraint. Note that by this, SCIP avoids the introduction of artificial variables for single bilinear and square terms.

Second-order cones Quadratic constraints of the form

$$\gamma + \sum_{i=1}^{n-1} (\alpha_i(x_i + \beta_i))^2 \leq (\alpha_n(x_n + \beta_n))^2, \tag{28}$$

where $\alpha_i, \beta_i \in \mathbb{R}$, $i = 1,\dots,n$, $\gamma \in \mathbb{R}_+$, and $\underline{x}_n \geq -\beta_n$ or $\bar{x}_n \leq -\beta_n$ are recognized as *second-order cone* constraints by SCIP. Assume w.l.o.g. $\alpha_n \geq 0$ and $\underline{x}_n \geq -\beta_n$. Then (28) can be reformulated as

$$\sqrt{\gamma + \sum_{i=1}^{n-1} (\alpha_i(x_i + \beta_i))^2} \leq \alpha_n(x_n + \beta_n). \tag{29}$$

As the left-hand-side of (29) is convex, SCIP can apply (16) to generate a cut.

Factorable quadratic functions SCIP checks, whether a quadratic constraint (25) can be written in a form

$$(\langle a^1, x\rangle + b^1)\,(\langle a^2, x\rangle + b^2) \leq u, \tag{30}$$

where $a^1, a^2 \in \mathbb{R}^n$, $b^1, b^2 \in \mathbb{R}$, and $u \in \mathbb{R} \setminus \{0\}$. If a form (30) exists and $\langle a^1, x\rangle + b^1$ or $\langle a^2, x\rangle + b^2$ is bounded away from 0 on $[\underline{x}, \bar{x}]$, then (30) can be reformulated by dividing by $\langle a^1, x\rangle + b^2$ or $\langle a^2, x\rangle + b^2$. Assume, w.l.o.g., that $\langle a^1, x\rangle + b^1 > 0$ on $[\underline{x}, \bar{x}]$. Then (30) can equivalently be written as

$$\langle a^2, x\rangle + b^2 - \frac{u}{\langle a^1, x\rangle + b^1} \leq 0. \tag{31}$$

Since $y \mapsto \frac{1}{y}$ is convex for $y > 0$, $-\frac{u}{\langle a^1, x\rangle + b^1}$ is convex for $u < 0$ and concave for $u > 0$. Thus, for $u < 0$, a cut that supports the (convex) set defined by (25) is generated by applying (16) to (31).

To find the form (30), Sylvester's law of inertia can be utilized, see [87, Section 7.5.3]. If linear variables exist in constraint (25), then a reformulation into the form (30) does not exist. However, one may find a reformulation into a form

$$(\langle a^1, x\rangle + b^1)\,(\langle a^2, x\rangle + b^2) \leq u + \langle c, x\rangle, \tag{32}$$

where $c \in \mathbb{R}^n$. For $u + \langle c, x\rangle < 0$, similar to (16), a linear underestimator of $\langle a^1, x\rangle + b^1 - \frac{u + \langle c, \hat{x}\rangle}{\langle a^2, x\rangle + b^2}$ can be derived for fixed linear variables ($\langle c, x\rangle$ fixed to $\langle c, \hat{x}\rangle$). Next, [13] has shown how to lift such an underestimator to be valid for all $\hat{x} \in [\underline{x}, \bar{x}]$. SCIP uses COUENNE's implementation of these so-called *lifted tangent inequalities*.

2.5 Reformulation

Even more than in mixed-integer linear programming, the formulation of an MINLP is crucial for its solvability. In SCIP, reformulation techniques that aim at improving the formulation of nonlinear constraints are applied during the presolving phase. They are interleaved with standard MIP preprocessing, MINLP-specific bound tightening, and convexity detection. This is motivated by the fact that some reformulations only become valid after other problem reductions have been performed, e.g., fixing variables or tightening bounds of variables; and vice versa, reformulated constraints may lead to further problem reductions. This interaction of reformulation and reduction is facilitated by the organization of presolving in iterated rounds. In each presolve round, SCIP calls both the methods for constraint-wise reformulations and reductions and more general preprocessing algorithms such as reduced cost tightening or OBBT.

2.5.1 Reformulating the Expression Graph

Nonlinear constraints that are given neither by a sum of convex, concave, power, nor quadratic functions are reformulated by SCIP to obey one of these forms. The aim is to reformulate the MINLP into a form such that the convex envelope is known for all resulting nonlinear constraints and linear underestimators can be generated by stronger methods than the typically weak interval gradients (22).

The reformulation routine inspects the expression graph and replaces nodes that correspond to certain subexpressions by new variables, while adding a new constraint to assign the subexpression to the new variable. That is, for a subexpression $f(g_1(x),\dots,g_m(x))$, the following happens:

- if $f(g(x))$ is known to be convex or concave (cf. Section 2.3), do nothing,
- if the function $y \mapsto f(y)$ is quadratic or convex or concave or a (signed) power function, ensure that all arguments correspond to linear expressions by replacing nonlinear $g_j(x)$ by new auxiliary variables z_j and adding new constraints $z_j = g_j(x)$,
- if $f(g(x)) = \alpha \prod_{j=1}^{m} g_j(x)^{\beta_j}$, $m \geq 2$, add auxiliary variables z_1 and z_2, new constraints $z_1 = \prod_{j=1}^{\lfloor m/2 \rfloor} g_j(x)^{\beta_j}$, $z_2 = \prod_{j=\lfloor m/2 \rfloor+1}^{m} g_j(x)^{\beta_j}$, and replace $f(g(x))$ by $\alpha z_1 z_2$,
- if $f(g(x)) = \sum_{j=1}^{k} \alpha_j g_1(x)^{\beta_{j,1}} \cdots g_{m_j}(x)^{\beta_{j,m_j}}$, $k \geq 2$, add auxiliary variables z_j, new constraints $z_j = g_1(x)^{\beta_{j,1}} \cdots g_{m_j}(x)^{\beta_{j,m_j}}$, $j = 1,\dots,k$, and replace $f(g(x))$ by $\sum_{j=1}^{k} \alpha_j z_j$.

For expressions that are created by these rules, the same reformulation rules are applied.

2.5.2 Quadratic Constraints with Binary Variables

Binary variables in quadratic constraints (25) experience a special treatment by SCIP. Obviously, the square of a binary variable can be replaced by the binary variable itself. Further, if a quadratic term contains a product of a binary variable with a linear term,

$$x_i \sum_{j=1}^{n} Q_{i,j} x_j,$$

where x_i is a binary variable, $Q_{i,i} = 0$, and all x_j with $Q_{i,j} \neq 0$ have finite bounds, then this product is replaced by a new variable $z \in \mathbb{R}$ and the linear constraints

$$\underline{M}_1 x_i \leq z \leq \overline{M}_1 x_i$$

$$\sum_{j=1}^{n} Q_{i,j} x_j - \overline{M}_0 (1 - x_i) \leq z \leq \sum_{j=1}^{n} Q_{i,j} x_j - \underline{M}_0 (1 - x_i),$$

where $[\underline{M}_0, \overline{M}_0]$ and $[\underline{M}_1, \overline{M}_1]$ are bounds on $\sum_{j=1}^{n} Q_{i,j} x_j$ for the cases $x_i = 0$ and $x_i = 1$, respectively[2].

[2]The case $[\underline{M}_0, \overline{M}_0] \neq [\underline{M}_1, \overline{M}_1]$ can occur by taking implications of the form $x_i = a \to (\underline{b} \leq x_j \leq \overline{b})$ into account, $a \in \{0,1\}$. SCIP stores these implications in a central data structure [1, Section 3.3].

2.5.3 Quadratic Complementarity Constraints

A quadratic constraint

$$Q_{i,j}x_ix_j + q_ix_i + q_jx_j + \frac{q_iq_j}{Q_{i,j}^2} = 0 \tag{33}$$

is equivalent to

$$(x_i - a)(x_j - b) = 0 \quad \text{with} \quad a = -\frac{q_j}{Q_{i,j}} \text{ and } b = -\frac{q_i}{Q_{i,j}}. \tag{34}$$

Thus, (33) can also be written as $(x_i = a) \vee (x_j = b)$, which yields the following conjunction of *bound disjunction constraints*:

$$(x_i \leq a \vee x_j \leq b) \wedge (x_i \leq a \vee x_j \geq b) \wedge (x_i \geq a \vee x_j \leq b) \wedge (x_i \geq a \vee x_j \geq b). \tag{35}$$

SCIP replaces a quadratic complementarity constraint (34) by the bound disjunctions (35). Similar reformulations are applied for quadratic constraints that can be written as $(x_i - a)(x_j - b) \geq 0$ or $(x_i - a)(x_j - b) \leq 0$.

2.5.4 Signed Square Functions

For pairs of constraints of the form (23) with $\ell = u$ and $a = 2$, a special presolving is applied. Assume, two constraints

$$\operatorname{sign}(x + b_1)(x + b_1)^2 + c_1 z = u_1 \tag{36a}$$

$$\operatorname{sign}(x + b_2)(x + b_2)^2 + c_2 z = u_2 \tag{36b}$$

with $c_1 \neq 0$, $c_2 \neq 0$, and $c_1u_2 = c_2u_1$ are given.

Subtracting c_1·(36b) from c_2·(36a) yields

$$c_2 \operatorname{sign}(x + b_1)(x + b_1)^2 = c_1 \operatorname{sign}(x + b_2)(x + b_2)^2 \tag{37}$$

If $c_1 = c_2$, then also $u_1 = u_2$. If $b_1 \neq b_2$, then (37) (and thus the system (36)) has no solution due to strict monotonicity of $\operatorname{sign}(x)|x|$. In the opposite case, $b_1 = b_2$, (36a) and (36b) are identical.

If $c_1 \neq c_2$, then (37) can be shown to have a unique solution, which is

$$x = \frac{b_2 - b_1}{\operatorname{sign}(c_2/c_1)\sqrt{|c_2/c_1|} - 1} - b_1.$$

As a consequence,

$$z = \frac{b_1 - \operatorname{sign}(x + b_1)(x + b_1)^2}{c_1}$$

and both constraints (36a) and (36b) can be removed from the problem.

2.6 Branching

If an infeasible relaxation solution $\tilde{x}$ is not cut off by bound tightening or separation SCIP performs variable-based branching. The violations of integrality and of nonlinear constraints are treated hierarchically as suggested in [11]: if a fractional integer variable $\tilde{x}_i \notin \mathbb{Z}$, $i \in I$, is present, integer branching is performed; if $\tilde{x}$ is integer feasible, spatial branching on a variable that appears nonlinearly in a violated nonlinear constraint is performed.

For *integer branching* SCIP applies a hybrid reliability rule in order to select one variable among the candidate set of fractional variables, see [1, 4] for details. Hybrid reliability branching has proven one of the most successful rules in state-of-the-art mixed-integer linear programming solvers. For *spatial branching*, each unfixed variable x_i that appears in a nonconvex nonlinear constraint that is violated by the current relaxation solution $\tilde{x}$ is included into the set of branching candidates:

- for general nonlinear constraints (15), include variables x_i with $i \in J_j$ for concave or indefinite $h_j(\cdot)$, $j = 1,\ldots,k$;
- for odd or signed power constraints (23), include the variable x;
- for nonconvex quadratic constraints (25), include variables in bilinear terms[3] and in concave terms ax_i^2 with $a < 0$.

By default, *pseudocosts* [11, 16] are used in order to select a promising candidate variable for spatial branching. Pseudocosts give an estimate of the objective change in the LP relaxation relative to the change caused by branching downwards and upwards. In classical pseudocost branching for integer variables, this change is quantified by the distance of $\tilde{x}_i$ to the nearest integers [16]. For continuous variables, a measure that is similar to "rb-int-br" in [11] is used: the distance of $\tilde{x}_i$ to the bounds $\underline{x}_i$ and $\overline{x}_i$ for a variable x_i. If the domain of x_i is unbounded, then a measure for the "infeasibility" of the variable x_i similar to "rb-inf" in [11] is used.

The domain $[\underline{x}_i, \overline{x}_i]$ of the selected variable is split at a branching point x_i' in the interior of the domain. This point is chosen by projecting $\tilde{x}_i$ onto $[\lambda\underline{x}_i + (1-\lambda)\overline{x}_i, \lambda\overline{x}_i + (1-\lambda)\underline{x}_i]$ with $\lambda = 0.2$. When branching on variables appearing in violated power constraints (23) with $\underline{x} < -b < \overline{x}$, $x' := -b$ is chosen as branching point. This allows to exploit convexity/concavity of the power terms in the resulting child nodes. For more details see [87, Section 6.1.4].

2.7 Primal Heuristics

The primal bound on the optimal objective function value is used to prune suboptimal nodes of the search tree and to perform problem reductions in the bound tightening procedures described in Section 2.2, which in turn leads to tighter underestimators and an improved dual bound. Hence, finding good feasible solutions early during the search is not only beneficial in practice, but helps to speed up the global solution process – this is the objective of primal heuristics. In the following we will give a brief overview over the primal heuristics available in SCIP. For more details and a discussion on recent research efforts in this direction we refer to the exhaustive treatment of the topic in [19].

2.7.1 MIP Heuristics

When solving MINLPs, SCIP still makes use of all its default MIP primal heuristics [17]. Most of these heuristics aim at finding good integer and LP feasible solutions starting from an optimum of the LP relaxation or the incumbent solution. If lucky, the solutions found also satisfy the nonlinear constraints. Otherwise, they provide a reference point for the NLP local search heuristic.

2.7.2 NLP Local Search

The NLP local search heuristic considers the space of continuous variables, i.e., it searches for a local optimum of the NLP obtained from the MINLP by fixing all integer variables to the values of a reference point (e.g., integer-feasible solution of the LP relaxation), which is also used as a starting point for the NLP solver. Each feasible solution of this sub-NLP is also a feasible solution of the MINLP. The framework of SCIP allows to switch between several NLP solvers, which may be beneficial depending on the problem at hand. Currently, SCIP uses only the interior point solver IPOPT [89].

After fixing the discrete variables and before transferring the resulting problem to the NLP solver, SCIP applies presolving in order to find further problem reductions implied by the discrete fixings. This may also eliminate redundancies that could create trouble in the NLP solver and may potentially detect infeasibility of the NLP early.

[3] For bilinear terms $x_i x_j$ that involve binary variables, only the binary variable is considered for branching, since it linearizes this term in both child nodes. If a bilinear term involves unbounded variables, only the unbounded variables are considered for branching.

The heuristic is called at the root node and every few hundred nodes during the search. SCIP uses a dynamic strategy based on iteration limits, on the earlier success of the heuristic, and on the availability of an integer feasible reference point to control exactly when the heuristic is executed. Note, that also the solutions found by the MINLP-specific heuristics outlined in the following are used as starting points.

2.7.3 Undercover Heuristic

The Undercover heuristic developed in [21, 22] is based on the observation that it often suffices to fix only a comparatively small number of variables such as to yield a subproblem with all constraints being linear. This sub-MIP is less complex to solve and its solutions are immediately feasible for the original MINLP. The variables to fix are chosen by solving a set covering problem, which aims at minimizing the number of variables to fix. The values for the fixed variables are taken from the solution of the LP or NLP relaxation or a known feasible solution of the MINLP.

The sub-MIP is solved by a new SCIP instance, where limits on the number of nodes and the number of nodes without improvement in the primal bound are used to restrict the effort spend for solving the subproblem. By default, this powerful heuristic is called during root node processing.

2.7.4 Sub-MINLP Heuristics

The idea of large neighborhood search (LNS) heuristics is to restrict the search for "good" solutions to a neighborhood of promising, usually near-optimal or almost feasible solutions. The heuristically restricted problem is obtained by fixing variables or imposing additional constraints that make the search easier and hopefully still allow for finding high-quality solutions.

SCIP's LNS heuristics for the MIP relaxation have been extended to work on the full constraint integer program in a generic fashion [26], and are hence also available for MINLPs. The definition of the neighborhood is the distinguishing feature of an LNS heuristic and is crucial for its success. In the following, we list LNS heuristics typically used by SCIP.

RINS, short for *relaxation induced neighborhood search* [34] searches the neighborhood of the incumbent MINLP solution and the LP relaxation defined by fixing all integer variables that take the same value in both solutions.

RENS, short for *relaxation enforced neighborhood search* [20] searches the neighborhood of all feasible integer roundings of the LP or NLP relaxation. Because it does not require an incumbent solution, it can be used as a start heuristic.

Crossover is an improvement heuristic that is inspired by genetic algorithms [17, 79] and requires more than one feasible solution. For a set of feasible solutions it fixes variables that take identical values in all of them.

Local Branching [38] measures the distance to the starting point in Manhattan norm on the integer variables and only considers solutions which are inside a k-neighborhood of the reference solution where k is typically between 10 and 20.

DINS [46] combines the ideas of RINS and local branching. It defines the neighborhood by introducing a distance function between the incumbent solution and the optimum of the LP relaxation. When applied during a branch-and-bound search, it further takes into account how variables change their values at different nodes of the tree.

3 Impact of Solver Components

In the following, we investigate the impact of individual SCIP components onto the computational performance for solving MINLPs. This analysis is inspired by the computational study in [27], which investigates the impact of solver components for an earlier version of SCIP over a smaller set of mixed-integer quadratically constrained programs. By now SCIP can handle general MINLPs and the considerably extended instance library MINLPLib2[4] provides a much larger test bed. To

[4] `http://www.gamsworld.org/minlp/minlplib2/html/index.html`

measure the impact of individual components, we compare SCIP's performance with default settings to its performance with one feature disabled or switched to a different strategy. Since many MINLP instances contain a considerable linear and discrete part, we also investigate the effect of the classical MIP components. All in all, we compare 14 alternative settings against the SCIP default.

3.1 Computational Setup

For our experiments we used the publicly available benchmark library MINLPLib2[4] as of April 4, 2014 (revision 171). Amongst others, this includes instances from the first MINLPLib [32] and from the recent CMU-IBM intiative minlp.org [51]. MINLPLib2 puts a focus on models that bear relevance in practice. At the time of writing it constitutes the largest single collection of publicly available MINLP test instances. We included all 789 instances from MINLPLib2 that were available in OSiL format and could be read by SCIP (in particular, this excludes instances that utilize trigonometic functions, which are not supported by SCIP so far).

The computations were performed on a cluster of Dell PowerEdge M610 blades with 48 GB RAM, Intel Xeon X5672 CPUs running at 3.20 GHz, and Linux 3.13 (64bit). We used SCIP 3.1 with SoPlex 2.0.0 as LP solver and Ipopt 3.11.8 as NLP solver. Ipopt was built with MA27 as underlying linear equations solver. We set a memory limit of 30 GB in SCIP. Note that this applies to its internal data structures and does not include the memory usage of the LP and NLP solvers.

We set an optimality gap tolerance of 10^{-4} and a primal feasibility tolerance of 10^{-6} with respect to the original problem formulation. For increased numerical stability, we modified SCIP such as to use a slightly tighter feasibility tolerance of 10^{-7} during the solving process, which is with respect to the presolved problem formulation.

For our final comparison, we imposed a time limit of one hour per instance and setting. However, in order to use our available computing resources most effectively and to avoid unnecessary computations, we performed a preliminary run with default settings and a time limit of two hours and removed the instances that could not be solved. Arguably, since the alternative settings described below are typically worse than the default, most of these instances would time out also with the other 14 parameter settings. This resulted in a test set of 475 instances (listed in the supplementary material). Out of these, SCIP default solves 455 instances within the time limit of one hour. For 11 of the 12 instances where SCIP stopped at the time limit, a feasible solution was found. On 8 more instances, SCIP declared a non-optimal solution as optimal or aborted due to numerical difficulties. None of the instances terminated due to reaching the memory limit with any of the evaluated settings.

3.2 Benchmarking Methodology

In addition to the comparison on the whole set of instances, we evaluated the performance measures on the following subsets of instances:

solved + ≥ 100 s: Comparisons of solution times or number of nodes for instances where at least one of the solvers stopped at a time limit are strongly biased by the choice of the time limit. Hence, we consider the subset of instances that were solved by both settings. Additionally, in order to focus on interesting, harder instances, we exclude trivial instances that could be solved with both default and non-default setting in less than 100 seconds.

diff: Many of the analyzed features may only have an effect on a certain subset of the test set, e.g., changing the tree search strategy only has an effect for instances that are not already solved in the root node. For this reason, we consider the set of instances where the runs with default and non-default settings produced different numbers of exploited branch-and-bound nodes or different accumulated numbers of LP simplex iterations.

diff + solved: As described above, we also consider the subset of instances from the "diff" set that were solved by both settings.

diff + ≥ 100 s: As described above, we also consider the subset of harder instances from the "diff" set where SCIP ran for at least 100 seconds with default or non-default settings.

diff + solved + ≥ 100 s: Finally, we consider the set of instances given by the combination of all these restrictions.

For some settings, SCIP fails on a few instances, e.g., by computing a wrong dual bound. We discard these instances for the evaluation of this setting. In the following tables, the column "size" gives the number of instances in the respective subsets. The remaining columns provide several performance indicators for a comparison with the SCIP default settings. As a rough indicator of the usefulness of a component, the third column reports how many instances more or less were solved. Next, for instances that could not be solved within the time limit, we count on how many instances the primal and dual bounds were "better" or "worse" by at least 10%, respectively. Additionally, we display the absolute number of instances for which a particular setting was faster or slower by more than 10% and by at least one second.

Further, we compare the shifted geometric means of the primal-dual integral, the number of branch-and-bound nodes, and the overall running time. The primal-dual integral (PDI) introduced in [18] is computed as the integral of the relative gap between the primal and dual bound over the overall running time. This measure favors runs where good solutions and tight relaxations are produced early. As pointed out by [19, p. 38], compared to other performance measures, "it is less prone, though not immune, to common weaknesses of standard performance measures, notably the dependence on an (arbitrarily chosen) time limit". For the PDI and the running time, a shift of one second was used, for the number of nodes a shift of 100 nodes.

With considering three different performance measures we aim at making our evaluation more robust. However, comparing performance only based on mean values may still be misleading: it does not allow us to draw conclusions on how a change in the mean is distributed over the test set. A reduction in mean running time may stem from a consistent improvement over the whole test set, or could be the result of a drastic speedup on only few instances while the performance on the majority of instances deteriorates. Especially for experiments with algorithms that exhibit high performance variability, it is crucial to carefully analyze computational results in this respect. By *performance variability*, we understand the occurrence of considerable changes in the performance of an algorithm caused by small and seemingly insignificant modifications to its implementation. As noted by [35], this phenomenon can be quite pronounced for state-of-the-art MIP solvers. Though this has not been thoroughly studied yet, it is our experience that performance variability looms even larger on mixed-integer nonlinear programs, amongst others because of the effects of branching on continuous variables.

Hence, we use *statistical hypothesis testing* to analyze how consistently a change in performance is distributed across the test set. Because there is no ground to assume that, e.g., running times adhere to a specific distribution such as the normal distribution, we employ methods from nonparametric statistics. A suitable nonparametric test for the setting described above is the *Wilcoxon signed-rank test* [90], see also the detailed description in [52, Sec. 3.2] in the context of comparing MIP solvers. We use the implementation of the Wilcoxon signed-rank test available in the SciPy package [64, 74] with parameters `correction=False` (default) and `zero_method="pratt"`. The Pratt treatment removes anomalies due to samples with no performance difference [76]. In the tables, we indicate the significance of the relative change in overall solution time by one, two, or three '+' signs if the p-value is below 5%, below 0.5%, and below 0.05%, respectively. Note that the concept of significance is independent of the magnitude of change in the shifted geometric mean.

3.3 Reformulation

Table 1 summarizes the impact of reformulations working on quadratic and on general nonlinear equations, respectively.

Table 1: Impact of components affecting the reformulation.

settings		better : worse				relative difference		
subset	size	solved	primal	dual	time	PDI	nodes	time
quadratic reformulation	465	4 : 10	0 : 1	4 : 2	33 : 55	9%	-13%	8% +++
solved + $\geq$ 100 s	89	0 : 0	0 : 0	0 : 0	15 : 17	5%	-30%	-2% +
diff	167	4 : 10	0 : 1	4 : 2	33 : 54	23%	-32%	18% +++
diff + solved	144	0 : 0	0 : 0	0 : 0	29 : 43	13%	-34%	7% ++
diff + $\geq$ 100 s	57	4 : 10	0 : 1	4 : 2	19 : 28	43%	-48%	33%
diff + solved + $\geq$ 100 s	34	0 : 0	0 : 0	0 : 0	15 : 17	12%	-61%	-6%
expression reformulation	464	0 : 66	2 : 10	19 : 35	4 : 76	167%	283%	153% +++
solved + $\geq$ 100 s	85	0 : 0	0 : 0	0 : 0	1 : 8	29%	86%	43% +++
diff	108	0 : 66	2 : 10	19 : 35	4 : 76	4542%	28250%	4593% +++
diff + solved	30	0 : 0	0 : 0	0 : 0	4 : 10	166%	1217%	269% ++
diff + $\geq$ 100 s	87	0 : 66	2 : 10	19 : 35	1 : 74	10762%	87520%	11223% +++
diff + solved + $\geq$ 100 s	9	0 : 0	0 : 0	0 : 0	1 : 8	1102%	35964%	3004% +

3.3.1 Reformulation of Quadratic Constraints

With setting quadratic reformulation, we evaluate disabling the recognition of second-order cone constraints (Section 2.4.3), the reformulation of products with binary variables (Section 2.5.2), and the recognition of complementarity constraints (Section 2.5.3).

Out of the 465 instances, only 167 were affected by this setting. These are mostly instances that contain products with binary variables or second-order cone (SOC) constraints. The recognition of the latter is crucial for all affected instances, as not recognizing structure (29) leads to weak cuts and branching on variable x_n in the original formulation (28). For example, on the `portfol_robust*` and `portfol_shortfall*` instances, disabling SOC recognition leads to a substantial increase in both computing time and number of nodes.

The reformulation of products with binary variables (Section 2.5.2) is sometimes but not always beneficial. With respect to the number of solved instances, the reformulation is essential for 10 instances, but also obstructive for 4 other instances. To understand this, consider the `edgecross*` instances, which become a MIP after this reformulation step. Even though solving this MIP requires significantly more nodes than the (non-reformulated) MINLP, the node processing time for the MIP is also much reduced. A reason for this behavior is that in the MINLP formulation cuts are generated throughout the tree search. These cuts are essentially the same big-M constraints that are created by the reformulation step, but with a tighter value of M due to tightened variable bounds. A potential for improving SCIP's linearizations of a product with a binary variable is to find a compromise between using a "static" big-M during presolve and a "dynamic", i.e., tighter, but only locally valid big-M during tree search, possibly by employing SCIP's implementation of indicator constraints. See also [15] for a detailed discussion.

Note that the average reduction of the number of nodes stems from an increase on 62 instances (probably due to turning off SOC detection) and a decrease on 74 instances (probably due to not reformulating products with binary variables). We refer to [27, 87] for previous, separate analyses on the impact of these reformulation steps.

3.3.2 Reformulation of the Expression Graph

With setting expression reformulation, we evaluate the loss in performance when not moving subexpressions of nonlinear expressions into additional constraints (Section 2.5.1). When disabling this reformulation, indefinite non-quadratic terms in general nonlinear constraints are under- and

Table 2: Impact of components related to bound tightening.

settings		better : worse				relative difference			
subset	size	solved	primal	dual	time	PDI	nodes	time	
domain propagation	456	0 : 36	0 : 3	7 : 18	57 : 143	85%	150%	86%	+++
solved + $\geq$ 100 s	91	0 : 0	0 : 0	0 : 0	25 : 50	77%	146%	70%	+
diff	430	0 : 36	0 : 3	7 : 18	57 : 143	89%	162%	92%	+++
diff + solved	380	0 : 0	0 : 0	0 : 0	57 : 104	34%	67%	32%	+++
diff + $\geq$ 100 s	141	0 : 36	0 : 3	7 : 18	25 : 89	313%	730%	363%	+++
diff + solved + $\geq$ 100 s	91	0 : 0	0 : 0	0 : 0	25 : 50	77%	146%	70%	+
OBBT	466	1 : 25	0 : 1	13 : 4	67 : 76	32%	89%	46%	+++
solved + $\geq$ 100 s	92	0 : 0	0 : 0	0 : 0	24 : 22	-5%	67%	33%	
diff	354	1 : 25	0 : 1	13 : 4	67 : 76	43%	129%	63%	+++
diff + solved	317	0 : 0	0 : 0	0 : 0	66 : 51	-6%	32%	7%	+++
diff + $\geq$ 100 s	111	1 : 25	0 : 1	13 : 4	25 : 47	199%	833%	362%	++
diff + solved + $\geq$ 100 s	74	0 : 0	0 : 0	0 : 0	24 : 22	-6%	89%	42%	
conflict analysis	467	0 : 4	0 : 0	2 : 2	44 : 58	1%	7%	2%	+++
solved + $\geq$ 100 s	89	0 : 0	0 : 0	0 : 0	19 : 31	2%	15%	6%	
diff	257	0 : 4	0 : 0	2 : 2	44 : 58	2%	12%	4%	+++
diff + solved	241	0 : 0	0 : 0	0 : 0	44 : 54	1%	11%	2%	+++
diff + $\geq$ 100 s	92	0 : 4	0 : 0	2 : 2	19 : 35	5%	20%	12%	+
diff + solved + $\geq$ 100 s	76	0 : 0	0 : 0	0 : 0	19 : 31	2%	17%	7%	

overestimated using interval gradients, see (22). Note, that domain propagation does not suffer from disabling this reformulation.

Only 108 instances are affected by this setting. Conversely, this shows that for three quarters of the instance set all nonlinear constraint functions are either quadratic, convex, or univariate. However, the numbers in Table 1 show clearly that this reformulation is essential: if disabled, 66 instances are solved less, no instance is solved additionally, and the time and number of nodes required for the 30 instances in the diff + solved subset increases considerably. Thus, the interval gradient based estimator (22) in its current form does not provide a tight relaxation in many situations.

However, there are instances in the test set that can be solved faster by disabling this reformulation. On the instance `clay0205h` from the subset solved + $\geq$ 100 s, disabling reformulation increases the number of nodes by 440%, but decreases the size of the LP relaxation (less cutting planes and a smaller non-reformulated problem) and hence the node processing time. As a result, the overall running time is decreased by 36%.

3.4 Domain Propagation and Conflict Analysis

Table 2 summarizes the impact of variable bound tightening.

3.4.1 Domain Propagation

With setting domain propagation, we evaluate the loss in performance when disabling all domain propagation routines (Section 2.2) during presolve and in node preprocessing. Note, that constraint-based bound tightening may still be performed when enforcing violated constraints.

With this setting, a significant and large degradation in performance on a broad set of instances can be observed: 94% of the instance set is affected. This underlines the importance of bound tightening for MINLP. On the set of instances that were solved neither with or without domain propagation, a

worsening of the dual bound at termination can be observed in the majority of instances, while the primal bound stayed unaffected for most of these instances.

3.4.2 Optimization-Based Bound Tightening

With setting OBBT, we evaluate the effect of disabling only OBBT (Section 2.2.2) on the performance of SCIP. With 76% of the instance set, OBBT affects many instances. When disabling OBBT, SCIP solves 24 instances less and on average OBBT improves the running time. However, the numbers also show that there are many instances for which the extra time spent by OBBT often does not seem to pay off. This shows that designing a good automatic default strategy for OBBT is nontrivial. We refer to [48] for a more in-depth analysis of OBBT performance.

3.4.3 Conflict Analysis

With setting conflict analysis, we evaluate the effect of disabling conflict analysis, that is, not trying to learn additional constraints from the analysis of bound changes that lead to pruned subproblems [2]. This setting affects 55% of the test set. When averaging over these instances, disabling conflict analysis leads to almost no difference in running time and slightly increases the number of enumerated nodes. On harder instances (solving time $\geq$ 100s), conflict analysis slightly improves running time. Most importantly, SCIP solves four more instances and no instance less when conflict analysis is enabled, making it a good default setting.

We want to note that the significance of this comparison is limited by the fact that currently most nonlinear constraints in SCIP do not participate in conflict analysis. Extending SCIP in this respect could shorten the conflicts that are found and thus lead to a larger improvement of the performance due to conflict analysis.

3.5 Separation

Table 3 summarizes the impact of generating a tight linear relaxation.

3.5.1 MIP Cutting Planes

With setting MIP cuts, we evaluate the effect of disabling cutting planes that cut off fractional solutions, e.g., Gomory, mixed-integer rounding, and flowcover cuts. This setting leads to different search trees or number of simplex iterations on 85% of the instances. The average tree sizes and running times increase considerably when turning these cuts off. On 77 instances, however, the time to compute MIP cuts and to solve the resulting LPs does not pay off. On hard instances ($\geq$ 100s) the improvement due to MIP cuts is substantial, especially since 32 instances are solved less when disabling MIP cuts. Overall, MIP cuts are beneficial, both on average and in the number of solved instances.

3.5.2 Outer Approximation of Nonlinear Constraints

With setting outer approximation, we evaluate the effect of disabling the separation of linear cuts that approximate the nonlinear constraints (see Section 2.4) during node preprocessing. Note, that cuts of this type may still be generated while resolving violations of nonlinear constraints by the LP relaxation solution. However, as SCIP first resolves fractionalities in integer variables, an outer approximation of nonlinear constraints will only be separated for nodes with integer feasible LP relaxation solution.

This setting affects most instances that were not reformulated into a MIP during presolve. It has drastic effects on the number of solved instances (99 less), average solving time, and dual bounds (both when hitting the time limit and in the early phase of the tree search, see PDI). The separation also has a positive effect on the primal bound for 12 instances, probably because the tighter relaxation helped to guide the search of primal heuristics that depend on the relaxation such as

Table 3: Impact of cutting plane/outer approximation separators.

settings		better : worse				relative difference		
subset	size	solved	primal	dual	time	PDI	nodes	time
MIP cuts	460	4 : 36	1 : 0	20 : 14	77 : 119	66%	110%	59% +++
solved + ≥ 100 s	85	0 : 0	0 : 0	0 : 0	29 : 37	52%	103%	55%
diff	393	4 : 36	1 : 0	20 : 14	77 : 119	78%	136%	71% +++
diff + solved	344	0 : 0	0 : 0	0 : 0	73 : 83	20%	65%	20% +++
diff + ≥ 100 s	127	4 : 36	1 : 0	20 : 14	33 : 73	327%	487%	313% +++
diff + solved + ≥ 100 s	78	0 : 0	0 : 0	0 : 0	29 : 37	57%	116%	61%
outer approximation	465	1 : 100	0 : 12	5 : 85	39 : 205	352%	691%	288% +++
solved + ≥ 100 s	71	0 : 0	0 : 0	0 : 0	12 : 47	172%	749%	215% ++
diff	414	1 : 100	0 : 12	5 : 85	39 : 205	431%	914%	353% +++
diff + solved	301	0 : 0	0 : 0	0 : 0	38 : 105	67%	235%	73% +++
diff + ≥ 100 s	176	1 : 100	0 : 12	5 : 85	13 : 147	2679%	7115%	2012% +++
diff + solved + ≥ 100 s	63	0 : 0	0 : 0	0 : 0	12 : 47	209%	1015%	265% ++

rounding, diving, or some large neighborhood search heuristics. However, there are also 39 instances where the extra time for generating these cuts and solving the corresponding LPs does not pay off.

3.6 Tree Search

Table 4 summarizes the impact of node selection and branching rules.

3.6.1 Node Selection

With setting breadth first, we evaluate the effect of switching the node selection rule from the default best estimate with plunging rule to selecting nodes in breadth first manner. As expected, this setting affects the majority of the instances (83%). The numbers show that the default node selection rule is significantly better than breadth first search with respect to all considered performance measures. The increase in running time is even larger than the increase in the number of nodes, because more "jumps" through the tree cause higher node processing times.

3.6.2 Branching

Here we evaluate the effect of switching the rule for selecting a nonlinear variable during spatial branching to

- inference branching: select a variable that previously lead to many bound tightenings after reducing its domain [1],
- most infeasible branching: select a variable that appears in a nonconvex term of a most-violated nonlinear constraint, and
- random branching: select randomly any variable that appears in a nonconvex term of a violated constraint.

See Section 2.6 for the default behavior. Note, that we did not alter the branching rule for selecting an integer variable with fractional solution value in the LP relaxation.

These settings only affect instances where branching on (several) variables in nonconvex terms is necessary. Apparently, this is the case for less than a quarter of the instance set. With inference

Table 4: Impact of components affecting the tree search.

settings		better : worse				relative difference		
subset	size	solved	primal	dual	time	PDI	nodes	time
breadth first	465	1 : 21	0 : 8	1 : 20	17 : 170	47%	25%	38% +++
solved + $\geq$ 100 s	92	0 : 0	0 : 0	0 : 0	9 : 74	137%	82%	110% +++
diff	384	1 : 21	0 : 8	1 : 20	17 : 170	55%	30%	47% +++
diff + solved	351	0 : 0	0 : 0	0 : 0	16 : 150	46%	30%	41% +++
diff + $\geq$ 100 s	124	1 : 21	0 : 8	1 : 20	10 : 94	157%	68%	118% +++
diff + solved + $\geq$ 100 s	91	0 : 0	0 : 0	0 : 0	9 : 74	139%	84%	111% +++
inference branching	467	0 : 26	0 : 3	0 : 11	8 : 35	24%	32%	31% +++
solved + $\geq$ 100 s	82	0 : 0	0 : 0	0 : 0	2 : 2	-3%	-2%	-1% +++
diff	104	0 : 26	0 : 3	0 : 11	8 : 35	136%	236%	209% +++
diff + solved	65	0 : 0	0 : 0	0 : 0	8 : 8	-7%	3%	3% ++
diff + $\geq$ 100 s	50	0 : 26	0 : 3	0 : 11	2 : 29	453%	1028%	855% +++
diff + solved + $\geq$ 100 s	11	0 : 0	0 : 0	0 : 0	2 : 2	-18%	-15%	-6%
most infeasible branching	466	0 : 22	0 : 1	0 : 6	4 : 37	19%	32%	29% +++
solved + $\geq$ 100 s	84	0 : 0	0 : 0	0 : 0	2 : 6	-1%	12%	11% +++
diff	109	0 : 22	0 : 1	0 : 6	4 : 37	99%	211%	179% +++
diff + solved	74	0 : 0	0 : 0	0 : 0	4 : 14	0%	25%	25% +++
diff + $\geq$ 100 s	49	0 : 22	0 : 1	0 : 6	2 : 29	309%	968%	731% +++
diff + solved + $\geq$ 100 s	14	0 : 0	0 : 0	0 : 0	2 : 6	-3%	94%	93%
random branching	465	0 : 22	0 : 0	0 : 7	4 : 37	23%	31%	30% +++
solved + $\geq$ 100 s	85	0 : 0	0 : 0	0 : 0	2 : 7	3%	24%	18% +++
diff	104	0 : 22	0 : 0	0 : 7	4 : 37	114%	217%	183% +++
diff + solved	70	0 : 0	0 : 0	0 : 0	4 : 15	11%	44%	36% +++
diff + $\geq$ 100 s	49	0 : 22	0 : 0	0 : 7	2 : 29	324%	892%	672% +++
diff + solved + $\geq$ 100 s	15	0 : 0	0 : 0	0 : 0	2 : 7	17%	231%	152%

branching, the least number of instances are solved, but on those that are solved it gives on average the same performance as the default rule, maybe because there are not too many branching candidates to choose from.

For most infeasible branching and random branching, the degradation in average solving time on the diff + solved instances is relatively small: 25% and 36%, respectively. However, when taking unsolved instances into account, then, due to timeouts on 22 instances, the average running time increases significantly: by almost a factor of three on all affected instances and by about a factor of eight on hard instances. Finally, we want to note that the most infeasible branching rule is only slightly faster than random branching.

3.7 Primal Heuristics

Table 5 summarizes the impact of the primal heuristics in SCIP, see also Section 2.7.

3.7.1 All Primal Heuristics

With setting all heuristics, we evaluate the effect of disabling all primal heuristics in SCIP. Then feasible solutions are only found when the solution of a node LP relaxation is feasible for the MINLP.

This setting affects almost all instances. Disabling primal heuristics leads to an increase in the average PDI and the number of nodes, as feasible solutions are found later and thus less pruning can be done. On the set diff + solved of affected instances that can be solved with and without

Table 5: Impact of primal heuristics.

settings		better : worse				relative difference			
subset	size	solved	primal	dual	time	PDI	nodes	time	
all heuristics	464	3 : 21	1 : 15	3 : 8	120 : 81	57%	36%	9%	+++
solved + $\geq$ 100 s	91	0 : 0	0 : 0	0 : 0	47 : 30	46%	13%	-5%	
diff	444	3 : 21	1 : 15	3 : 8	120 : 81	59%	38%	10%	+++
diff + solved	411	0 : 0	0 : 0	0 : 0	119 : 60	26%	12%	-11%	+++
diff + $\geq$ 100 s	124	3 : 21	1 : 15	3 : 8	48 : 51	193%	130%	85%	
diff + solved + $\geq$ 100 s	91	0 : 0	0 : 0	0 : 0	47 : 30	46%	13%	-5%	
LNS heuristics	466	0 : 7	0 : 0	4 : 3	70 : 39	5%	19%	5%	+++
solved + $\geq$ 100 s	94	0 : 0	0 : 0	0 : 0	32 : 19	2%	11%	5%	+
diff	265	0 : 7	0 : 0	4 : 3	69 : 39	9%	34%	10%	+++
diff + solved	246	0 : 0	0 : 0	0 : 0	69 : 32	-6%	11%	-5%	+++
diff + $\geq$ 100 s	100	0 : 7	0 : 0	4 : 3	32 : 26	45%	85%	54%	
diff + solved + $\geq$ 100 s	81	0 : 0	0 : 0	0 : 0	32 : 19	3%	13%	6%	+
aggressive heuristics	465	3 : 6	0 : 1	2 : 2	50 : 162	19%	-5%	24%	+++
solved + $\geq$ 100 s	98	0 : 0	0 : 0	0 : 0	31 : 52	8%	-7%	9%	
diff	408	3 : 6	0 : 1	2 : 2	50 : 159	20%	-6%	25%	+++
diff + solved	390	0 : 0	0 : 0	0 : 0	49 : 153	16%	-11%	21%	+++
diff + $\geq$ 100 s	116	3 : 6	0 : 1	2 : 2	32 : 58	22%	11%	26%	
diff + solved + $\geq$ 100 s	98	0 : 0	0 : 0	0 : 0	31 : 52	8%	-7%	9%	

heuristics, disabling primal heuristics reduces the average running time by 11%. However, this comes at the cost of increasing the average PDI by 26%, and looking at the restricted subset diff + solved + $\geq$ 100 s this reductions seems to stem from easier instances. Most importantly, 18 instances less are solved within the time limit when disabling primal heuristics completely. On hard instances (diff + $\geq$ 100 s) the average running time is almost doubled.

3.7.2 Large Neighborhood Search Heuristics

With setting LNS heuristics, we evaluate the effect of disabling all LNS heuristics in SCIP (Crossover, RENS, RINS, and Undercover), see also Section 2.7.4. That is, only the comparatively cheaper rounding, diving, and sub-NLP heuristics remain enabled.

This setting affects 57% of the instance set. On hard instances, disabling LNS heuristics leads to solving 7 instances less and an increase in PDI, average running time, and number of nodes. However, on easy instances or instances that can be solved without primal heuristics, LNS heuristics seem to be mostly a waste of computing time. Even the PDI decreases slightly by disabling LNS heuristics, which indicates that cheaper heuristics can replace LNS heuristics to find sufficiently good solutions (when averaged over the diff + solved test set). A similar conclusion can be drawn from the numbers in the column "primal": On none of the instances, LNS heuristics improve the final primal bound by more than 10%.

3.7.3 Aggressive Primal Heuristics

With setting aggressive heuristics, we evaluate the effect of running the primal heuristics more frequently, e.g., root-only heuristics may now also run during tree search and heuristics disabled by default are now enabled.

This setting affects 88% of the instances. The numbers indicate that the extra effort spent for primal heuristics does not pay off for the majority of the instances: three less instances are solved,

computing time and PDI are increased, and the number of nodes is reduced only marginally. The average increase in computing time is not significant on hard instances, though.

4 Conclusion

This paper described the extensions that were added to the constraint integer programming framework SCIP to allow it to solve (convex and nonconvex) mixed-integer nonlinear programs to global optimality. These extensions are centered around an expression graph representation of nonlinear constraints, which allows for bound tightening, detection of convex sub-expressions, and reformulation. The latter is necessary to compute and update a linear outer-approximation based on convex over- and underestimation of nonconvex functions. Additionally, we discussed SCIP's implementations of optimization-based bound tightening (OBBT), branching rules, and primal heuristics for MINLP.

In a detailed computational study, we analyzed the impact of several SCIP components on the MINLP solving performance. The results show that disabling any of the investigated components leads to a decrease in the number of solved instances. Except in few cases, the average running time increases whenever a component is disabled. This indicates that the default settings are reasonable.

In particular, we saw that the MINLP-specific features with largest performance impact were the reformulation of the expression graph, the tightening of the outer-approximation on nodes with fractional integer variables, and pseudo-cost-based branching on nonlinear variables. Also domain propagation, OBBT, and MIP cutting planes give a clear benefit. However, the extra effort for OBBT and MIP cutting planes does not necessarily pay off on easy instances. For the selection of a nonlinear variable during spatial branching, our computational comparison indicates that focusing on improving the dual bound is more important than trying to minimize constraint violation. This is parallel to empirical observations made for branching on fractional integer variables in MIP solvers.

The goal of minimizing constraint violations and eventually finding feasible solutions may rather be assigned to primal heuristics, which very much help to find good feasible solutions early. Here, "simple" heuristics like finding a feasible solution for the MIP relaxation via rounding or diving and a subsequent (local) solve of a corresponding NLP seems sufficient for many instances. The more expensive LNS heuristics are mostly beneficial for difficult instances.

Finally, we wish to note that the computational insights gained are valid specifically for SCIP's current implementation and it would be desirable to conduct similar studies for other MINLP solvers. SCIP is actively developed and further improvements that have been made after the release of version 3.1 were not discussed in this paper. In the context of MINLP, these were the addition of a separator for edge-concave cuts, an extended recognition of second-order cone constraints, improvements for the separation of convex quadratic constraints, and enhanced ordering strategies for OBBT, see [43].

Acknowledgements. We want to thank Tobias Achterberg for his initial creation of SCIP and all SCIP developers for their continuous dedication to this project. We would also like to thank the two anonymous referees for their valuable remarks that helped to improve the clarity of the paper.

This work has been supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin, the Berlin Mathematical School, and the Research Campus Modal *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). All responsibility for the content of this publication is assumed by the authors.

References

[1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, TU Berlin, 2007. urn:nbn:de:0297-zib-11129.

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

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

[4] T. Achterberg and T. Berthold. Hybrid branching. In W.-J. van Hoeve and J. N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 309–311. Springer, 2009. doi:10.1007/978-3-642-01929-6_23.

[5] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In L. Perron and M. A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, volume 5015 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2008. doi:10.1007/978-3-540-68155-7_4.

[6] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs – I. Theoretical advances. *Computers & Chemical Engineering*, 22:1137–1158, 1998. doi:10.1016/S0098-1354(98)00027-1.

[7] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. Global optimization of mixed-integer nonlinear problems. *Journal of the American Institute of Chemical Engineers*, 46(9):1769–1797, 2000. doi:10.1002/aic.690460908.

[8] T. Ahadi-Oskui, S. Vigerske, I. Nowak, and G. Tsatsaronis. Optimizing the design of complex energy conversion systems by Branch and Cut. *Computers & Chemical Engineering*, 34(8): 1226–1236, 2010. doi:10.1016/j.compchemeng.2010.03.007.

[9] X. Bao. Automatic convexity detection for global optimization. Master's thesis, University of Illinois at Urbana-Champaign, 2007.

[10] E. M. L. Beale. Branch and bound methods for numerical optimization of non-convex functions. In M. M. Barritt and D. Wishart, editors, *COMPSTAT 80 Proceedings in Computational Statistics*, pages 11–20, Vienna, 1980. Physica-Verlag.

[11] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634, 2009. doi:10.1080/10556780903087124.

[12] P. Belotti, S. Cafieri, J. Lee, and L. Liberti. Feasibility-based bounds tightening via fixed points. In W. Wu and O. Daescu, editors, *Combinatorial Optimization and Applications*, volume 6508 of *Lecture Notes in Computer Science*, pages 65–76. Springer, Berlin/Heidelberg, 2010. doi:10.1007/978-3-642-17458-2_7.

[13] P. Belotti, A. Miller, and M. Namazifar. Linear inequalities for bounded products of variables. *SIAG/OPT Views-and-News*, 12(1):1–8, March 2011.

[14] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013. doi:10.1017/S0962492913000032.

[15] P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gómez, and D. Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3):545–566, 2016. doi:10.1007/s10589-016-9847-8.

[16] M. Bénichou, J. M. Gauthier, P. Girodet, G. Hentges, R. Ribiére, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971. doi:10.1007/BF01584074.

[17] T. Berthold. Primal heuristics for mixed integer programs. Master's thesis, Technische Universität Berlin, 2006. urn:nbn:de:0297-zib-10293.

[18] T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614, 2013. doi:10.1016/j.orl.2013.08.007.

[19] T. Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, TU Berlin, 2014.

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

[21] T. Berthold and A. M. Gleixner. Undercover – a primal heuristic for MINLP based on sub-MIPs generated by set covering. In P. Bonami, L. Liberti, A. J. Miller, and A. Sartenaer, editors, *Proceedings of the European Workshop on Mixed Integer Nonlinear Programming (EWMINLP)*, pages 103–112, CIRM Marseille, France, April 2010.

[22] T. Berthold and A. M. Gleixner. Undercover: a primal MINLP heuristic exploring a largest sub-MIP. *Mathematical Programming*, 144(1-2):315–346, 2014. doi:10.1007/s10107-013-0635-2.

[23] T. Berthold, S. Heinz, and M. E. Pfetsch. Nonlinear pseudo-boolean optimization: relaxation or propagation? In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing – SAT 2009*, number 5584 in Lecture Notes in Computer Science, pages 441–446. Springer, 2009. doi:10.1007/978-3-642-02777-2_40.

[24] T. Berthold, S. Heinz, and S. Vigerske. Extending a CIP framework to solve MIQCPs. In Lee and Leyffer [59], pages 427–444. doi:10.1007/978-1-4614-1927-3_15.

[25] T. Berthold, S. Heinz, M. Lübbecke, R. H. Möhring, and J. Schulz. A constraint integer programming approach for resource-constrained project scheduling. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 313–317. Springer, 2010. doi:10.1007/978-3-642-13520-0_34.

[26] T. Berthold, S. Heinz, M. E. Pfetsch, and S. Vigerske. Large neighborhood search beyond MIP. In L. D. Gaspero, A. Schaerf, and T. Stützle, editors, *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, pages 51–60, 2011. urn:nbn:de:0297-zib-12989.

[27] T. Berthold, A. M. Gleixner, S. Heinz, and S. Vigerske. Analyzing the computational impact of MIQCP solver components. *Numerical Algebra, Control and Optimization*, 2(4):739–748, 2012. doi:10.3934/naco.2012.2.739.

[28] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002. doi:10.1287/opre.50.1.3.17780.

[29] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. MIP: theory and practice – closing the gap. In M. J. D. Powell and S. Scholtes, editors, *System Modelling and Optimization: Methods, Theory and Applications*, pages 19–49. Kluwer Dordrecht, 2000. ISBN 0792378814.

[30] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012. doi:10.1016/j.sorms.2012.08.001.

[31] M. R. Bussieck and S. Vigerske. MINLP solver software. In J. J. C. et.al., editor, *Wiley Encyclopedia of Operations Research and Management Science*. Wiley & Sons, Inc., 2010. doi:10.1002/9780470400531.eorms0527.

[32] M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib - a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003. `http://www.gamsworld.org/minlp/minlplib.htm`. doi:10.1287/ijoc.15.1.114.15159.

[33] A. Caprara and M. Locatelli. Global optimization problems and domain reduction strategies. *Mathematical Programming*, 125:123–137, 2010. doi:10.1007/s10107-008-0263-4.

[34] E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2004. doi:10.1007/s10107-004-0518-7.

[35] E. Danna. Performance variability in mixed integer programming. Talk at Workshop on Mixed Integer Programming 2008, Columbia University, New York, NY, USA, June 2014.

[36] F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 15 (3):404–429, 2010. doi:10.1007/s10601-009-9076-1.

[37] J. E. Falk and K. R. Hoffman. A successive underestimation method for concave minimization problems. *Mathematics of Operations Research*, 1(3):251–259, 1976. doi:10.1287/moor.1.3.251.

[38] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003. doi:10.1007/s10107-003-0395-5.

[39] C. A. Floudas. *Nonlinear and Mixed Integer Optimization: Fundamentals and Applications*. Oxford University Press, New York, 1995. ISBN 0195100565.

[40] C. A. Floudas, I. G. Akrotirianakis, C. Caratzoulas, C. A. Meyer, and J. Kallrath. Global optimization in the 21st century: Advances and challenges. *Computers & Chemical Engineering*, 29(6):1185–1202, 2005. doi:10.1016/j.compchemeng.2005.02.006.

[41] J. J. H. Forrest and J. A. Tomlin. Branch and bound, integer, and non-integer programming. *Annals of Operations Research*, 149(1):81–87, 2007. doi:10.1007/s10479-006-0112-x.

[42] R. Fourer, C. Maheshwari, A. Neumaier, D. Orban, and H. Schichl. Convexity and concavity detection in computational graphs: Tree walks for convexity assessment. *INFORMS Journal on Computing*, 22(1):26–43, 2009. doi:10.1287/ijoc.1090.0321.

[43] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, S. Vigerske, D. Weninger, M. Winkler, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 3.2. ZIB Report 15-60, Zuse Institute Berlin, 2016. urn:nbn:de:0297-zib-57675.

[44] C. Gau and L. Schrage. Implementation and testing of a branch-and-bound based method for deterministic global optimization: Operations research applications. In C. A. Floudas and P. M. Pardalos, editors, *Frontiers in Global Optimization*, volume 74 of *Nonconvex Optimization and Its Applications*, pages 145–164. Springer, 2003. ISBN 978-1-4020-7699-2. doi:10.1007/978-1-4613-0251-3_9.

[45] D. M. Gay. Bounds from slopes. Technical report, Sandia National Laboratories, 2010. URL `https://cfwebprod.sandia.gov/cfdocs/CCIM/docs/bounds10.pdf`.

[46] S. Ghosh. DINS, a MIP improvement heuristic. In M. Fischetti and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization - 12th International Conference*, volume 4513 of *Lecture Notes in Computer Science*, pages 310–323, 2007. doi:10.1007/978-3-540-72792-7_24.

[47] A. M. Gleixner and S. Weltge. Learning and propagating Lagrangian variable bounds for mixed-integer nonlinear programming. In C. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013*, volume 7874 of *LNCS*, pages 355–361. Springer, 2013. doi:10.1007/978-3-642-38171-3_26.

[48] A. M. Gleixner, T. Berthold, B. Müller, and S. Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67:731–757, 2017. doi:10.1007/s10898-016-0450-4.

[49] C. Gounaris and C. A. Floudas. A review of recent advances in global optimization. *Journal of Global Optimization*, 45:3–38, 2009. doi:10.1007/s10898-008-9332-8.

[50] I. E. Grossmann and Z. Kravanja. Mixed-integer nonlinear programming: A survey of algorithms and applications. In A. R. Conn, L. T. Biegler, T. F. Coleman, and F. N. Santosa, editors, *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*, volume 93 of *The IMA Volumes in Mathematics and its Applications*, pages 73–100. Springer, New York, 1997. doi:10.1007/978-1-4612-1960-6_5.

[51] I. E. Grossmann and J. Lee. Cyberinfrastructure for mixed-integer nonlinear programming. *SIAG/OPT Views-and-News*, 22(1):8–12, 2011. URL `http://www.minlp.org`.

[52] G. Hendel. Empirical analysis of solving phases in mixed integer programming. Master's thesis, Technische Universität Berlin, 2014. urn:nbn:de:0297-zib-54270.

[53] J. N. Hooker. *Integrated Methods for Optimization*, volume 170 of *International Series in Operations Research & Management Science*. Springer, New York, 2007. doi:10.1007/978-1-4614-1900-6.

[54] R. Horst and P. Pardalos. *Handbook of Global Optimization*, volume 2 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 1995. ISBN 978-0-7923-3120-9.

[55] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, Berlin, 1990. ISBN 978-3-540-61038-0.

[56] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244(5):1093–1096, 1979. english translation in Soviet Math. Dokl. 20(1):191–194, 1979.

[57] 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 – Mixed Integer Programming Library version 5. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi:10.1007/s12532-011-0025-9.

[58] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[59] J. Lee and S. Leyffer, editors. *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*. Springer, 2012. doi:10.1007/978-1-4614-1927-3.

[60] L. Liberti and C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25(2):157–168, 2003. doi:10.1023/A:1021924706467.

[61] M. Locatelli and F. Schoen. *Global Optimization: Theory, Algorithms, and Applications*. Number 15 in MOS-SIAM Series on Optimization. SIAM, 2013.

[62] C. D. Maranas and C. A. Floudas. Global optimization in generalized geometric programming. *Computers & Chemical Engineering*, 21(4):351–369, 1997. doi:10.1016/S0098-1354(96)00282-7.

[63] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I – convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976. doi:10.1007/BF01580665.

[64] K. J. Millman and M. Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011. doi:10.1109/MCSE.2011.36.

[65] R. Misener and C. A. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57(1):3–50, 2012. doi:10.1007/s10898-012-9874-7.

[66] R. Misener and C. A. Floudas. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014. doi:10.1007/s10898-014-0166-2.

[67] M. Mönnigmann. Efficient calculation of bounds on spectra of Hessian matrices. *SIAM Journal on Scientific Computing*, 30(5):2340–2357, 2008. doi:10.1137/070704186.

[68] R. E. Moore. *Interval Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1966.

[69] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. Other Titles in Applied Mathematics. SIAM, 2009. doi:10.1137/1.9780898717716.

[70] I. P. Nenov, D. H. Fylstra, and L. V. Kolev. Convexity determination in the Microsoft Excel solver using automatic differentiation techniques. Extended abstract, Frontline Systems Inc., 2004. URL `http://www.autodiff.org/ad04/abstracts/Nenov.pdf`.

[71] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2nd edition, 2006. ISSN 978-0-387-30303-1.

[72] I. Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming*, volume 152 of *International Series of Numerical Mathematics*. Birkhäuser Verlag, Basel, 2005. doi:10.1007/3-7643-7374-1.

[73] I. Nowak and S. Vigerske. LaGO: a (heuristic) branch and cut algorithm for nonconvex MINLPs. *Central European Journal of Operations Research*, 16(2):127–138, 2008. doi:10.1007/s10100-007-0051-x.

[74] T. E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3): 10–20, 2007. doi:10.1109/MCSE.2007.58.

[75] J. D. Pintér, editor. *Global Optimization: Scientific and Engineering Case Studies*, volume 85 of *Nonconvex Optimization and Its Applications*. Springer, 2006. doi:10.1007/0-387-30927-6.

[76] J. W. Pratt. Remarks on zeros and ties in the Wilcoxon signed rank procedures. *Journal of the American Statistical Association*, 54(287):655–667, 1959. doi:10.2307/2282543.

[77] I. Quesada and I. E. Grossmann. Global optimization algorithm for heat exchanger networks. *Industrial & Engineering Chemistry Research*, 32(3):487–499, 1993. doi:10.1021/ie00015a012.

[78] I. Quesada and I. E. Grossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6:39–76, 1995. doi:10.1007/BF01106605.

[79] 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.

[80] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996. doi:10.1007/BF00138689.

[81] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS Journal on Computing*, 6:445–454, 1994. doi:10.1287/ijoc.6.4.445.

[82] H. Schichl and A. Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33(4):541–562, 2005. doi:10.1007/s10898-005-0937-x.

[83] E. M. B. Smith and C. C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimization of nonconvex MINLPs. *Computers & Chemical Engineering*, 23(4-5):457–478, 1999. doi:10.1016/S0098-1354(98)00286-5.

[84] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, volume 65 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 2002. ISBN 978-1-4020-1031-6.

[85] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005. doi:10.1007/s10107-005-0581-8.

[86] S. A. Vavasis. Complexity issues in global optimization: A survey. In Horst and Pardalos [54], pages 27–41.

[87] S. Vigerske. *Decomposition of Multistage Stochastic Programs and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD thesis, Humboldt-Universität zu Berlin, 2013. urn:nbn:de:kobv:11-100208240.

[88] X.-H. Vu, H. Schichl, and D. Sam-Haroud. Interval propagation and search on directed acyclic graphs for numerical constraint solving. *Journal of Global Optimization*, 45(4):499–531, 2009. doi:10.1007/s10898-008-9386-7.

[89] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106 (1):25–57, 2006. URL `http://projects.coin-or.org/Ipopt`. doi:10.1007/s10107-004-0559-y.

[90] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi:10.2307/3001968.

[91] J. M. Zamora and I. E. Grossmann. A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. *Journal of Global Optimization*, 14(3):217–249, 1999. doi:10.1023/A:1008312714792.

Supplementary Material

A Test Set

The following table lists the 475 instances of the MINLP test set (see also Section 3.1). For each instance, we list the total number of variables ("variables"), the number of binary variables ("binary"), the number of general integer variables ("integer"), the total number of constraints ("constraints"), the number of quadratic constraints ("quadratic"), and the number of general nonlinear constraints ("nonlinear"). The presented numbers are as they are reported from SCIP after reading an instance from the OSiL input file (and before presolving). Thereby, a constraint is classified as nonlinear if the OSiL file specifies a nonlinear expression for this constraint and is classified as quadratic if no nonlinear expression but quadratic terms have been specified. The remaining constraints are linear (except for instance `meanvarxsc`, where additionally 12 bound-disjunction constraints are created by SCIP to represent the semi-continuity condition on 12 of the variables).

As SCIP can handle only linear objective functions, the OSiL reader replaces a nonlinear function $f(x)$ in the objective by an additional continuous variable z and adds the constraint $f(x) \leq z$ (if minimization) or $f(x) \geq z$ (if maximization).

Name	variables	binary	integer	constraints	quadratic	nonlinear
alan	9	4	0	8	1	0
autocorr_bern20-03	21	20	0	1	1	0
autocorr_bern20-05	21	20	0	1	0	1
autocorr_bern20-10	21	20	0	1	0	1
autocorr_bern20-15	21	20	0	1	0	1
autocorr_bern25-03	26	25	0	1	1	0
autocorr_bern25-06	26	25	0	1	0	1
autocorr_bern30-04	31	30	0	1	0	1
batch0812_nc	77	36	0	206	96	2
batch0812	101	60	0	218	0	2
batchdes	20	9	0	20	0	2
batch_nc	35	12	0	68	30	2
batch	47	24	0	74	0	2
batchs101006m	279	129	0	1020	0	2
batchs121208m	407	203	0	1512	0	2
batchs151208m	446	203	0	1782	0	2
batchs201210m	559	251	0	2328	0	2
blend029	102	36	0	213	12	0
blend480	312	124	0	884	32	0
blend531	272	104	0	736	32	0
blend721	222	87	0	627	24	0
carton7	328	200	56	687	64	0
casctanks	501	40	0	518	157	61
clay0203h	90	18	0	132	0	24
clay0203m	30	18	0	54	24	0
clay0204h	164	32	0	234	0	32
clay0204m	52	32	0	90	32	0
clay0205h	260	50	0	365	0	40
clay0205m	80	50	0	135	40	0

Name	variables	binary	integer	constraints	quadratic	nonlinear
clay0303h	99	21	0	150	0	36
clay0303m	33	21	0	66	36	0
clay0304h	176	36	0	258	0	48
clay0304m	56	36	0	106	48	0
clay0305h	275	55	0	395	0	60
clay0305m	85	55	0	155	60	0
contvar	297	88	0	285	0	120
crudeoil_lee1_05	535	40	0	1240	160	0
crudeoil_lee1_06	642	48	0	1503	192	0
crudeoil_lee1_07	749	56	0	1776	224	0
crudeoil_lee1_08	856	64	0	2059	256	0
crudeoil_lee1_09	963	72	0	2352	288	0
crudeoil_lee1_10	1070	80	0	2655	320	0
crudeoil_lee2_05	1155	70	0	2581	420	0
crudeoil_lee2_06	1386	84	0	3117	504	0
crudeoil_lee2_07	1617	98	0	3670	588	0
crudeoil_lee2_08	1848	112	0	4240	672	0
crudeoil_lee2_09	2079	126	0	4827	756	0
crudeoil_lee2_10	2310	140	0	5431	840	0
crudeoil_lee3_05	1280	70	0	2786	490	0
crudeoil_lee3_06	1536	84	0	3359	588	0
crudeoil_lee3_07	1792	98	0	3949	686	0
crudeoil_lee3_08	2048	112	0	4556	784	0
crudeoil_lee3_09	2304	126	0	5180	882	0
crudeoil_lee3_10	2560	140	0	5821	980	0
crudeoil_lee4_05	1955	95	0	4241	760	0
crudeoil_lee4_06	2346	114	0	5093	912	0
crudeoil_lee4_07	2737	133	0	5965	1064	0
crudeoil_lee4_08	3128	152	0	6857	1216	0
crudeoil_lee4_09	3519	171	0	7769	1368	0
crudeoil_lee4_10	3910	190	0	8701	1520	0
crudeoil_li06	964	132	0	2436	192	0
csched1a	29	15	0	23	0	1
csched1	77	63	0	23	0	1
du-opt5	22	0	13	10	1	0
du-opt	22	0	13	10	1	0
edgecross10-010	91	90	0	481	1	0
edgecross10-020	91	90	0	481	1	0
edgecross10-030	91	90	0	481	1	0
edgecross10-040	91	90	0	481	1	0
edgecross10-050	91	90	0	481	1	0
edgecross10-060	91	44	0	481	1	0
edgecross10-070	91	90	0	481	1	0
edgecross10-080	91	74	0	481	1	0
edgecross10-090	91	90	0	481	1	0
edgecross14-019	183	182	0	1457	1	0
edgecross14-039	183	80	0	1457	1	0
edgecross14-058	183	182	0	1457	1	0
edgecross14-176	183	182	0	1457	1	0
edgecross20-040	381	380	0	4561	1	0
elf	54	24	0	38	27	0
eniplac	141	24	0	189	0	24
enpro48	154	92	0	215	0	2
enpro48pb	154	92	0	215	0	2
enpro56	128	73	0	192	0	2
enpro56pb	128	73	0	192	0	2

Name	variables	binary	integer	constraints	quadratic	nonlinear
ex1221	5	3	0	5	1	1
ex1222	5	1	0	4	1	1
ex1223a	9	4	0	10	5	0
ex1223b	8	4	0	10	4	1
ex1223	12	4	0	14	4	1
ex1224	12	8	0	8	0	4
ex1225	8	6	0	10	0	1
ex1226	5	3	0	5	0	1
ex1243	69	16	0	97	0	1
ex1244	96	23	0	130	0	1
ex1252a	25	3	6	35	9	4
ex1263a	24	4	20	35	4	0
ex1263	92	72	0	55	4	0
ex1264a	24	4	20	35	4	0
ex1264	88	68	0	55	4	0
ex1265a	35	5	30	44	5	0
ex1265	130	100	0	74	5	0
ex1266a	48	6	42	53	6	0
ex1266	180	138	0	95	6	0
ex3	33	8	0	31	0	5
ex3pb	33	8	0	31	0	5
ex4	37	25	0	31	26	0
fac1	23	6	0	19	0	1
fac3	67	12	0	34	1	0
feedtray2	88	36	0	284	147	0
fin2bb	588	175	0	618	0	21
flay02h	46	4	0	51	0	2
flay02m	14	4	0	11	0	2
flay03h	122	12	0	144	0	3
flay03m	26	12	0	24	0	3
flay04h	234	24	0	282	0	4
flay04m	42	24	0	42	0	4
flay05h	382	40	0	465	0	5
flay05m	62	40	0	65	0	5
flay06m	86	60	0	93	0	6
fo7_2	114	42	0	211	0	14
fo7_ar2_1	112	0	42	269	0	14
fo7_ar25_1	112	0	42	269	0	14
fo7_ar3_1	112	0	42	269	0	14
fo7_ar4_1	112	0	42	269	0	14
fo7_ar5_1	112	0	42	269	0	14
fo7	114	42	0	211	0	14
fo8_ar2_1	144	0	56	347	0	16
fo8_ar25_1	144	0	56	347	0	16
fo8_ar3_1	144	0	56	347	0	16
fo8_ar4_1	144	0	56	347	0	16
fo8_ar5_1	144	0	56	347	0	16
fo8	146	56	0	273	0	16
fo9_ar2_1	180	0	72	435	0	18
fo9_ar25_1	180	0	72	435	0	18
fo9_ar3_1	180	0	72	435	0	18
fo9_ar4_1	180	0	72	435	0	18
fo9_ar5_1	180	0	72	435	0	18
fo9	182	72	0	343	0	18
fuel	17	3	0	16	4	0
gasprod_sarawak01	132	38	0	212	34	0

Name	variables	binary	integer	constraints	quadratic	nonlinear
gastrans	106	21	0	149	3	21
gbd	5	3	0	5	1	0
gear2	29	24	0	5	0	1
gear3	9	0	4	5	0	1
gear4	6	0	4	1	0	1
gear	5	0	4	1	0	1
genpooling_lee1	49	9	0	82	20	0
genpooling_lee2	53	9	0	92	30	0
ghg_1veh	30	12	0	38	6	22
gkocis	11	3	0	8	0	2
graphpart_2g-0044-1601	49	48	0	17	1	0
graphpart_2g-0055-0062	76	75	0	26	1	0
graphpart_2g-0066-0066	109	108	0	37	1	0
graphpart_2g-0077-0077	148	147	0	50	1	0
graphpart_2pm-0044-0044	49	48	0	17	1	0
graphpart_2pm-0055-0055	76	75	0	26	1	0
graphpart_2pm-0066-0066	109	108	0	37	1	0
graphpart_3g-0234-0234	73	72	0	25	1	0
graphpart_3g-0244-0244	97	96	0	33	1	0
graphpart_3g-0333-0333	82	81	0	28	1	0
graphpart_3g-0334-0334	109	108	0	37	1	0
graphpart_3g-0344-0344	145	144	0	49	1	0
graphpart_3pm-0234-0234	73	72	0	25	1	0
graphpart_3pm-0244-0244	97	96	0	33	1	0
graphpart_3pm-0333-0333	82	81	0	28	1	0
graphpart_3pm-0334-0334	109	108	0	37	1	0
graphpart_clique-20	61	60	0	21	1	0
heatexch_spec3	261	60	0	251	0	1
heatexch_trigen	291	45	0	262	0	1
hmittelman	17	16	0	8	0	8
hybriddynamic_fixed	73	10	0	80	1	0
hybriddynamic_var	82	10	0	101	20	1
jit1	26	0	4	33	0	1
johnall	195	190	0	193	0	191
kport20	101	7	33	27	0	20
lip	61	52	0	84	0	1
m3	26	6	0	43	0	6
m6	86	30	0	157	0	12
m7_ar2_1	112	0	42	269	0	14
m7_ar25_1	112	0	42	269	0	14
m7_ar3_1	112	0	42	269	0	14
m7_ar4_1	112	0	42	269	0	14
m7_ar5_1	112	0	42	269	0	14
m7	114	42	0	211	0	14
meanvarx	36	14	0	45	1	0
meanvarxsc	36	14	0	43	1	0
minlphix	85	20	0	93	0	5
multiplants_mtg2	230	112	0	307	36	1
multiplants_mtg5	191	78	0	309	48	1
netmod_dol1	1999	462	0	3138	1	0
netmod_dol2	1999	462	0	3081	1	0
netmod_kar1	457	136	0	667	1	0
netmod_kar2	457	136	0	667	1	0
no7_ar2_1	112	0	42	269	0	14
no7_ar25_1	112	0	42	269	0	14
no7_ar3_1	112	0	42	269	0	14

Name	variables	binary	integer	constraints	quadratic	nonlinear
no7_ar4_1	112	0	42	269	0	14
no7_ar5_1	112	0	42	269	0	14
nous1	51	2	0	44	29	0
nous2	51	2	0	44	29	0
nvs01	4	0	2	4	0	3
nvs02	10	0	5	4	4	0
nvs03	4	0	2	3	2	0
nvs04	3	0	2	1	0	1
nvs05	9	0	2	10	0	9
nvs06	4	0	2	1	0	1
nvs07	4	0	3	3	1	1
nvs08	5	0	2	4	2	2
nvs09	11	0	10	1	0	1
nvs10	3	0	2	3	3	0
nvs11	4	0	3	4	4	0
nvs12	5	0	4	5	5	0
nvs13	6	0	5	6	6	0
nvs14	10	0	5	4	4	0
nvs15	5	0	3	2	1	0
nvs16	3	0	2	1	0	1
nvs17	8	0	7	8	8	0
nvs18	7	0	6	7	7	0
nvs19	9	0	8	9	9	0
nvs20	17	0	5	9	0	1
nvs21	4	0	2	3	0	3
nvs22	9	0	4	10	0	9
nvs23	10	0	9	10	10	0
nvs24	11	0	10	11	11	0
o7_2	114	42	0	211	0	14
o7_ar2_1	112	0	42	269	0	14
o7_ar25_1	112	0	42	269	0	14
o7_ar3_1	112	0	42	269	0	14
o7_ar4_1	112	0	42	269	0	14
o7_ar5_1	112	0	42	269	0	14
o7	114	42	0	211	0	14
oaer	9	3	0	7	0	2
oil2	936	2	0	926	2	282
oil	1535	19	0	1546	24	394
ortez	87	18	0	74	21	6
parallel	206	25	0	116	0	5
pooling_epa1	215	30	0	341	32	18
portfol_classical050_1	150	50	0	103	1	0
portfol_robust050_34	203	51	0	156	2	0
portfol_robust100_09	403	101	0	306	2	0
portfol_shortfall050_68	204	51	0	157	2	0
prob02	6	0	6	8	5	0
prob03	2	0	2	1	1	0
procsel	10	3	0	7	0	2
product2	2842	128	0	3125	528	0
product	1553	107	0	1925	132	0
ravem	113	54	0	187	0	2
ravempb	113	54	0	187	0	2
risk2b	464	14	0	581	0	1
risk2bpb	464	14	0	581	0	1
rsyn0805h	308	37	0	429	0	3
rsyn0805m02h	700	148	0	1045	0	6

Name	variables	binary	integer	constraints	quadratic	nonlinear
rsyn0805m02m	360	148	0	769	0	6
rsyn0805m03h	1050	222	0	1698	0	9
rsyn0805m03m	540	222	0	1284	0	9
rsyn0805m04h	1400	296	0	2438	0	12
rsyn0805m04m	720	296	0	1886	0	12
rsyn0805m	170	69	0	286	0	3
rsyn0810h	343	42	0	483	0	6
rsyn0810m02h	790	168	0	1188	0	12
rsyn0810m02m	410	168	0	866	0	12
rsyn0810m03h	1185	252	0	1935	0	18
rsyn0810m03m	615	252	0	1452	0	18
rsyn0810m04h	1580	336	0	2784	0	24
rsyn0810m04m	820	336	0	2140	0	24
rsyn0810m	185	74	0	312	0	6
rsyn0815h	387	47	0	552	0	11
rsyn0815m02h	898	188	0	1361	0	22
rsyn0815m02m	470	188	0	981	0	22
rsyn0815m03h	1347	282	0	2217	0	33
rsyn0815m03m	705	282	0	1647	0	33
rsyn0815m04m	940	376	0	2430	0	44
rsyn0815m	205	79	0	347	0	11
rsyn0820h	417	52	0	604	0	14
rsyn0820m02h	978	208	0	1500	0	28
rsyn0820m02m	510	208	0	1074	0	28
rsyn0820m03m	765	312	0	1809	0	42
rsyn0820m	215	84	0	371	0	14
rsyn0830h	494	62	0	716	0	20
rsyn0830m02h	1172	248	0	1794	0	40
rsyn0830m02m	620	248	0	1272	0	40
rsyn0830m03h	1758	372	0	2934	0	60
rsyn0830m	250	94	0	425	0	20
rsyn0840h	568	72	0	837	0	28
rsyn0840m02h	1360	288	0	2106	0	56
rsyn0840m02m	720	288	0	1480	0	56
rsyn0840m	280	104	0	484	0	28
sep1	29	2	0	31	6	0
sepasequ_convent	641	20	0	1128	81	140
slay04h	142	24	0	175	1	0
slay04m	46	24	0	55	1	0
slay05h	232	40	0	291	1	0
slay05m	72	40	0	91	1	0
slay06h	344	60	0	436	1	0
slay06m	104	60	0	136	1	0
slay07h	478	84	0	610	1	0
slay07m	142	84	0	190	1	0
slay08h	634	112	0	813	1	0
slay08m	186	112	0	253	1	0
slay09h	812	144	0	1045	1	0
slay09m	236	144	0	325	1	0
slay10h	1012	180	0	1306	1	0
slay10m	292	180	0	406	1	0
smallinvDAXr1b010-011	31	0	30	4	1	0
smallinvDAXr1b020-022	31	0	30	4	1	0
smallinvDAXr1b050-055	31	0	30	4	1	0
smallinvDAXr1b100-110	31	0	30	4	1	0
smallinvDAXr1b150-165	31	0	30	4	1	0

Name	variables	binary	integer	constraints	quadratic	nonlinear
smallinvDAXr1b200-220	31	0	30	4	1	0
smallinvDAXr2b010-011	31	0	30	4	1	0
smallinvDAXr2b020-022	31	0	30	4	1	0
smallinvDAXr2b050-055	31	0	30	4	1	0
smallinvDAXr2b100-110	31	0	30	4	1	0
smallinvDAXr2b150-165	31	0	30	4	1	0
smallinvDAXr2b200-220	31	0	30	4	1	0
smallinvDAXr3b010-011	31	0	30	4	1	0
smallinvDAXr3b020-022	31	0	30	4	1	0
smallinvDAXr3b050-055	31	0	30	4	1	0
smallinvDAXr3b100-110	31	0	30	4	1	0
smallinvDAXr3b150-165	31	0	30	4	1	0
smallinvDAXr4b010-011	31	0	30	4	1	0
smallinvDAXr4b020-022	31	0	30	4	1	0
smallinvDAXr4b050-055	31	0	30	4	1	0
smallinvDAXr4b100-110	31	0	30	4	1	0
smallinvDAXr4b150-165	31	0	30	4	1	0
smallinvDAXr5b010-011	31	0	30	4	1	0
smallinvDAXr5b020-022	31	0	30	4	1	0
smallinvDAXr5b050-055	31	0	30	4	1	0
smallinvDAXr5b100-110	31	0	30	4	1	0
smallinvDAXr5b150-165	31	0	30	4	1	0
smallinvDAXr5b200-220	31	0	30	4	1	0
smallinvSNPr1b010-011	101	0	100	4	1	0
smallinvSNPr1b020-022	101	0	100	4	1	0
smallinvSNPr1b050-055	101	0	100	4	1	0
smallinvSNPr2b010-011	101	0	100	4	1	0
smallinvSNPr2b020-022	101	0	100	4	1	0
smallinvSNPr2b050-055	101	0	100	4	1	0
smallinvSNPr3b010-011	101	0	100	4	1	0
smallinvSNPr3b020-022	101	0	100	4	1	0
smallinvSNPr3b050-055	101	0	100	4	1	0
smallinvSNPr3b100-110	101	0	100	4	1	0
smallinvSNPr4b010-011	101	0	100	4	1	0
smallinvSNPr4b020-022	101	0	100	4	1	0
smallinvSNPr4b050-055	101	0	100	4	1	0
smallinvSNPr4b100-110	101	0	100	4	1	0
smallinvSNPr4b150-165	101	0	100	4	1	0
smallinvSNPr4b200-220	101	0	100	4	1	0
smallinvSNPr5b010-011	101	0	100	4	1	0
smallinvSNPr5b020-022	101	0	100	4	1	0
smallinvSNPr5b050-055	101	0	100	4	1	0
smallinvSNPr5b100-110	101	0	100	4	1	0
smallinvSNPr5b150-165	101	0	100	4	1	0
smallinvSNPr5b200-220	101	0	100	4	1	0
spectra2	69	30	0	72	8	0
sporttournament06	16	15	0	1	1	0
sporttournament08	29	28	0	1	1	0
sporttournament10	46	45	0	1	1	0
sporttournament12	67	66	0	1	1	0
sporttournament16	121	120	0	1	1	0
spring	18	11	1	9	1	5
squfl010-025	261	10	0	276	1	0
squfl010-025persp	510	10	0	525	250	0
squfl010-040persp	810	10	0	840	400	0
squfl010-080persp	1610	10	0	1680	800	0

Name	variables	binary	integer	constraints	quadratic	nonlinear
squfl015-060persp	1815	15	0	1860	900	0
squfl015-080persp	2415	15	0	2480	1200	0
squfl020-040persp	1620	20	0	1640	800	0
squfl020-050persp	2020	20	0	2050	1000	0
squfl020-150persp	6020	20	0	6150	3000	0
squfl025-025persp	1275	25	0	1275	625	0
squfl025-030persp	1525	25	0	1530	750	0
squfl025-040persp	2025	25	0	2040	1000	0
sssd08-04	60	44	0	40	0	12
sssd08-04persp	60	44	0	40	12	0
st_e13	2	1	0	2	1	0
st_e14	12	4	0	14	4	1
st_e15	5	3	0	5	1	1
st_e27	6	2	0	7	1	0
st_e29	12	8	0	8	0	4
st_e31	112	24	0	135	5	0
st_e32	36	1	18	19	2	11
st_e35	33	7	0	40	0	1
st_e36	3	0	1	3	0	3
st_e38	5	0	2	4	0	2
st_e40	4	0	3	8	1	3
st_miqp1	6	5	0	2	1	0
st_miqp2	5	2	2	4	1	0
st_miqp3	3	0	2	2	1	0
st_miqp4	7	3	0	5	1	0
st_miqp5	8	2	0	14	1	0
stockcycle	481	432	0	98	0	1
st_test1	6	5	0	2	1	0
st_test2	7	5	1	3	1	0
st_test3	14	10	3	11	1	0
st_test4	7	2	4	6	1	0
st_test5	11	10	0	12	1	0
st_test6	11	10	0	6	1	0
st_test8	25	0	24	21	1	0
st_testgr1	11	0	10	6	1	0
st_testgr3	21	0	20	21	1	0
st_testph4	4	0	3	11	1	0
supplychain	27	3	0	30	6	0
supplychainp1_020306	151	27	0	256	0	1
supplychainp1_022020	2941	460	0	5301	0	1
supplychainp1_030510	446	70	0	836	0	1
supplychainr1_020306	94	27	0	115	0	1
supplychainr1_022020	1441	460	0	1841	0	1
supplychainr1_030510	231	70	0	281	0	1
syn05h	42	5	0	58	0	3
syn05m02h	104	20	0	151	0	6
syn05m02m	60	20	0	101	0	6
syn05m03h	156	30	0	249	0	9
syn05m03m	90	30	0	174	0	9
syn05m04h	208	40	0	362	0	12
syn05m04m	120	40	0	262	0	12
syn05m	20	5	0	28	0	3
syn10h	77	10	0	112	0	6
syn10m02h	194	40	0	294	0	12
syn10m02m	110	40	0	198	0	12
syn10m03h	291	60	0	486	0	18

Name	variables	binary	integer	constraints	quadratic	nonlinear
syn10m03m	165	60	0	342	0	18
syn10m04h	388	80	0	708	0	24
syn10m04m	220	80	0	516	0	24
syn10m	35	10	0	54	0	6
syn15h	121	15	0	181	0	11
syn15m02h	302	60	0	467	0	22
syn15m02m	170	60	0	313	0	22
syn15m03h	453	90	0	768	0	33
syn15m03m	255	90	0	537	0	33
syn15m04h	604	120	0	1114	0	44
syn15m04m	340	120	0	806	0	44
syn15m	55	15	0	89	0	11
syn20h	151	20	0	233	0	14
syn20m02h	382	80	0	606	0	28
syn20m02m	210	80	0	406	0	28
syn20m03h	573	120	0	999	0	42
syn20m03m	315	120	0	699	0	42
syn20m04h	764	160	0	1452	0	56
syn20m04m	420	160	0	1052	0	56
syn20m	65	20	0	113	0	14
syn30h	228	30	0	345	0	20
syn30m02h	576	120	0	900	0	40
syn30m02m	320	120	0	604	0	40
syn30m03m	480	180	0	1041	0	60
syn30m04h	1152	240	0	2160	0	80
syn30m04m	640	240	0	1568	0	80
syn30m	100	30	0	167	0	20
syn40h	302	40	0	466	0	28
syn40m02h	764	160	0	1212	0	56
syn40m02m	420	160	0	812	0	56
syn40m03m	630	240	0	1398	0	84
syn40m04h	1528	320	0	2904	0	112
syn40m04m	840	320	0	2104	0	112
syn40m	130	40	0	226	0	28
synthes1	8	3	0	7	0	3
synthes2	13	5	0	15	0	4
synthes3	19	8	0	24	0	5
tanksize	47	9	0	74	20	1
tln2	8	2	6	12	2	0
tln4	24	4	20	24	4	0
tln5	35	5	30	30	5	0
tloss	48	6	42	53	6	0
tls12	812	656	12	384	0	12
tls2	37	31	2	24	0	2
tls4	105	85	4	64	0	4
tltr	48	12	36	54	3	0
tspn05	21	10	0	11	5	1
unitcommit1	961	720	0	5330	1	0
util	145	28	0	167	4	0
watercontamination0202	106713	7	0	107210	1	0
watercontamination0202r	196	7	0	284	1	0
waternd1	75	20	0	84	16	1
watertreatnd_conc	360	5	0	319	24	5
watertreatnd_flow	420	5	0	379	150	5