Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TIMO BERTHOLD
STEFAN HEINZ
STEFAN VIGERSKE*

Extending a CIP framework to solve MIQCPs

Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.
* Humboldt Universität zu Berlin, 10099 Berlin, Germany

ZIB-Report 09-23 (July 2009; revised May 2010)

EXTENDING A CIP FRAMEWORK TO SOLVE MIQCPS*

TIMO BERTHOLD†, STEFAN HEINZ†, AND STEFAN VIGERSKE‡

Abstract. This paper discusses how to build a solver for mixed integer quadratically constrained programs (MIQCPs) by extending a framework for constraint integer programming (CIP). The advantage of this approach is that we can utilize the full power of advanced MILP and CP technologies, in particular for the linear relaxation and the discrete components of the problem. We use an outer approximation generated by linearization of convex constraints and linear underestimation of nonconvex constraints to relax the problem. Further, we give an overview of the reformulation, separation, and propagation techniques that are used to handle the quadratic constraints efficiently.

We implemented these methods in the branch-cut-and-price framework SCIP. Computational experiments indicating the potential of the approach and evaluating the impact of the algorithmic components are provided.

Key words. mixed integer quadratically constrained programming, constraint integer programming, branch-and-cut, convex relaxation, domain propagation, primal heuristic, nonconvex

AMS(MOS) subject classifications. 90C11, 90C20, 90C26, 90C27, 90C57.

1. Introduction.

In recent years, substantial progress has been made in the solvability of generic *mixed integer linear programs (MILPs)* [2, 12]. Furthermore, it has been shown that successful MILP solving techniques can often be extended to the more general case of *mixed integer nonlinear programs (MINLPs)* [1, 6, 13]. Analogously, several authors have shown that an integrated approach of *constraint programming (CP)* and MILP can help to solve optimization problems that were intractable with either of the two methods alone, for an overview see [17].

The paradigm of *constraint integer programming (CIP)* [2, 4] combines modeling and solving techniques from the fields of constraint programming (CP), mixed integer programming, and *satisfiability testing (SAT)*. The concept of CIP aims at restricting the generality of CP modeling as little as needed while still retaining the full performance of MILP solving techniques. Such a paradigm allows us to address a wide range of optimization problems. For example, in [2], it is shown that CIP includes MILP and constraint programming over finite domains as special cases.

The goal of this paper is to show, how a framework for CIPs can be extended towards a competitive solver for *mixed integer quadratically constrained programs (MIQCPs)*, which are an important subclass of MINLPs. This framework allows us to utilize the power of already existing MILP and

*Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin, http://www.matheon.de.

†Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, {berthold,heinz}@zib.de

‡Humboldt University Berlin, Unter den Linden 6, 10099 Berlin, Germany, stefan@math.hu-berlin.de

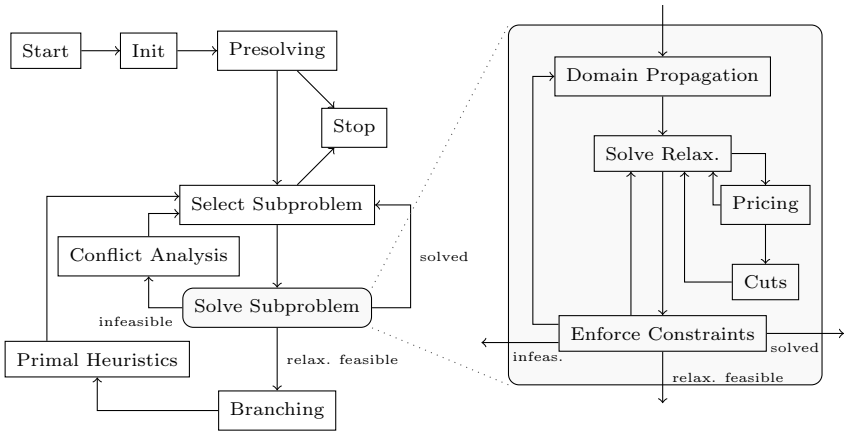

FIG. 1. *Flowchart of the main solving loop of SCIP*

CP technologies for handling the linear and the discrete parts of the problem. The integration of MIQCP is a first step towards the incorporation of MINLP into the concept of constraint integer programming.

We extended the branch-cut-and-price framework SCIP (Solving Constraint Integer Programs) [2, 3] by adding methods for MIQCP. SCIP incorporates the idea of CIP and implements several state-of-the-art techniques for solving MILPs. Due to its plugin-based design, it can be easily customized, e.g., by adding problem specific separation, presolving, or domain propagation algorithms.

The framework SCIP solves CIPs by a branch-and-bound algorithm. The problem is recursively split into smaller subproblems, thereby creating a search tree and implicitly enumerating all potential solutions. At each subproblem, domain propagation is performed to exclude further values from the variables' domains, and a relaxation may be solved to achieve a local lower bound – assuming the problem is to minimize the objective function. The relaxation may be strengthened by adding further valid constraints (e.g., linear inequalities), which cut off the optimal solution of the relaxation. In case a subproblem is found to be infeasible, conflict analysis is performed to learn additional valid constraints. Primal heuristics are used as supplementary methods to improve the upper bound. Figure 1 illustrates the main algorithmic components of SCIP. In the context of this article, the relaxation employed in SCIP is a linear program (LP).

The remainder of this article is organized as follows. In Section 2, we formally define MIQCP and CIP, in Sections 3, 4, and 5, we show how to handle quadratic constraints inside SCIP, and in Section 6, we present computational results.

2. Problem definition. An MIQCP is an optimization problem of the form

$$\begin{aligned} \min \quad & d^T x && \\ \text{s.t.} \quad & x^T A_i x + {b_i}^T x + c_i \leq 0 && \text{for } i = 1, \ldots, m \\ & x_k^L \leq x_k \leq x_k^U && \text{for all } k \in N \\ & x_k \in \mathbb{Z} && \text{for all } k \in I, \end{aligned} \tag{2.1}$$

where $I \subseteq N \coloneqq \{1, \ldots, n\}$ is the index set of the integer variables, $d \in \mathbb{Q}^n$, $A_i \in \mathbb{Q}^{n \times n}$ and symmetric, $b_i \in \mathbb{Q}^n$, $c_i \in \mathbb{Q}$ for $i = 1, \ldots, m$, $x^L \in \overline{\mathbb{Q}}^n$ and $x^U \in \overline{\mathbb{Q}}^n$, with $\overline{\mathbb{Q}} \coloneqq \mathbb{Q} \cup \{\pm\infty\}$, are the lower and upper bounds of the variables x, respectively ($\mathbb{Q}$ denotes the rational numbers). Note that we do not require the matrices A_i to be positive semidefinite, hence we also allow for nonconvex quadratic constraints. If $I = \emptyset$, we call (2.1) a *quadratically constrained program (QCP)*.

The definition of CIP, as given in [2, 4], requires a linear objective function. This is, however, just a technical prerequisite, as a quadratic (or more general) objective $f(x)$ can be modeled by introducing an auxiliary objective variable z that is linked to the actual nonlinear objective function with a constraint $f(x) \leq z$. Thus, formulation (2.1) also covers the general case of mixed integer quadratically constrained quadratic problems.

In this article, we use a definition of CIP which is slightly different from the one given in [2, 4]. A constraint integer program consists of solving

$$\begin{aligned} \min \quad & d^T x && \\ \text{s.t.} \quad & \mathcal{C}_i(x) = 1 && \text{for } i = 1, \ldots, m \\ & x_k \in \mathbb{Z} && \text{for all } k \in I, \end{aligned}$$

with a finite set of constraints $\mathcal{C}_i : \mathbb{Q}^n \to \{0, 1\}$, for $i = 1, \ldots, m$, the index set $I \subseteq N$ of the integer variables, and an objective function vector $d \in \mathbb{Q}^n$.

In [2, 4], it is required that the subproblem remaining after fixing all integer variables be a linear program in order to guarantee termination in finite time. In this article, we, however, require a subproblem with all integer variables fixed to be a QCP. Note that, using *spatial* branch-and-bound algorithms, QCPs with finite bounds on the variables can be solved in finite time up to a given tolerance [18].

3. A constraint handler for quadratic constraints. In SCIP, a constraint handler defines the semantics and the algorithms to process constraints of a certain class. A single constraint handler is responsible for all the constraints belonging to its constraint class. Each constraint handler has to implement an enforcement method. In enforcement, the handler has to decide whether a given solution, e.g., the optimum of a relaxation[1], sat-

[1]For this section, we assume that the LP relaxation is bounded. In our implementation, the so-called pseudo solution, see [2, 3] for details, will be used in the case of unbounded LP relaxations.

isfies all of its constraints. If the solution violates one or more constraints, the handler may resolve the infeasibility by adding another constraint, performing a domain reduction, or a branching.

For speeding up computation, a constraint handler may further implement additional features like presolving, cutting plane separation, and domain propagation for its particular class of constraints. Besides that, a constraint handler can add valid linear inequalities to the initial LP relaxation. For example, all constraint handler for (general or specialized) linear constraints add their constraints to the initial LP relaxation. The constraint handler for quadratic constraints adds one linear inequality that is obtained by the method given in Section 3.2 below.

In the following, we discuss the presolving, separation, propagation, and enforcement algorithms that are used to handle quadratic constraints.

3.1. Presolving. During the presolving phase, a set of reformulations and simplifications are tried. If SCIP fixes or aggregates variables, e.g., using global presolving methods like dual bound reduction [2], then the corresponding reformulations will also be realized in the quadratic constraints. Bounds on the variables are tightened using the domain propagation method described in Section 3.3. If, due to reformulations, the quadratic part of a constraint vanishes, it is replaced by the corresponding linear constraint. Furthermore, the following reformulations are performed.

Binary Variables. A square of a binary variable is replaced by the binary variable itself. Further, if a constraint contains a product of a binary variable with a linear term, i.e., $x \sum_{i=1}^{k} a_i y_i$, where x is a binary variable, y_i are variables with finite bounds, and $a_i \in \mathbb{Q}$, $i = 1, \ldots, k$, then this product will be replaced by a new variable $z \in \mathbb{R}$ and the linear constraints

$$\begin{aligned} y^L x &\leq z \leq y^U x \\ \sum_{i=1}^{k} a_i y_i - y^U(1-x) &\leq z \leq \sum_{i=1}^{k} a_i y_i - y^L(1-x), \text{ where} \\ y^L &:= \sum_{\substack{i=1,\\ a_i>0}}^{k} a_i y_i^L + \sum_{\substack{i=1,\\ a_i<0}}^{k} a_i y_i^U, \text{ and} \\ y^U &:= \sum_{\substack{i=1,\\ a_i>0}}^{k} a_i y_i^U + \sum_{\substack{i=1,\\ a_i<0}}^{k} a_i y_i^L. \end{aligned} \tag{3.1}$$

In the case that $k = 1$ and y_1 is also a binary variable, the product xy_1 can also be handled by SCIP's handler for AND constraints [11].

Second-Order Cone (SOC) Constraints. Constraints of the form

$$\gamma + \sum_{i=1}^{k} (\alpha_i (x_i + \beta_i))^2 \leq (\alpha_0 (y + \beta_0))^2, \tag{3.2}$$

with $k \geq 2$, $\alpha_i, \beta_i \in \mathbb{Q}$, $i = 0, \ldots, k$, $\gamma \in \mathbb{Q}_+$, and $y^L \geq -\beta_0$ are recognized as SOC constraints and handled by a specialized constraint handler, cf. Section 4.

Convexity. After the presolving phase, each quadratic function is checked for convexity by computing the sign of the minimum eigenvalue of the coefficient matrix A. This information will be used for separation.

3.2. Separation. If the current LP solution $\tilde{x}$ violates some constraints, a constraint handler may add valid cutting planes in order to strengthen the formulation.

For a violated convex constraint i, this is always possible by linearizing the constraint function at $\tilde{x}$. Thus, we add the valid inequality

$$c_i - \tilde{x}^T A_i \tilde{x} + (b_i^T + 2\tilde{x}^T A_i)x \leq 0 \tag{3.3}$$

to separate $\tilde{x}$. In the important special case that $x^T A_i x \equiv a x_j^2$ for some $a > 0$ and $j \in I$ with $\tilde{x}_j \notin \mathbb{Z}$, we generate the cut

$$c_i + b_i^T x + a(2\lfloor \tilde{x}_j \rfloor + 1)x_j - a\lfloor \tilde{x}_j \rfloor \lceil \tilde{x}_j \rceil \leq 0, \tag{3.4}$$

which is obtained by underestimating $x_j \in \mathbb{Z} \mapsto x_j^2$ by the secant defined by the points $(\lfloor \tilde{x}_j \rfloor, \lfloor \tilde{x}_j \rfloor^2)$ and $(\lceil \tilde{x}_j \rceil, \lceil \tilde{x}_j \rceil^2)$. Note that the violation of (3.4) by $\tilde{x}$ is larger than that of (3.3).

For a violated nonconvex constraint i, we currently underestimate each term of $x^T A_i x$ separately. A term $a x_j^2$ with $a > 0$, $j \in N$, is underestimated as just discussed. For the case $a < 0$, however, the tightest linear underestimation for the term $a x_j^2$ is given by the secant approximation $a(x_j^L + x_j^U)x_j - a x_j^L x_j^U$, if x_j^L and x_j^U are finite. Otherwise, if $x_j^L = -\infty$ or $x_j^U = \infty$, we skip separation for constraint i. For a bilinear term $a x_j x_k$ with $a > 0$, we utilize the McCormick underestimators [21]

$$\begin{aligned} a x_j x_k &\geq a x_j^L x_k + a x_k^L x_j - a x_j^L x_k^L, \\ a x_j x_k &\geq a x_j^U x_k + a x_k^U x_j - a x_j^U x_k^U. \end{aligned}$$

If $(x_j^U - x_j^L)\tilde{x}_k + (x_k^U - x_k^L)\tilde{x}_j \leq x_j^U x_k^U - x_j^L x_k^L$ and the bounds x_j^L and x_k^L are finite, the former is used for cut generation, elsewise the latter is used. If both x_j^L or x_k^L and x_j^U or x_k^U are infinite, we skip separation for constraint i. Similar, for a bilinear term $a x_j x_k$ with $a < 0$, the McCormick underestimators are

$$\begin{aligned} a x_j x_k &\geq a x_j^U x_k + a x_k^L x_j - a x_j^U x_k^L, \\ a x_j x_k &\geq a x_j^L x_k + a x_k^U x_j - a x_j^L x_k^U. \end{aligned}$$

If $(x_j^U - x_j^L)\tilde{x}_k - (x_k^U - x_k^L)\tilde{x}_j \leq x_j^U x_k^L - x_j^L x_k^U$ and the bounds x_j^U and x_k^L are finite, the former is used for cut generation, elsewise the latter is used.

In the case that a linear inequality generated by this method does not cut off the current LP solution $\tilde{x}$, the infeasibility has to be resolved in enforcement, see Section 3.4. Besides others, the enforcement method may apply a spatial branching operation on a variable x_j, creating two subproblems, which both contain a strictly smaller domain for x_j. This results in tighter linear underestimators.

3.3. Propagation. In the domain propagation call, a constraint handler may deduce new restrictions upon local domains of variables. Such deductions may yield stronger linear underestimators in the separation procedures, prune nodes due to infeasibility of a constraint, or result in further deductions for other constraints. For quadratic constraints, we implemented an interval-arithmetic based method similar to [16]. To allow for an efficient propagation, we write a quadratic constraint in the form

$$\sum_{j\in J} d_j x_j + \sum_{k\in K} \left(e_k + p_{k,k}x_k + \sum_{r\in K} p_{k,r}x_r \right) x_k \in [\ell, u], \tag{3.5}$$

such that $d_j, e_k, p_{k,r} \in \mathbb{Q}$, $\ell, u \in \overline{\mathbb{Q}}$, $J \cup K \subseteq N$, $J \cap K = \emptyset$, and $p_{k,r} = 0$ for $k > r$. For a given $a \in \mathbb{Q}$, an interval $[b^L, b^U]$, and a variable y with domain $[y^L, y^U]$, we denote by $q(a, [b^L, b^U], y)$ the set $\{by + ay^2 : y \in [y^L, y^U], b \in [b^L, b^U]\}$. This set can be computed analytically [16].

The forward propagation step aims at tightening the bounds $[\ell, u]$ in (3.5). For this purpose, we replace the variables x_j and x_r in (3.5) by their domain to obtain the "interval-equation"

$$\sum_{j\in J} d_j [x_j^L, x_j^U] + \sum_{k\in K} ([f_k^L, f_k^U] x_k + p_{k,k} x_k^2) \in [\ell, u],$$

where $[f_k^L, f_k^U] \coloneqq [e_k, e_k] + \sum_{r\in K} p_{k,r}[x_r^L, x_r^U]$. Computing $[h^L, h^U] \coloneqq \sum_{j\in J} d_j [x_j^L, x_j^U] + \sum_{k\in K} q(p_{k,k}, [f_k^L, f_k^U], x_k)$ yields an interval that contains all values that the left hand side of (3.5) can take w.r.t. the current variables' domains. If $[h^L, h^U] \cap [\ell, u] = \emptyset$, then (3.5) cannot be satisfied for any $x \in [x^L, x^U]$ and the current branch-and-bound node can be pruned. Otherwise, the interval $[\ell, u]$ can be tightened to $[\ell, u] \cap [h^L, h^U]$.

The backward propagation step aims at inferring domain deductions on the variables in (3.5) using the interval $[\ell, u]$. For a "linear" variable x_j, $j \in J$, we can easily infer the bounds

$$\frac{1}{d_j} \left([\ell, u] - \sum_{j'\in J, j\neq j'} d_{j'} [x_{j'}^L, x_{j'}^U] - \sum_{k\in K} q(p_{k,k}, [f_k^L, f_k^U], x_k) \right).$$

For a "quadratic" variable x_k, $k \in K$, one way to compute potentially

tighter bounds is by solving the quadratic interval-equation

$$\sum_{j\in J} d_j[x_j^L, x_j^U] + \sum_{k'\in K, k'\neq k} q(p_{k',k'}, [e_{k'}, e_{k'}] + \sum_{r\in K, r\neq k'} p_{k,r}[x_r^L, x_r^U], x_{k'}) + ([e_k, e_k] + \sum_{r\in K}(p_{k,r} + p_{r,k})[x_r^L, x_r^U])x_k + p_{k,k}x_k^2 \in [\ell, u].$$

However, since evaluating the argument of $q(\cdot)$ for each $k \in K$ may produce a huge computational overhead, especially for constraints with many bilinear terms, we compute the solution set of

$$\sum_{j\in J} d_j[x_j^L, x_j^U] + \sum_{\substack{k'\in K\\ k'\neq k}} \left(q(p_{k',k'}, [e_{k'}, e_{k'}], x_{k'}) + \sum_{\substack{r\in K\\ r\neq k'}} p_{k',r}[x_{k'}^L, x_{k'}^U][x_r^L, x_r^U] \right) + ([e_k, e_k] + \sum_{r\in K}(p_{k,r} + p_{r,k})[x_r^L, x_r^U])x_k + p_{k,k}x_k^2 \in [\ell, u], \quad (3.6)$$

which can be performed more efficiently. If the intersection of the current domain $[x_k^L, x_k^U]$ of x_k with the solution set of (3.6) is empty, we can deduce infeasibility and prune the corresponding node. Otherwise, we may be able to tighten the bounds of x_k.

As in [16], all interval operations detailed in this section are performed in outward rounding mode.

3.4. Enforcement. In the enforcement call, a constraint handler has to check whether the current LP solution $\tilde{x}$ is feasible for all its constraints. It can resolve an infeasibility by either adding cutting planes that separate $\tilde{x}$ from the relaxation, by tightening bounds on a variable such that $\tilde{x}$ is separated from the current domain, by pruning the current node from the branch-and-bound tree, or by performing a branching operation.

We have configured SCIP to call the enforcement method of the quadratic constraint handler with a lower priority than the enforcement method for the handler of integrality constraints. Thus, at the point where quadratic constraints are enforced, all integer variables take an integral value in the LP optimum $\tilde{x}$. For a violated quadratic constraint, we first perform a forward propagation step, see Section 3.3, which may prune the current node. If the forward propagation does not declare infeasibility, we call the separation method, see Section 3.2. If the separator fails to cut off $\tilde{x}$, we perform a spatial branching operation. We use the following branching rule to resolve infeasibility in a nonconvex quadratic constraint.

Branching Rule. We consider each unfixed variable x_j that appears in a violated nonconvex quadratic constraint as a branching candidate. Let x_j^l, $x_j^u \in \overline{\mathbb{Q}}$ be the local lower and upper bounds of x_j, and $x_j^b \in (x_j^l, x_j^u)$ be the potential branching point for branching on x_j. Usually, we choose $x_j^b = \tilde{x}_j$. If, however, $\tilde{x}_j$ is very close to one of the bounds, x_j^b is shifted

inwards the interval. Thus, for $x_j^l, x_j^u \in \mathbb{Q}$, we let $x_j^b := \min\{\max\{\tilde{x}_j, \lambda x_j^l + (1-\lambda)x_j^u\}, \lambda x_j^u + (1-\lambda)x_j^l\}$, where the parameter λ is set to 0.2 in our experiments.

As suggested in [6], we select the branching variable w.r.t. its pseudocost values. The pseudocosts are used to estimate the objective change in the LP relaxation when branching downwards and upwards on a particular variable. The pseudocosts of a variable are defined as the average objective gains per unit change, taken over all nodes, where this variable has been chosen for branching, see [8] for details.

In classical pseudocost branching for integer variables, the distances of $\tilde{x}_j$ to the nearest integers are used as multipliers of the pseudocosts. For continuous variables, we use another measure similar to "rb-int-br-rev" which was suggested in [6]: the distance of x_j^b to the bounds x_j^L and x_j^U for a variable x_j. This measure is motivated by the observation that the length of the domain determines the quality of the convexification. If the domain of x_j is unbounded, then the "convexification error of the variable x_j" will be used as multiplicator. This value is computed by assigning to each variable the gap evaluated in $\tilde{x}$ that is introduced by using a secant or McCormick underestimator for a nonconvex term that includes this variables.

As in [2], we combine the two estimates for downwards and upwards branching by multiplication rather than by a convex sum.

4. A constraint handler for Second-Order Cone constraints. Constraints of the form

$$\sqrt{\sum_{i=1}^{k} \gamma + (\alpha_i(x_i + \beta_i))^2} \leq \alpha_0 y + \beta_0, \qquad \alpha_0 y \geq -\beta_0, \tag{4.1}$$

where $\alpha_i, \beta_i \in \mathbb{Q}$, $i = 0, \ldots, k$, $\gamma \in \mathbb{Q}_+$ are handled by a constraint handler for second-order cone constraints. Note that SOC constraints are convex, i.e., the nonlinear function on the left hand side of (4.1) is convex. Therefore, unlike nonconvex quadratic constraints, SOC constraints can be enforced by separation routines solely. First, the inequality $\alpha_0 y \geq -\beta_0$ is ensured by tightening the bounds of y accordingly. Next, if the current LP solution $(\tilde{x}, \tilde{y})$ violates some SOC constraint (4.1), then we add the valid gradient-based inequality

$$\eta + \frac{1}{\eta} \sum_{i=1}^{k} \alpha_i^2 (\tilde{x}_i + \beta_i)(x_i - \tilde{x}_i) \leq \alpha_0 y + \beta_0,$$

where $\eta := \sqrt{\sum_{i=1}^{k} \gamma + (\alpha_i(\tilde{x}_i + \beta_i))^2}$. Note that since $(\tilde{x}, \tilde{y})$ violates (4.1), one has $\eta > \alpha_0 \tilde{y} + \beta_0 \geq 0$. For the initial linear relaxation, no inequalities are added.

We also experimented with adding a linear outer-approximation as suggested in [7] a priori, but did not observe computational benefits. Thus, this option has been disabled for the experiments in Section 6.

5. Primal heuristics. When solving MIQCPs, we still make use of all default MILP primal heuristics of SCIP. Most of these heuristics aim at finding good integer and LP feasible solutions starting from the optimum of the LP relaxation. For details and a computational study of the primal MILP heuristics available in SCIP, see [9].

So far, we have implemented two additional primal heuristics for solving MIQCPs in SCIP, both of which are based on *large neighborhood search*.

QCP local search. There are several cases, where the MILP primal heuristics already yield feasible solutions for the MIQCP. However, the heuristics usually construct a point $\hat{x}$ which is feasible for the MILP relaxation, i.e., the LP relaxation plus the integrality requirements, but violates some of the quadratic constraints. Such a point may, nevertheless, provide useful information, since it can serve as starting point for a local search.

The local search we currently use considers the space of continuous variables, i.e., if there are continuous variables in a quadratic part of a constraint, we solve a QCP obtained from the MIQCP by fixing all integer variables to the values of $\hat{x}$, using $\hat{x}$ as starting point for the QCP solver. Each feasible solution of this QCP also is a feasible solution of the MIQCP.

RENS. Furthermore, we implemented an extended form of the relaxation enforced neighborhood search (RENS) heuristic [10]. This heuristic creates a sub-MIQCP problem by exploiting the optimal solution $\tilde{x}$ of the LP relaxation at some node of the branch-and-bound-tree. In particular, it fixes all integer variables which take an integral value in $\tilde{x}$ and restricts the bounds of all integer variables with fractional LP solution value to the two nearest integral values. This, hopefully much easier, sub-MIQCP is then partially solved by a separate SCIP instance. Obviously, each feasible solution of the sub-MIQCP is a feasible solution of the original MIQCP.

Note that, during the solution process of the sub-MIQCP, the QCP local search heuristic may be used along with the default SCIP heuristics. For some instances this works particularly well since, amongst others, RENS performs additional presolving reductions on the sub-MIQCP – which yields a better performance of the QCP solver.

6. Computational Experiments. We conducted numerical experiments on three different test sets. The first is a test set of *mixed integer quadratic programs (MIQPs)* [22], i.e., problems with a quadratic objective function and linear constraints. Secondly, we selected a test set of *mixed integer conic programs (MICPs)* [27], which have been formulated as MIQCP. Finally, we assembled a test set of 24 general MIQCPs from the MINLPLib [14] and six constrained layout problems (`clay*`) from [15].

We will refer to these test sets as MIQP, MICP, and MINLP test sets. In Tables 1–3, each entry shows the number of seconds a certain solver needs to solve a problem. If the problem was not solved within the given time limit, the lower and upper bounds at termination are given. For

each instance, the fastest solution time or – in case all solvers hit the time limit – the best bounds, are depicted in bold face. Further, for each solver we calculated the geometric mean of the solution time (in which unsolved instances are accounted for with the time limit), and collected statistics on how often a solver solved a problem, computed the best dual bound, found the best primal solution value, or was the fastest among all solvers.

For our benchmark, we ran SCIP 1.2.0.7 using CPLEX 11.2.1 [19] as LP solver, Ipopt 3.8 [28] as QCP solver for the heuristics (cf. Section 5), and LAPACK 3.1.0 to compute eigenvalues. For comparison, we ran BARON 9.0.2 [26], Couenne 0.3 [6], CPLEX 12.1, LindoGlobal 6.0.1 [20], and MOSEK 6.0.0.55 [23]. Note that BARON, Couenne, and LindoGlobal can also be applied to general MINLPs. All solvers were run with a time limit of one hour, a final gap tolerance of 10^{-4}, and a feasibility tolerance of 10^{-6} on a 2.5 GHz Intel Core2 Duo CPU with 4 GB RAM and 6 MB Cache.

Mixed Integer Quadratic Programs. Table 6 presents the 25 instances from the MIQP test set [22]. We observe that due to the reformulation (3.1), 15 instances could be reformulated as mixed integer linear programs in the presolving state.

Table 1 compares the performance of SCIP, BARON, Couenne, and CPLEX on the MIQP test set. We did not run LindoGlobal since many of the MIQP instances exceed limitations of our LindoGlobal license. Note that some of the instances are nonconvex before applying the reformulation described in Section 3.1, so that we did not apply solvers which have only been designed for convex problems. Instance `ivalues` is the only instance that cannot be handled by CPLEX due to nonconvexity. Altogether, SCIP performs much better than BARON and Couenne and slightly better than CPLEX w.r.t. the mean computation time.

Mixed Integer Conic Programs. The MICP test set consists of three types of optimization problems, see Table 5. The `classical_XXX_YY` instances contain one convex quadratic constraint of the form $\sum_{j=1}^{k} x_j^2 \leq u$ for some $u \in \mathbb{Q}$, where `XXX` stand for the dimension k and `YY` is a problem index. The `robust_XXX_YY` instances contain one convex quadratic and one SOC constraint of dimension k. The `shortfall_XXX_YY` instances contain two SOC constraints of dimension k.

Table 2 compares the performance of BARON, Couenne, CPLEX, MOSEK, LindoGlobal, and SCIP on the MICP test set. We observe that on this specific test set SCIP outperforms BARON, Couenne, and LindoGlobal. It solves one instance more but is about 20% slower than the commercial solvers CPLEX and MOSEK.

Mixed Integer Quadratically Constrained Programs. The instances `lop97ic`, `lop97icx`, `pb302035`, `pb351535`, `qap`, and `qapw` were transformed into MILPs by presolving – which is due to the reformulation (3.1). The instances `nuclear*`, `space25`, `space25a`, and `waste` are

TABLE 1
Results on MIQP *instances. Each entry shows the number of seconds to solve a problem, or the bounds obtained after the one hour limit.*

instance	BARON	Couenne	CPLEX	SCIP
iair04	$[-\infty, \infty]$	fail	**37.52**	228.27
iair05	$[-\infty, \infty]$	$[25886, \infty]$	**30.71**	113.65
ibc1	$[1.792, 3.72]$	$[1.696, 3.98]$	895.54	**43.06**
ibell3a	58.95	198.90	**3.96**	14.59
ibienst1	1048.04	$[-\infty, 49.11]$	2836.05	**31.53**
icap6000	$[-2448496, -2441852]$	fail	6.28	**6.10**
icvxqp1	$[324603, 613559]$	fail	$[\mathbf{327522}, \mathbf{410439}]$	$[0, 4451398]$
ieilD76	$[729.5, 1081]$	$[808.3, 898.5]$	**13.50**	41.28
ilaser0	$[-\infty, \infty]$	fail	$[2409925, \mathbf{2412734}]$	fail
imas284	$[89193, 92241]$	3139.12	**4.36**	27.33
imisc07	$[2432, 2814]$	$[1696, 3050]$	70.02	**30.52**
imod011	$[\mathbf{-3.823}, \mathbf{-3.843}]$	fail	$[-\infty, \infty]$	$[-\infty, 0]$
inug06-3rd	$[177.8, \mathbf{1434}]$	fail	$[527.2, \mathbf{1434}]$	$[\mathbf{1152}, \infty]$
inug08	$[1451, 14696]$	$[683.1, \infty]$	2126.68	**24.83**
iportfolio	$[-\infty, 0]$	$[\mathbf{-0.4944}, \infty]$	$[\mathbf{-0.4944}, \mathbf{-0.4937}]$	$[-0.5251, 0]$
iqap10	$[-\infty, \infty]$	$[329.8, \infty]$	1411.26	**657.71**
iqiu	$[-402.5, -108.6]$	$[-357.5, -126.3]$	91.77	**64.53**
iran13x13	$[2930, 3355]$	$[3014, 3476]$	**20.02**	68.44
iran8x32	$[5013, 5454]$	$[5034, 5629]$	25.24	**8.31**
isqp0	$[-\infty, \infty]$	$[-\infty, -20137]$	$[\mathbf{-20338}, \mathbf{-20320}]$	$[-\infty, -19895]$
isqp1	$[-\infty, \infty]$	$[-\infty, -18801]$	$[\mathbf{-19028}, \mathbf{-18993}]$	$[-\infty, -17883]$
isqp	$[-\infty, \infty]$	$[-\infty, -20722]$	$[\mathbf{-21071}, \mathbf{-21001}]$	$[-\infty, \infty]$
iswath2	$[335.6, 661.9]$	$[335.9, 411.8]$	212.15	**121.29**
itointqor	$[-\infty, -1146]$	fail	$[\mathbf{-1150}, \mathbf{-1147}]$	$[-\infty, 0]$
ivalues	$[-12.88, -0.4168]$	$[\mathbf{-6.054}, \mathbf{-1.056}]$	–	$[-172.6, \infty]$
mean time	2923.78	3193.55	423.387	**303.013**
#solved	2	2	**15**	**15**
#best dual bound	4	5	**21**	16
#best primal sol.	5	3	**23**	15
#fastest	0	0	6	**9**

particularly difficult since they contain continuous variables that appear in quadratic terms with at least one bound at infinity. This prohibits to use the reformulation (3.1) for products of binary variables with a linear term. Further, generating secant and McCormick cuts for nonconvex terms is not possible. Thus, if the propagation algorithm cannot reduce domains for such unbounded variables, it may require many branching operations until reasonable variable bounds and a lower bound can be computed.

Table 3 compares the performance of BARON, Couenne, LindoGlobal, and SCIP on the MINLP test set. Figure 2 shows a performance profile for this particular test set. Regarding the number of solved instances, LindoGlobal performs best: it could solve two instances more than BARON and SCIP, which both solved six instances more than Couenne. SCIP was, however, significantly faster than the other solvers.

BARON wrongly declared the instance `product` to be infeasible and hit the time limit while parsing the instances `pb302035` and `pb351535`. Couenne wrongly declared the instances `product` and `waste` to be infeasible. Using a time limit of 3600 seconds, LindoGlobal and Couenne did not stop after 4000 seconds for `pb351535` and for `pb302035`, `pb351535`, respectively.

TABLE 2
Results on MICP *instances. Each entry shows the number of seconds to solve a problem, or the bounds obtained after the one hour limit.*

instance	BARON	Couenne	CPLEX	LindoGlobal	MOSEK	SCIP
classical_40_0	207.24	178.72	**1.60**	38.25	12.79	33.16
classical_40_1	7.09	114.49	**1.01**	217.55	22.28	5.53
classical_50_0	[−0.09191, −0.09074]	[−0.09447, −0.09054]	**41.33**	[−0.09572, −0.09074]	161.17	2664.58
classical_50_1	250.46	[−0.09595, −0.09459]	**7.38**	[−0.09593, −0.09476]	30.62	210.65
classical_200_0	[−0.1247, −0.1077]	[−0.1255, −0.0951]	[**−0.1231**, **−0.1106**]	[−0.1256, −0.08574]	[−0.124, −0.1103]	[−0.1284, −0.108]
classical_200_1	[−0.1269, −0.1149]	[−0.1283, −0.1036]	[**−0.1257**, **−0.1164**]	[−0.1284, −0.1093]	[−0.1266, −0.1162]	[−0.1311, −0.1162]
robust_40_0	3473.03	[−0.09706, −0.07602]	**0.67**	3600.95	1.37	4.03
robust_40_1	1752.70	[−0.116, −0.07646]	**0.64**	249.72	2.88	2.29
robust_50_0	[−0.08615, −0.08609]	[−0.1263, −0.0861]	1.88	165.45	**0.90**	1.51
robust_50_1	[−0.08574, −0.08569]	[−0.1274, −0.0857]	**2.32**	506.36	3.18	8.71
robust_100_0	[−0.1013, −0.0932]	[−0.1514, −0.09747]	[−0.1043, −0.09721]	[−0.1542, −0.08833]	**1210.86**	1392.47
robust_100_1	[−0.07501, −0.0703]	[−0.1257, −0.0677]	525.14	[−0.1269, 0]	**292.21**	592.92
robust_200_0	[−0.1722, −0.1363]	fail	[**−0.145**, **−0.1411**]	[−1, 0]	[−0.1468, **−0.1411**]	[−0.1452, **−0.1411**]
robust_200_1	[−0.1477, −0.1424]	[−0.1995, −0.1377]	[**−0.1454**, −0.1425]	[−1, 0]	[−0.1456, **−0.1427**]	[−0.1467, −0.1413]
shortfall_40_0	102.17	333.76	247.99	1027.02	45.71	**15.39**
shortfall_40_1	5.99	133.49	5.71	288.12	13.72	**3.00**
shortfall_50_0	[−1.098, −1.095]	[−1.102, −1.095]	1913.00	[−1.104, −1.095]	**405.93**	1602.81
shortfall_50_1	91.84	[−1.103, −1.099]	**13.13**	[−1.104, −1.102]	21.73	15.44
shortfall_100_0	[−1.12, **−1.114**]	[−1.126, −1.102]	[−1.132, −1.112]	[−1.125, −1.114]	[**−1.116**, **−1.114**]	[−1.121, **−1.114**]
shortfall_100_1	[−1.109, −1.106]	[−1.113, −1.091]	3301.75	[−1.113, −1.105]	[−1.111, −1.106]	**2152.56**
shortfall_200_0	[−1.149, −1.12]	[−1.15, −1.094]	[**−1.146**, −1.125]	[−1.479, −1.08]	[**−1.146**, **−1.126**]	[−1.149, −1.119]
shortfall_200_1	[−1.15, −1.131]	[−1.152, −1.101]	[−1.15, −1.133]	[−1.361, −1.089]	[−1.151, **−1.135**]	[**−1.148**, −1.134]
mean time	1202.41	2092.28	**226.932**	1552.67	228.392	288.956
#solved	8	4	14	8	14	**15**
#best dual bound	8	4	**19**	8	16	16
#best primal sol.	13	9	17	12	**19**	17
#fastest	0	0	**8**	0	4	3

TABLE 3
Results on MINLP *instances. Each entry shows the number of seconds to solve a problem, or the bounds obtained after the one hour limit.*

instance	BARON	Couenne	LindoGlobal	SCIP
clay0203m	1.56	2.03	43.13	**0.15**
clay0204m	48.25	4.79	85.50	**0.52**
clay0205m	971.83	27.73	1162.00	**6.49**
clay0303m	1.29	$[-\infty, 29911]$	62.17	**0.37**
clay0304m	14.77	16.31	187.38	**0.90**
clay0305m	3584.73	27.65	1112.42	**7.45**
du-opt	137.39	$[-8727, \infty]$	2204.70	**1.07**
du-opt5	150.54	$[-2437, 9.012]$	697.10	**0.47**
lop97ic	$[2549, \infty]$	$[\mathbf{3826}, \infty]$	$[-\infty, \infty]$	$[3069, \mathbf{4547}]$
lop97icx	$[2812, 4415]$	$[\mathbf{3903}, 4272]$	$[0, 5259]$	$[3763, \mathbf{4099}]$
nous1	641.19	$[1.345, 1.567]$	**41.44**	$[1.195, 1.567]$
nous2	0.97	2.69	**0.36**	1349.72
nuclear14a	$[-\mathbf{12.26}, \infty]$	$[-\mathbf{12.26}, \infty]$	$[-\infty, \infty]$	$[-228.2, -\mathbf{1.105}]$
nuclear14b	$[-\mathbf{2.078}, -1.107]$	$[-2.234, \infty]$	$[-\infty, \infty]$	$[-198.3, -\mathbf{1.118}]$
nuclear14	$[-\infty, \infty]$	$[-\infty, -1.12]$	$[-\infty, -\mathbf{1.126}]$	$[-\infty, -1.122]$
nuclearva	$[-\infty, \infty]$	$[-\infty, -\mathbf{1.005}]$	$[-\infty, \infty]$	$[-\infty, \infty]$
nvs19	12.14	778.31	457.04	**0.21**
nvs23	44.54	$[-1380, -1109]$	2533.14	**0.40**
pb302035	$[-\infty, \infty]$	fail	$[622588, \infty]$	$[\mathbf{1138613}, \mathbf{4019210}]$
pb351535	$[-\infty, \infty]$	fail	fail	$[\mathbf{1710093}, \mathbf{4976433}]$
product	fail	fail	$[-2200, -2092]$	**326.76**
qap	$[\mathbf{103040}, \mathbf{388250}]$	$[0, \infty]$	$[-\infty, \infty]$	$[24761, 410450]$
qapw	$[\mathbf{265372}, \mathbf{391210}]$	$[0, \infty]$	$[0, 405354]$	$[32191, 400150]$
space25a	$[99.99, 490.2]$	$[35.09, \infty]$	$[\mathbf{330.6}, \mathbf{489.2}]$	$[72.46, \infty]$
space25	$[\mathbf{84.91}, \mathbf{520.9}]$	$[42.68, \infty]$	$[33.07, 638.8]$	$[72.46, \infty]$
tln12	$[32.73, \infty]$	$[16.19, \infty]$	$[\mathbf{85.8}, 139.1]$	$[16.41, \mathbf{91.6}]$
tln5	798.56	$[6.592, 10.3]$	174.02	**32.56**
tln6	$[13.75, 15.3]$	$[7.801, 15.3]$	**182.23**	$[10.21, 15.3]$
tln7	$[12.38, 15.6]$	$[5.038, 16.1]$	$[\mathbf{14.2}, 15.6]$	$[7.016, \mathbf{15}]$
waste	$[306.7, 712.3]$	fail	**1532.71**	$[306.7, 670.6]$
mean time	755.466	1215.11	995.697	**400.464**
#solved	13	7	**15**	13
#best dual bound	**18**	10	**18**	15
#best primal sol.	17	11	17	**23**
#fastest	0	0	4	**12**

Further, no bounds were reported in the log file.

CPLEX can be applied to 11 instances of this test set. The clay* and du-opt* instances were solved within seconds; 4 times CPLEX was fastest, 4 times SCIP was fastest. For the instances pb302035, pb351535, and qap, CPLEX found good primal solutions, but very weak lower bounds.

Evaluation of implemented MIQCP techniques. In order to evaluate the computational effects of the implemented techniques, we compare the default settings of SCIP with a series of settings where a single technique has been turned off at a time. The methods we evaluated are the reformulation (3.1) for products that involve binary variables, cf. Section 3.1, the handling of SOC constraints by the SOC constraint handler, cf. Section 4, the domain propagation for quadratic constraints during branch-and-bound, cf. Section 3.3, the detection of convexity for multivariate quadratic functions, cf. Section 3.1, the QCP local search heuristic,

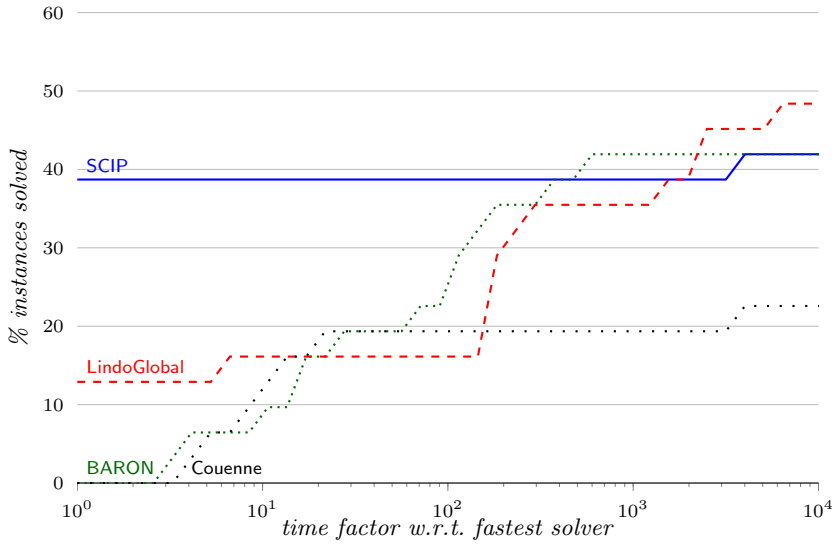

FIG. 2. *Performance profile for* MINLP *test set.*

cf. Section 5, and the extended RENS heuristic, cf. Section 5.

For each method, we evaluate the performance only on those instances from the test sets MIQP, MICP, and MINLP where the method to evaluate may have an effect (e.g., disabling the reformulation (3.1) is only evaluated on instances where this reformulation can be applied). The results are summarized in Table 4. For a number of performance measures we report the relative change caused by disabling a particular method.

We observe that deactivating one of the methods always leads to more deteriorations than improvements for both, the dual and primal bounds at termination. Except for one instance in the case of switching off binary reformulations, the number of solved instances remains equal or decreases.

Recognizing SOC constraints and convexity allows to solve instances of those special types much faster. Disabling domain propagation or one of the primal heuristics yields a small improvement w.r.t. computation time for easy instances, but results in weaker bounds for those instances which could not be solved within the time limit. We further observed that switching off the QCP local search heuristic increases the time until the first feasible solution is found by 93% and the time until the optimal solution is found by 26%. For RENS, the numbers are 12% and 43%, accordingly. Therefore, we still consider applying these techniques to be worthwhile.

7. Conclusions. In this paper, we have shown how a framework for constraint integer programming can be extended towards a solver for general MIQCPs. We implemented methods to correctly handle the quadratic constraints. In order to speed up computations we further implemented

TABLE 4
Relative impact of implemented MIQCP methods. Percentages in columns 3–9 are relative to the size of the test set. Percentage in mean time column is relative to the mean time of SCIP with default settings.

			primal sol.		dual bound		running time		
disabled feature	size	solved	better	worse	better	worse	better	worse	mean
binary reform.	32	+3%	13%	22%	6%	25%	22%	34%	+3%
SOC upgrade	16	−69%	0%	69%	0%	100%	0%	69%	+1317%
domain prop.	48	±0%	4%	8%	6%	17%	29%	15%	−4%
convexity check	10	−20%	20%	20%	0%	30%	10%	30%	+159%
QCP local search	48	±0%	2%	17%	2%	4%	38%	17%	−6%
RENS heuristic	56	±0%	5%	9%	7%	7%	41%	11%	−3%

MIQCP specific presolving, propagation, and separation methods. Furthermore, we discussed two large neighborhood search heuristics for MIQCP. The computational results indicate that this already suffices for building a solver which is competitive to state-of-the-art solvers like CPLEX, BARON, Couenne, and LindoGlobal. SCIP performed particularly well on the MIQP and MICP test sets, which contain a large number of linear constraints and a few quadratic constraints. These results meet our expectations, since SCIP already features several sophisticated MILP technologies.

We conclude that the extension of a full-scale MILP solver for handling MIQCP is a promising approach. The next step towards a full-scale MIQCP solver will be the incorporation of further MIQCP specific components into SCIP, e.g., more sophisticated separation routines [5, 24] and specialized constraint handlers, e.g., for bilinear covering constraints [25].

Acknowledgment. We like to thank Ambros M. Gleixner and Marc E. Pfetsch for their valuable comments on the paper. We further thank the anonymous reviewers for their constructive suggestions.

REFERENCES

[1] K. ABHISHEK, S. LEYFFER, AND J. LINDEROTH, *FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs*, Tech. Rep. ANL/MCS-P1374-0906, Argonne National Laboratory, Mathematics and Computer Science Division, 2006.

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

[3] ———, *SCIP: Solving Constraint Integer Programs*, Math. Program. Comput., 1 (2009), pp. 1–41.

[4] T. ACHTERBERG, T. BERTHOLD, T. KOCH, AND K. WOLTER, *Constraint integer programming: A new approach to integrate CP and MIP*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008, L. Perron and M. Trick, eds., vol. 5015 of LNCS, Springer, 2008, pp. 6–20.

[5] X. BAO, N. SAHINIDIS, AND M. TAWARMALANI, *Multiterm polyhedral relaxations for nonconvex, quadratically-constrained quadratic programs*, Optimization Methods and Software, 24 (2009), pp. 485–504.

[6] 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 (2009), pp. 597–634.

[7] A. Ben-Tal and A. Nemirovski, *On polyhedral approximations of the second-order cone*, Math. Oper. Res., 26 (2001), pp. 193–205.

[8] M. Bénichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent, *Experiments in mixed-integer linear programming*, Math. Program., 1 (1971), pp. 76–94.

[9] T. Berthold, *Primal heuristics for mixed integer programs*. Diploma thesis, Technische Universität Berlin, 2006.

[10] ———, *RENS – relaxation enforced neighborhood search*, ZIB-Report 07-28, Zuse Institute Berlin, 2007.

[11] T. Berthold, S. Heinz, and M. E. Pfetsch, *Nonlinear pseudo-boolean optimization: relaxation or propagation?*, in Theory and Applications of Satisfiability Testing – SAT 2009, O. Kullmann, ed., no. 5584 in LNCS, Springer, 2009, pp. 441–446.

[12] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling, *MIP: theory and practice – closing the gap*, in System Modelling and Optimization: Methods, Theory and Applications, M. Powell and S. Scholtes, eds., Kluwer, 2000, pp. 19–50.

[13] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. W. Sawaya, and A. Wächter, *An algorithmic framework for convex mixed integer nonlinear programs*, Discrete Optim., 5 (2008), pp. 186–204.

[14] M. R. Bussieck, A. S. Drud, and A. Meeraus, *MINLPLib - a collection of test models for mixed-integer nonlinear programming*, INFORMS J. Comput., 15 (2003), pp. 114–119.

[15] CMU-IBM MINLP Project. `http://egon.cheme.cmu.edu/ibm/page.htm`.

[16] F. Domes and A. Neumaier, *Constraint propagation on quadratic constraints*, Constraints, to appear (2010).

[17] J. N. Hooker, *Integrated Methods for Optimization*, International Series in Operations Research & Management Science, Springer, New York, 2007.

[18] R. Horst and H. Tuy, *Global Optimization: Deterministic Approaches*, Springer, 1990.

[19] IBM, *CPLEX*. `http://ibm.com/software/integration/optimization/cplex`.

[20] Y. Lin and L. Schrage, *The global solver in the LINDO API*, Optimization Methods and Software, 24 (2009), pp. 657–668.

[21] G. McCormick, *Computability of global solutions to factorable nonconvex programs: Part I-Convex Underestimating Problems*, Math. Program., 10 (1976), pp. 147–175.

[22] H. Mittelmann, *MIQP test instances*. `http://plato.asu.edu/ftp/miqp.html`.

[23] MOSEK Corporation, *The MOSEK optimization tools manual*, 6.0 ed., 2009.

[24] A. Saxena, P. Bonami, and J. Lee, *Convex relaxations of non-convex mixed integer quadratically constrained programs: Projected formulations*, Tech. Rep. RC24695, IBM Research, 2008. to appear in Math. Program.

[25] M. Tawarmalani, J.-P. P. Richard, and K. Chung, *Strong valid inequalities for orthogonal disjunctions and bilinear covering sets*. `http://www.optimization-online.org/DB_HTML/2008/09/2100.html`, 2008.

[26] M. Tawarmalani and N. Sahinidis, *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, Kluwer Academic Publishers, 2002.

[27] J. P. Vielma, S. Ahmed, and G. L. Nemhauser, *A lifted linear programming branch-and-bound algorithm for mixed integer conic quadratic programs*, INFORMS J. Comput., 20 (2008), pp. 438–450.

[28] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Math. Program., 106 (2006), pp. 25–57.

APPENDIX

In this section, detailed problem statistics are presented for the three test sets, MICP (Table 5), MIQP, and MINLP (both in Table 6). The columns belonging to "original problem" state the structure of the original problem. The "presolved problem" columns show statistics about the MIQCP after SCIP has applied its presolving routines, including the ones described in Section 3.1. The columns "vars", "int", and "bin" show the number of all variables, the number of integer variables, and the number of binary variables, respectively. The columns "linear", "quad", and "soc" show the number of linear, quadratic, and second-order cone constraints, respectively. The column "conv" indicates whether all quadratic constraints of the presolved MIQCP are convex or whether at least one of them is nonconvex.

TABLE 5
Statistics of instances in MICP *test set.*

	original problem					presolved problem					
instance	vars	int	bin	linear	quad	vars	int	bin	linear	quad	soc
classical_40_0	120	0	40	82	1	120	0	40	82	1	0
classical_40_1	120	0	40	82	1	120	0	40	82	1	0
classical_50_0	150	0	50	102	1	150	0	50	102	1	0
classical_50_1	150	0	50	102	1	150	0	50	102	1	0
classical_200_0	600	0	200	402	1	600	0	200	402	1	0
classical_200_1	600	0	200	402	1	600	0	200	402	1	0
robust_40_0	163	0	41	124	2	163	0	41	124	1	1
robust_40_1	163	0	41	124	2	163	0	41	124	1	1
robust_50_0	203	0	51	154	2	203	0	51	154	1	1
robust_50_1	203	0	51	154	2	203	0	51	154	1	1
robust_100_0	403	0	101	304	2	403	0	101	304	1	1
robust_100_1	403	0	101	304	2	403	0	101	304	1	1
robust_200_0	803	0	201	604	2	803	0	201	604	1	1
robust_200_1	803	0	201	604	2	803	0	201	604	1	1
shortfall_40_0	164	0	41	125	2	164	0	41	125	0	2
shortfall_40_1	164	0	41	125	2	164	0	41	125	0	2
shortfall_50_0	204	0	51	155	2	204	0	51	155	0	2
shortfall_50_1	204	0	51	155	2	204	0	51	155	0	2
shortfall_100_0	404	0	101	305	2	404	0	101	305	0	2
shortfall_100_1	404	0	101	305	2	404	0	101	305	0	2
shortfall_200_0	804	0	201	605	2	804	0	201	605	0	2
shortfall_200_1	804	0	201	605	2	804	0	201	605	0	2

TABLE 6
Statistics of instances in MIQP *(first part) and* MINLP *(second part) test set.*

	original problem					presolved problem					
instance	vars	int	bin	linear	quad	vars	int	bin	linear	quad	conv
iair04	8905	0	8904	823	1	12848	0	7362	17464	0	✓
iair05	7196	0	7195	426	1	10574	0	6117	14218	0	✓
ibc1	1752	0	252	1913	1	866	0	252	1438	0	✓
ibell3a	123	29	31	104	1	129	29	31	161	1	✓
ibienst1	506	0	28	576	1	473	0	28	592	0	✓
icap6000	6001	0	6000	2171	1	7323	0	5865	6362	0	✓
icvxqp1	10001	10000	0	5000	1	10003	9998	2	5006	1	✓
ieilD76	1899	0	1898	75	1	2685	0	1898	3168	0	✓
ilaser0	1003	151	0	2000	1	1003	151	0	1000	1	✓
imas284	152	0	150	68	1	228	0	150	299	0	✓
imisc07	261	0	259	212	1	360	0	238	598	0	✓
imod011	10958	1	96	4480	1	8963	1	96	2730	1	✓
inug06-3rd	2887	0	2886	3972	1	3709	0	2886	7779	0	✓
inug08	1633	0	1632	912	1	2217	0	1632	3076	0	✓
iportfolio	1201	192	775	201	1	1201	192	775	201	1	✓
iqap10	4151	0	4150	1820	1	5879	0	4150	9047	0	✓
iqiu	841	0	48	1192	1	871	0	48	1285	0	✓
iran13x13	339	0	169	195	1	468	0	169	585	0	✓
iran8x32	513	0	256	296	1	651	0	256	713	0	✓
isqp0	1001	50	0	249	1	1001	50	0	249	1	✓
isqp1	1001	0	100	249	1	1068	0	100	480	1	
isqp	1001	50	0	249	1	1001	50	0	249	1	✓
iswath2	6405	0	2213	483	1	8007	0	2213	5631	0	✓
itointqor	51	50	0	0	1	51	50	0	0	1	✓
ivalues	203	202	0	1	1	203	202	0	1	1	
clay0203m	30	0	18	30	24	27	0	15	27	24	✓
clay0204m	52	0	32	58	32	48	0	28	54	32	✓
clay0205m	80	0	50	95	40	75	0	45	90	40	✓
clay0303m	33	0	21	30	36	31	0	19	29	36	✓
clay0304m	56	0	36	58	48	54	0	34	57	48	✓
clay0305m	85	0	55	95	60	81	0	51	93	60	✓
du-opt	21	13	0	9	1	21	13	0	5	1	✓
du-opt5	21	13	0	9	1	19	11	0	4	1	✓
lop97ic	1754	831	831	52	40	5228	708	708	11521	0	✓
lop97icx	987	831	68	48	40	488	68	68	1138	0	✓
nous1	51	0	2	15	29	47	0	2	11	29	
nous2	51	0	2	15	29	47	0	2	11	29	
nvs19	9	8	0	0	9	9	8	0	0	9	
nvs23	10	9	0	0	10	10	9	0	0	10	
pb302035	601	0	600	50	1	1199	0	600	1847	0	✓
pb351535	526	0	525	50	1	1048	0	525	1619	0	✓
product	1553	0	107	1793	132	446	0	92	450	82	
qap	226	0	225	30	1	449	0	225	702	0	✓
qapw	451	0	225	255	1	675	0	225	930	0	✓
space25	893	0	750	210	25	767	0	716	118	25	
space25a	383	0	240	176	25	308	0	240	101	25	
nuclear14	1562	0	576	624	602	986	0	576	48	602	
nuclear14a	992	0	600	49	584	1568	0	600	2377	560	
nuclear14b	1568	0	600	1225	560	1568	0	600	1225	560	
nuclearva	351	0	168	50	267	327	0	144	24	267	
tln12	168	156	12	60	12	180	144	24	85	11	
tln5	35	30	5	25	5	35	30	5	20	5	
tln6	48	42	6	30	6	48	42	6	24	6	
tln7	63	56	7	35	7	63	56	7	28	7	
waste	2484	0	400	623	1368	1238	0	400	516	1230	