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 MIP and CP technologies. In particular, this addresses the linear relaxation and the discrete components of the problem. For relaxation, we use an outer approximation generated by linearization of convex constraints and linear underestimation of nonconvex constraints. 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 indicates the potential of the approach.

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 programs (MIPs)* [2, 10]. Furthermore, it has been shown that successful MIP solving techniques can often be extended to the more general case of *mixed integer nonlinear programs (MINLPs)* [1, 4, 11]. Analogously, several authors have shown that an integrated approach of *constraint programming (CP)* and mixed integer programming (MIP) can help to solve optimization problems that were intractable with either of the two methods alone, for an overview see [15, 28].

The paradigm of *constraint integer programming (CIP)* [2, 3] combines modeling and solving techniques from the fields of constraint programming (CP), mixed integer programming, and the solution of *satisfiability problems (SAT)*. The concept of CIP aims at restricting the generality of CP modeling as little as needed to retain the full performance of all MIP solving techniques. This still allows for a wide range of optimization problems. For example, in [2], it is shown that CIP includes MIP 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 allows to utilize the complete power of already existing MIP and CP technologies for handling the linear and the discrete parts of the problem.

*Supported by the DFG Research Center MATHEON *Mathematics for key technologies*.

†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

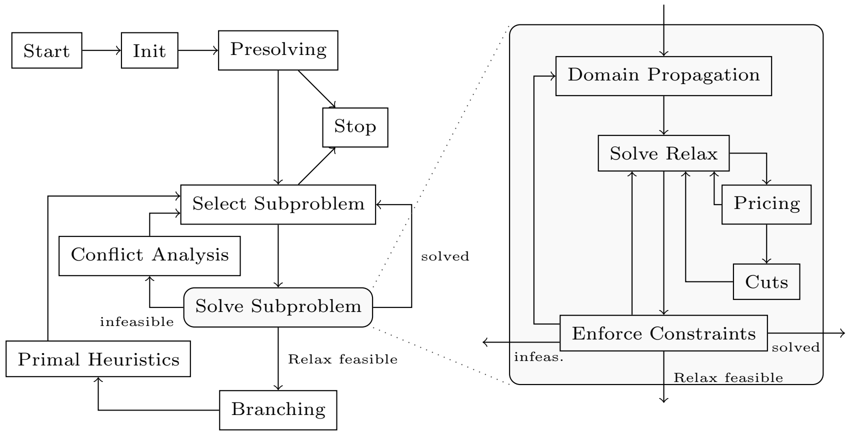


FIG. 1. *Flowchart of the main solving loop of SCIP*

For this purpose, we use the branch-cut-and-price framework SCIP (Solving Constraint Integer Programs). SCIP incorporates the idea of CIP and implements several state-of-the-art techniques for MIP solving. 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 branching 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 given in minimization form. The relaxation may be strengthened by adding further valid constraints, which cut off the optimum of the relaxation. In case of an infeasible subproblem, 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 interdependencies between the main algorithmic components of SCIP. In the context of this article, the relaxation employed in SCIP is a linear program (LP).

The integration of MIQCP is a first step towards the incorporation of MINLP into the concept of constraint integer programming. The remainder of this article is organized as follows. In Section 2, we give the necessary definitions of MIQCP and CIP, in Sections 3 and 4, we show how to handle quadratic constraints inside SCIP, and in Section 5, we present computational results.

2. Problem definition. A mixed integer quadratically constrained program (MIQCP) is an optimization problem of the form

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

where $I \subseteq N := \{1, \dots, n\}$ 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 $i = 1, \dots, m$, $x^L \in \overline{\mathbb{Q}}^n$ and $x^U \in \overline{\mathbb{Q}}^n$, with $\overline{\mathbb{Q}} := \mathbb{Q} \cup \{\pm\infty\}$, are the lower and upper bounds of the variables x, respectively. Note that we do not require the matrices A_i to be positive semidefinite, hence we also allow for nonconvex quadratic constraints.

The definition of CIP, as given in [2], 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 $f(x) \le z$. Thus, formulation (2.1) also covers the general case of mixed integer all quadratic problems.

In this article, we use a definition of CIP which is slightly different from the one given in [2, 3].

A constraint integer program (CIP) consists of solving

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

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

In [2, 3], it is required that the remaining subproblem after fixing all integer variables is a linear program – in order to guarantee finite solvability. In this article, we require, however, the remaining subproblem to be a *quadratically constrained program (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 [16].

3. A constraint handler for quadratic constraints. 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, satisfies 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 the computation, a constraint handler may further implement additional features like presolving, cut separation, and domain propagation for its particular class of constraints.

In the following, we discuss the presolving, separation, propagation, and enforcement algorithms that are used in our handler for quadratic constraints.

3.1. Presolving. During the presolving phase, a set of reformulations and simplifications are tried. When SCIP fixes or aggregates variables, then the corresponding reformulations are also 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. Squares of binary variables are 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 $a_i \in \mathbb{Q}$, $i = 1, \ldots, k$, this product is replaced by a new variable z and the linear constraints

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

In the case that $k = 1$ and y_1 is also a binary variables, the product xy_1 is replaced by a new variable z and the constraint $z = x \wedge y_1$ by using SCIP's handler for AND constraints [9].

Second-Order Cone Constraints. Constraints of the form

$$\sum_{i=1}^{k} (\alpha_i x_i)^2 \leq (\beta y)^2, \qquad y \geq 0, \tag{3.2}$$

where $\alpha_i \in \mathbb{Q}$, $i = 1 \ldots, k$, and $\beta \in \mathbb{Q}$ are recognized as second-order cone constraints. For the case $k = 2$, we add a linear outer-approximation as suggested in [5] as follows: A parameter $N > 0$ determines the number of additional variables. We add variables $y_0, y_1, \ldots, y_N$ and $z_0, z_1, \ldots, z_N$ and the following linear constraints:

$$\begin{aligned}
y_0 &\geq |\alpha_1 x_1|, \\
z_0 &\geq |\alpha_2 x_2|, \\
y_j &= \cos\left(\frac{\pi}{2^{j+1}}\right) y_{j-1} + \sin\left(\frac{\pi}{2^{j+1}}\right) z_{j-1}, && j = 1, \ldots, N, \\
z_j &\geq \left| -\sin\left(\frac{\pi}{2^{j+1}}\right) y_{j-1} + \cos\left(\frac{\pi}{2^{j+1}}\right) z_{j-1} \right|, && j = 1, \ldots, N, \\
y_N &\leq \beta x_3, \\
z_N &\leq \tan\left(\frac{\pi}{2^{N+1}}\right) y_N.
\end{aligned} \tag{3.3}$$

The gap between the outer-approximation given by (3.3) and the set given by the original constraint (3.2) is of order $O(4^{-N})$. In our implementation, we add the constraint set (3.3) during the presolving phase. Note that we also keep the original quadratic constraint (3.2). The idea is to use all MIP features such as presolving and cut generation on the linear relaxation (3.3). Furthermore, separation, propagation, and branching routines for the quadratic constraint (3.2) ensure feasibility w.r.t. the original formulation.

For the general case of a second-order cone constraint of dimension $k > 2$, we first add new positive variables w_1 and w_2 and the linear approximation of the second-order cone constraint $w_1^2 + w_2^2 \leq (\beta y)^2$. Then the method is applied recursively to the constraints $\sum_{i=1}^{\lfloor k/2 \rfloor} (\alpha_i x_i)^2 \leq w_1^2$ and $\sum_{i=\lfloor k/2 \rfloor+1}^{k} (\alpha_i x_i)^2 \leq w_2^2$.

Disaggregation. Assume that a quadratic constraint has the block-separable form

$$c_i + b_i^T x + \sum_{k=1}^{p} x_{J_k}^T A_{i,k} x_{J_k} \leq 0, \tag{3.4}$$

where J_k, $k = 1, \ldots, p$, are pairwise-disjoint subsets of $\{1, \ldots, n\}$, $p > 1$ is the number of blocks, and x_{J_k} denotes the vector $\{x_j\}_{j \in J_k}$. In this case, we introduce new variables $z_2, \ldots, z_p$ and replace the constraint (3.4) by the equivalent set of constraints

$$\begin{aligned}
c_i + b_i^T x + \sum_{k=2}^{p} z_k + x_{J_1}^T A_{i,1} x_{J_1} &\leq 0 \\
x_{J_k}^T A_{i,k} x_{J_k} - z_k &\leq 0 \qquad k = 2, \ldots, p.
\end{aligned} \tag{3.5}$$

There is computational evidence that this reformulation reduces the number of cuts needed to obtain a tight linear outer-approximation of the set defined by (3.4). Note, that requiring the sets J_k, $k = 2, \ldots, p$, to be pairwise disjoint ensures that convexity of a quadratic expression in a block of (3.4) is not destroyed by the reformulation (3.5). This can also lead to further bound tightenings in the domain propagation process.

Convexity. After the presolving phase is finished, each quadratic constraint is checked for convexity by computing the sign of the minimal 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 cutting plane

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

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

$$c_i + b_i^T x + a(2\lfloor \tilde{x}_j \rfloor + 1)x_j - a\lfloor \tilde{x}_j \rfloor(\lfloor \tilde{x}_j \rfloor + 1) \leq 0.$$

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. However, for the case $a < 0$, 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, the constraint handler may infer deductions of the variables' local domains. Domain deductions can yield stronger linear underestimators in the separation procedures, they may cut off nodes due to infeasibility of a constraint, and can result in further domain deductions on other constraints. For quadratic constraints, we implemented an interval-arithmetic based method similar to [14].

To allow for an efficient propagation, we write a quadratic constraint in the form

$$\sum_{j\in J} d_j x_j + \sum_{k\in K} (e_k + p_{k,k}x_k + \sum_{r\in K} p_{k,r}x_r)x_k \in [\ell, u], \tag{3.6}$$

such that $\ell, u \in \overline{\mathbb{Q}}$, $J \cup K \subseteq N$, $J \cap K = \emptyset$, and $p_{k,r} = 0$ for $k > r$. For a number a, 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 interval $\{by + ay^2 : y \in [y^L, y^U], b \in [b^L, b^U]\}$. This can be computed analytically [14].

The forward propagation step aims at tightening the bounds $[\ell, u]$ in (3.6). For this purpose, we replace the variables x_j and x_r in (3.6) 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 $[f_k^L, f_k^U] \coloneqq [e_k, e_k] + \sum_{r\in K} p_{k,r}[x_r^L, x_r^U]$ is computed by interval-arithmetic. 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 (3.6) can take w.r.t. to the current variables' domains. If $[h^L, h^U] \cap [\ell, u] = \emptyset$, then the constraint (3.4) cannot be satisfied for any $x \in [x^L, x^U]$. In this case, the current branch-and-bound node can be cut off. Otherwise, we can tighten $[\ell, u]$ to $[\ell, u] \cap [h^L, h^U]$.

The backward propagation step aims at inferring domain deductions on the variables in (3.4) from the bounds $[\ell, u]$. 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 tight bounds is by solving the quadratic interval-equation

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

However, since evaluating the coefficient of x_k for each $k \in K$ may produce a huge computational overhead, especially for constraints with many bilinear terms, we resort to compute the solution set of

$$\sum_{j\in J} d_j[x_j^L, x_j^U] + \sum_{\substack{k'\in K\\ k'\neq k}} \left(q(p_{k',k'}, [e_{k'}, e_{k'}], x_{k'}) + \sum_{\substack{r\in K\\ r\neq k'}} p_{k',r}[x_{k'}^L, x_{k'}^U][x_r^L, x_r^U] \right) + ([e_k, e_k] + \sum_{r\in K} (p_{k,r} + p_{r,k})[x_r^L, x_r^U])x_k + p_{k,k}x_k^2 \in [\ell, u], \quad (3.7)$$

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 (3.7) is empty, we can deduce infeasibility and cut off the corresponding node. Otherwise, we may be able to tighten the bounds of x_k.

As in [14], all interval operations detailed in this section are performed in outward rounding mode.

3.4. Enforcement. In the enforcement call, the constraint handler has to check whether the current LP solution $\tilde{x}$ is feasible for the constraints of the constraint handler. If it is not feasible, it can resolve this 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 cutting off 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 cut off 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 branching candidate. Let x_j^l, $x_j^u \in \overline{\mathbb{Q}}$ be the 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.

As suggested in [4], 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 [6] for details.

In classical pseudocost branching for integer variables, the distances of $\tilde{x}_j$ to the nearest integers are used as multiplicators for the pseudocosts. For continuous variables, we use another measure similar to "rb-int-br-rev" which was suggested in [4]: the distance of x_j^b to the bounds x_j^L and x_j^U for a variable x_j. This is motivated by the observation that the width 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" is 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 in which this variables appears.

We combine the two estimates for downwards and upwards branching by multiplication rather then by a convex sum, since this usually performs much better [2].

4. Primal heuristics. When solving MIQCPs, we still make use of all default MIP primal heuristics of SCIP. Most of these heuristics base on the LP relaxation and aim at finding good integer-feasible solutions starting from the optimum of the LP relaxation. For a detailed description and computational study of the primal MIP heuristics available in SCIP, see [7].

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

QCP local search. There are several cases, where the MIP primal heuristics already yield feasible solutions for the MIQCP. However, the heuristics usually construct a point $\hat{x}$ which is only feasible for the MIP relaxation, hence the LP relaxation plus the integrality requirements, but violate some of the quadratic constraints. Such a point may, nevertheless, still 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. That is, 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.

Relaxation enforced neighborhood search. Furthermore, we implemented an extended form of the relaxation enforced neighborhood search (RENS) heuristic [8]. This heuristic creates a sub-MIQCP problem by exploiting the optimum of the LP relaxation $\tilde{x}$ 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 next to the SCIP default 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.

5. Numerical Experiments. We conducted numerical experiments on three different testsets. The first is a testset of H. Mittelmann of *mixed integer quadratic programs (MIQPs)* [22], i.e., problems with a quadratic objective function and linear constraints. Second, we have selected a testset of *mixed integer conic programs (MICPs)*, which have been formulated as MIQCP. They represent three different classes of portfolio optimization problems and have been introduced in [26]. Finally, we have assembled a testset of general MIQCPs from the MINLPLib [12] and an IBM-CMU project on MINLP [13].

We will refer to these testsets as Miqp, Micp, and Minlp. In each of the following sections, detailed problem statistics are presented. 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" and "quad" show the number of linear and quadratic 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. In the tables with computational results, each entry shows the number of seconds to solve a problem, or the lower and upper bounds at termination, if the problem was not solved.

For our benchmark, we ran SCIP 1.1.0.10 using CPLEX 11.2.1 [17] as LP solver and Ipopt 3.6 [27] as QCP solver for the heuristics, see 4. For comparison, we ran CPLEX 11.2.1, BARON 8.1.5 [25], and LindoGlobal 5.0.1 [20]. 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 2.66 GHz Intel Core2 Quad CPU with 4 GB RAM and 4 MB Cache.

Mixed Integer Quadratic Programs. Table 4 presents the 24 instances from the Miqp testset [22]. Note that we consider the `clay*` instances in the Minlp testset, since they are not MIQPs.

We observe, that due to the reformulation (3.1), seven instances could be reformulated as mixed integer linear programs in the presolving state. For some instances, e.g. `ibell3a`, there is an increase in the number of variables and quadratic constraints. This is due to the disaggregation step (3.5).

Table 1 compares the performance of SCIP, BARON, and CPLEX on Miqp. We did not run LindoGlobal since many of the Miqp instances ex-

TABLE 1
Results on MIQP *instances.*

instance	SCIP	BARON	CPLEX
iair04	72.69	$[55577, \infty]$	**8.15**
iair05	43.35	$[25902, \infty]$	**22.29**
ibc1	**12.41**	$[0.8182, 6.411]$	24.28
ibell3a	15.55	$[-\infty, 878793]$	**8.14**
ibienst1	**23.91**	1221.23	709.88
icap6000	8.82	$[-2437987, -2429117]$	**4.69**
icvxqp1	93.86	$[1436360, \infty]$	**0.52**
ieilD76	15.61	$[682.4, 1488]$	**4.20**
ilaser0	15.78	fail	**0.00**
imas284	38.15	$[87604, 94665]$	**16.82**
imisc07	**49.71**	$[2019, \infty]$	155.23
imod011	485.54	30.67	**0.09**
inug06-3rd	84.13	$[7581, 8281]$	**0.81**
inug08	10.82	1661.85	**0.08**
iportfolio	$[-0.5253, \infty]$	$[\mathbf{-0.5248}, \mathbf{0}]$	$[-0.5253, \mathbf{0}]$
iqap10	845.77	$[319, 393.4]$	**40.82**
iqiu	**117.05**	$[-603.1, 1787]$	376.51
iran13x13	**127.19**	$[-\infty, 3535]$	168.40
iran8x32	112.25	$[4974, 5937]$	**65.68**
isqp0	54.92	$[-\infty, 1.40 \cdot 10^9]$	**0.78**
isqp1	$[-22016, -22002]$	$[-\infty, 1.45 \cdot 10^9]$	**25.41**
isqp	55.10	$[-\infty, 1.40 \cdot 10^9]$	**0.78**
iswath2	249.21	$[336.7, \infty]$	**21.88**
itointqor	0.02	$[-\infty, -1503]$	**0.00**
ivalues	0.02	0.08	**0.00**

ceed 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. Altogether, SCIP performs much better than BARON and slightly worse than CPLEX w.r.t. the number of solved instances of this testset. Although there are some examples which SCIP solves faster, CPLEX performs better w.r.t. average computation time.

Mixed Integer Conic Programs. The MICP testset consists of three types of optimization problems, see Table 5. The `classical_XXX_YY` instances contain only 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 instances `robust_XXX_YY` and `shortfall_XXX_YY` additionally contain a second-order cone constraint of dimension k. Due to its formulation as quadratic constraint with the term $(\beta y)^2$ on the right hand side of (3.2), it is categorized as nonconvex constraint. The large increase in the number of linear constraints is due to adding the linear relaxation (3.3) to the problem formulation, while the increase in the number of quadratic constraints is due to the disaggregation (3.5).

Table 2 compares the performance of SCIP, BARON, and LindoGlobal

Table 2
Results on Micp *instances*

instance	SCIP	BARON	LindoGlobal
classical_200_0	$[\mathbf{-0.1232}, \mathbf{-0.11}]$	$[-0.1257, -0.09588]$	$[-0.1257, -0.1077]$
classical_200_1	$[\mathbf{-0.1234}, \mathbf{-0.1167}]$	$[-0.1285, -0.1075]$	$[-0.1284, -0.1144]$
classical_40_0	**4.55**	101.59	44.19
classical_40_1	**1.84**	27.58	$[-0.08481, -0.08475]$
classical_50_0	**58.44**	1835.19	$[-0.09593, -0.09074]$
classical_50_1	**19.45**	149.72	$[-0.09632, -0.09476]$
robust_100_0	**1174.58**	$[-0.1048, -0.08674]$	$[-0.1542, -0.08404]$
robust_100_1	**389.04**	$[-0.07956, -0.06633]$	$[-0.1269, 0]$
robust_200_0	$[\mathbf{-0.1442}, \mathbf{-0.1411}]$	$[-0.1746, -0.07277]$	$[-0.2002, 0]$
robust_200_1	$[\mathbf{-0.1457}, \mathbf{-0.1427}]$	$[-0.1608, -0.1012]$	$[-0.1998, 0]$
robust_40_0	**6.22**	164.59	$[-0.07611, -0.07601]$
robust_40_1	**4.62**	83.79	$[-0.07652, -0.07646]$
robust_50_0	**2.14**	2046.31	139.74
robust_50_1	**8.88**	430.26	$[-0.08572, -0.08569]$
shortfall_100_0	$[\mathbf{-1.12}, \mathbf{-1.114}]$	$[-1.123, -1.112]$	$[-1.125, -1.113]$
shortfall_100_1	$[\mathbf{-1.109}, \mathbf{-1.106}]$	$[-1.112, -1.105]$	$[-1.112, \mathbf{-1.106}]$
shortfall_200_0	$[\mathbf{-1.147}, \mathbf{-1.124}]$	$[-1.15, -1]$	$[-1.161, -1.071]$
shortfall_200_1	$[\mathbf{-1.148}, \mathbf{-1.134}]$	$[-1.153, -1]$	$[-1.361, -1.079]$
shortfall_40_0	**52.70**	242.15	2550.00
shortfall_40_1	**15.43**	15.61	130.15
shortfall_50_0	**1222.03**	$[-1.101, -1.095]$	$[-1.103, -1.095]$
shortfall_50_1	**96.18**	278.38	$[-1.104, -1.102]$

on Micp. We observe that SCIP outperforms the other two solvers on this specific testset.

Mixed Integer Quadratically Constrained Programs. For the Minlp testset, we took 24 instances from the MINLPLib [12] and six convexified constrained layout problems (`clay*`) from [13].

The instances `lop97ic`, `lop97icx`, `pb302035`, `pb351535`, `qap`, and `qapw` were transformed into MIPs after presolving – which is due to the reformulation (3.1). The instances `nuclear*`, `space25`, `space25a`, and `waste` are 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, cut generation for nonconvex terms is not possible. Thus, if the propagation algorithm cannot find domain reductions for such unbounded variables, it may require many branching operations until meaningful variable bounds and a corresponding lower bound can be computed.

Table 3 compares the performance of SCIP, BARON, and LindoGlobal on Minlp. Figure 2 shows a performance profile for this testset. Again, SCIP performs better than BARON. Taking the number of solved instances into account, LindoGlobal slightly wins: it could solve one instance more than SCIP, which solved one instance more than BARON. Other compari-

TABLE 3
Results on MINLP *instances.*

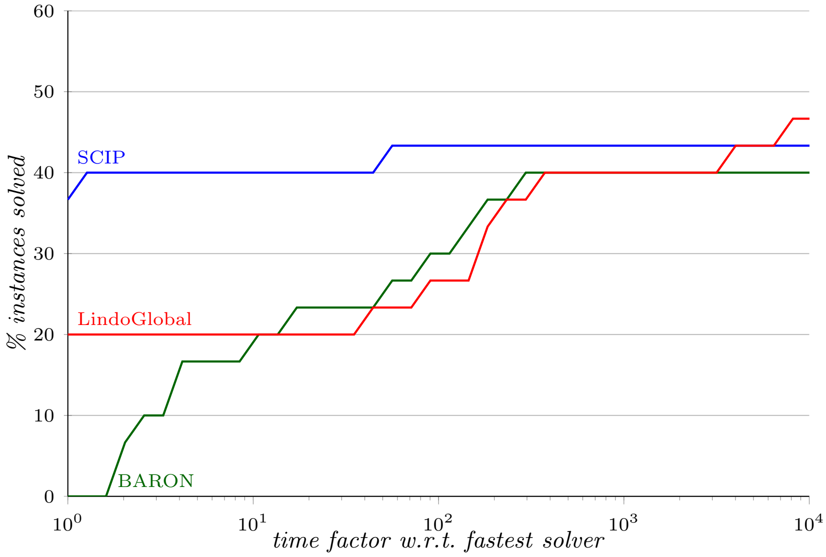
instance	SCIP	BARON	LindoGlobal
clay0203m	**0.18**	1.87	27.57
clay0204m	**2.78**	45.92	110.29
clay0205m	**12.07**	1057.00	1976.57
clay0303m	**0.75**	1.43	61.66
clay0304m	**4.39**	16.50	1007.32
clay0305m	**17.79**	$[6516, 8092]$	$[1320, 8093]$
du-opt5	**0.56**	142.28	1894.31
du-opt	**1.22**	142.37	453.35
lop97ic	$[\mathbf{2596}, \mathbf{4759}]$	$[2542, 8012]$	$[-\infty, \infty]$
lop97icx	$[3611, 4099]$	$[2739, 4377]$	**259.99**
nous1	$[1.006, 1.567]$	249.81	**110.12**
nous2	23.65	0.97	**0.50**
nuclear14a	$[-228.8, -1.101]$	$[\mathbf{-12.26}, \infty]$	$[\mathbf{-12.26}, \mathbf{-1.129}]$
nuclear14b	$[-197.9, \mathbf{-1.11}]$	$[\mathbf{-5.334}, \infty]$	$[-\infty, \infty]$
nuclear14	$[-\infty, \infty]$	$[-\infty, \infty]$	$[-\infty, \mathbf{-1.126}]$
nuclearva	$[-\infty, \infty]$	$[-\infty, \infty]$	$[-\infty, \mathbf{-1.011}]$
nvs19	**0.28**	14.24	2271.25
nvs23	**0.38**	60.12	$[-1570, -1125]$
pb302035	$[\mathbf{1228245}, 3804752]$	$[-\infty, \infty]$	abort
pb351535	$[\mathbf{1839388}, 4929561]$	$[1112854, 16612296]$	abort
product	**19.02**	fail	$[-2185, -2141]$
qap	$[\mathbf{88415}, 401300]$	$[40243, \mathbf{396014}]$	$[0, 396134]$
qapw	$[35610, 405328]$	$[264534, \mathbf{396172}]$	$[\mathbf{265684}, 398792]$
space25	$[73.01, \infty]$	$[83.01, 487.6]$	**166.13**
space25a	$[73.42, \infty]$	$[96.07, 501.2]$	$[\mathbf{233.6}, \mathbf{485}]$
tln12	$[16.26, \mathbf{91.4}]$	$[27.25, \infty]$	$[\mathbf{86}, 106.6]$
tln5	61.41	165.83	**50.14**
tln6	$[8.923, 15.3]$	$[13.72, 15.3]$	**185.18**
tln7	$[5.608, \mathbf{15}]$	$[12.5, 15.5]$	$[\mathbf{14.3}, 15.5]$
waste	$[\mathbf{346.8}, \mathbf{623.3}]$	$[-\infty, 712.3]$	$[0, 684.1]$

son criteria rather indicate a tie between SCIP and LindoGlobal which are both slightly better than BARON. SCIP is the fastest solver eleven times, LindoGlobal six times. There are five cases where SCIP has the best dual bound, versus five for LindoGlobal and two for BARON. Five times, SCIP has the best primal bound, compared to four times for LindoGlobal and two times for BARON. No solver, however, strictly dominates the others on this particular testset.

BARON wrongly declared the instance `product` to be infeasible and hit the time limit while parsing the instance pb302035. For the instances pb302035 and pb351535, LindoGlobal did not stop after 4000 seconds (using a 3600 seconds time limit) and did not report any bounds in the log file.

6. Conclusions. In this paper, we have shown how a framework for constraint integer programming can be extended towards a solver for general MIQCPs. We added the necessary methods to correctly handle the quadratic constraints, see Section 3.4. To speed up computations we fur-

Fig. 2. *Performance profile for MINLP testset.*



ther implemented MIQCP specific presolving, propagation, and separation methods, see Sections 3.1–3.3. Furthermore, we discussed two large neighborhood search heuristics for MIQCP, see Section 4. The computational results indicate that this already suffices to get a solver which is competitive to state-of-the-art solvers like CPLEX, BARON, and LindoGlobal. SCIP performed particularly well on the MIQP and MICP testsets, which contain a linear core that is complemented by a few quadratic constraints. This confirms our expectations, since SCIP already features several sophisticated MIP technologies.

We conclude that the extension of a full-scale MIP 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., advanced reformulations [18], more sophisticated separation routines [23], simplicial branching [19], and constraint handlers for specific types, e.g., bilinear covering constraints [24].

Acknowledgment. We like to thank Marc E. Pfetsch for contributing the implementation of the linear outer approximation for second-order cones and Ambros M. Gleixner for his valuable comments on the paper.

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] 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.

[4] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Waechter, *Branching and bounds tightening techniques for non-convex MINLP*, Tech. Rep. RC24620, IBM, 2008.

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

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

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

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

[9] 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.

[10] 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.

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

[12] 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 (2003), pp. 114–119.

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

[14] F. Domes and A. Neumaier, *Constraint propagation on quadratic constraints.* available online at `http://www.mat.univie.ac.at/~dferi/publ`, 2008.

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

[16] R. Horst and H. Tuy, *Global Optimization – Deterministic Approaches*, Springer, Berlin, 1990.

[17] ILOG, Inc., *CPLEX.* `http://www.ilog.com/products/cplex`.

[18] L. Liberti and C. Pantelides, *An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms*, J. Glob. Opt., 36 (2006), pp. 161–189.

[19] J. T. Linderoth, *A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs*, Math. Prog., 103 (2005), pp. 251–282.

[20] Lindo Systems, Inc., *LindoGlobal.* `http://www.lindo.com`.

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

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

[23] 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.

[24] 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.
[25] 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.
[26] J. P. Vielma, S. Ahmed, and G. L. Nemhauser, *A lifted linear programming branch-and-bound algorithm for mixed integer conic quadratic programs*, INFORMS Journal on Computing, 20 (2008), pp. 438–450.
[27] 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. Prog., 106 (2006), pp. 25–57.
[28] T. Yunes, I. D. Aron, and J. Hooker, *An integrated solver for optimization problems*, tech. rep., School of Business Administration, University of Miami, 2008.

APPENDIX

Table 4
Statistics of instances in Miqp *testset.*

instance	original problem					presolved problem					
	vars	int	bin	linear	quad	vars	int	bin	linear	quad	conv
iair04	8905	8904	0	823	1	7370	0	7370	601	0	✓
iair05	7196	7195	0	426	1	6115	0	6115	342	0	✓
ibc1	1752	252	0	1913	1	735	0	252	1045	0	✓
ibell3a	123	60	0	104	1	138	43	12	71	43	✓
ibienst1	506	28	0	576	1	477	28	0	520	28	✓
icap6000	6001	4099	1901	2171	1	5793	19	5740	1936	34	✓
icvxqp1	10001	10000	0	5000	1	19996	9998	0	4994	9998	✓
ieilD76	1899	1898	0	75	1	1823	0	1823	75	0	✓
ilaser0	1003	151	0	2000	1	1892	50	90	1030	901	✓
imas284	152	150	0	68	1	301	150	0	68	150	✓
imisc07	261	259	0	212	1	308	70	168	211	70	✓
imod011	10958	97	0	4480	1	20956	97	0	4480	10000	✓
inug06-3rd	2887	2886	0	3972	1	2886	0	2886	3972	0	✓
inug08	1633	1632	0	912	1	1632	0	1632	912	0	✓
iportfolio	1201	967	0	201	1	1400	967	0	201	200	✓
iqap10	4151	4150	0	1820	1	4150	0	4150	1820	0	✓
iqiu	841	48	0	1192	1	888	48	0	1192	48	✓
iran13x13	339	169	0	195	1	507	169	0	195	169	✓
iran8x32	513	256	0	296	1	768	256	0	296	256	✓
isqp0	1001	50	0	249	1	2000	50	0	249	1000	✓
isqp1	1001	100	0	249	1	2000	100	0	249	1000	✓
isqp	1001	50	0	249	1	2000	50	0	249	1000	✓
iswath2	6405	2213	0	483	1	6378	29	2184	482	29	✓
itointqor	51	50	0	0	1	100	50	0	0	50	✓
ivalues	203	202	0	1	1	404	202	0	1	202	✓

Table 5
Statistics of instances in Micp *testset.*

instance	original problem					presolved problem					
	vars	int	bin	linear	quad	vars	int	bin	linear	quad	conv
classical_40_0	120	0	40	82	1	159	0	40	82	40	✓
classical_40_1	120	0	40	82	1	159	0	40	82	40	✓
classical_50_0	150	0	50	102	1	199	0	50	102	50	✓
classical_50_1	150	0	50	102	1	199	0	50	102	50	✓
classical_200_0	600	0	200	402	1	799	0	200	402	200	✓
classical_200_1	600	0	200	402	1	799	0	200	402	200	✓
robust_40_0	163	0	41	124	2	904	0	41	1139	81	
robust_40_1	163	0	41	124	2	904	0	41	1138	81	
robust_50_0	203	0	51	154	2	1134	0	51	1429	101	
robust_50_1	203	0	51	154	2	1134	0	51	1429	101	
robust_100_0	403	0	101	304	2	2284	0	101	2879	201	
robust_100_1	403	0	101	304	2	2284	0	101	2879	201	
robust_200_0	803	0	201	604	2	4584	0	201	5779	401	
robust_200_1	803	0	201	604	2	4584	0	201	5779	401	
shortfall_40_0	164	0	41	125	2	1568	0	41	2153	82	
shortfall_40_1	164	0	41	125	2	1568	0	41	2141	82	
shortfall_50_0	204	0	51	155	2	1968	0	51	2705	102	
shortfall_50_1	204	0	51	155	2	1968	0	51	2705	102	
shortfall_100_0	404	0	101	305	2	3968	0	101	5455	202	
shortfall_100_1	404	0	101	305	2	3968	0	101	5455	202	
shortfall_200_0	804	0	201	605	2	7968	0	201	10955	402	
shortfall_200_1	804	0	201	605	2	7968	0	201	10955	402	

Table 6
Statistics of instances in Minlp *testset.*

instance	original problem					presolved problem					
	vars	int	bin	linear	quad	vars	int	bin	linear	quad	conv
clay0203m	30	0	18	30	24	51	0	15	27	48	✓
clay0204m	52	0	32	58	32	80	0	28	54	64	✓
clay0205m	80	0	50	95	40	115	0	45	90	80	✓
clay0303m	33	0	21	30	36	67	0	19	29	72	✓
clay0304m	56	0	36	58	48	102	0	34	57	96	✓
clay0305m	85	0	55	95	60	141	0	51	93	120	✓
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	72	0	2	13	52	
nous2	51	0	2	15	29	72	0	2	13	52	
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	1180	0	600	1790	0	✓
pb351535	526	0	525	50	1	1035	0	525	1580	0	✓
product	1553	0	107	1793	132	528	0	92	450	164	
qap	226	0	225	30	1	435	0	225	660	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	3048	0	576	48	2664	
nuclear14a	992	0	600	49	584	2808	0	600	2377	1800	
nuclear14b	1568	0	600	1225	560	2808	0	600	1225	1800	
nuclearva	351	0	168	50	267	1030	0	144	24	970	
tln12	168	156	12	60	12	301	144	24	85	132	
tln5	35	30	5	25	5	55	30	5	20	25	
tln6	48	42	6	30	6	78	42	6	24	36	
tln7	63	56	7	35	7	105	56	7	28	49	
waste	2484	0	400	623	1368	1238	0	400	516	1230	