Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TIMO BERTHOLD
AMBROS M. GLEIXNER

Undercover
a primal MINLP heuristic exploring a largest sub-MIP

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

ZIB-Report 12-07 (February 2012)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Undercover
a primal MINLP heuristic exploring a largest sub-MIP

Timo Berthold* Ambros M. Gleixner†

February 2, 2012

Abstract

We present Undercover, a primal heuristic for nonconvex mixed-integer nonlinear programming (MINLP) that explores a mixed-integer *linear* subproblem (sub-MIP) of a given MINLP. We solve a vertex covering problem to identify a minimal set of variables that need to be fixed in order to linearize each constraint, a so-called *cover*. Subsequently, these variables are fixed to values obtained from a reference point, e.g., an optimal solution of a linear relaxation. We apply domain propagation and conflict analysis to try to avoid infeasibilities and learn from them, respectively. Each feasible solution of the sub-MIP corresponds to a feasible solution of the original problem.

We present computational results on a test set of mixed-integer quadratically constrained programs (MIQCPs) and general MINLPs from MINLPLib. It turns out that the majority of these instances allow for small covers. Although general in nature, the heuristic appears most promising for MIQCPs, and complements nicely with existing root node heuristics in different state-of-the-art solvers.

Keywords: Primal Heuristic, Mixed-Integer Nonlinear Programming, Large Neighborhood Search, Mixed-Integer Quadratically Constrained Programming, Nonconvex Optimization
Mathematics Subject Classification: 90C11, 90C20, 90C26, 90C30, 90C59

1 Introduction

For mixed-integer (linear) programming it is well-known that, apart from complete solving methods, general purpose primal heuristics like the feasibility pump [4, 19, 21] are able to find high-quality solutions for a wide range of problems. Over the years, primal heuristics have become a substantial ingredient of state-of-the-art solvers for mixed-integer programming [6, 10]. For mixed-integer nonlinear programming (MINLP), the last three years saw an increasing interest of the research community in general-purpose primal heuristics [8, 9, 12, 13, 16, 30, 32, 33].
An MINLP is an optimization problem of the form

$$\begin{array}{lll} \min & d^{\mathsf{T}}x & \\ \text{s.t.} & g_k(x) \leqslant 0 & \text{for } k = 1, \ldots, m, \\ & L_i \leqslant x_i \leqslant U_i & \text{for } i = 1, \ldots, n, \\ & x_i \in \mathbb{Z} & \text{for } i \in \mathcal{I}, \end{array} \tag{1}$$

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, berthold@zib.de
†Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, gleixner@zib.de

where $\mathcal{I} \subseteq \{1, \ldots, n\}$ is the index set of the integer variables, $d \in \mathbb{R}^n$, $g_k : \mathbb{R}^n \to \mathbb{R}$ for $k = 1, \ldots, m$, and $L \in (\mathbb{R} \cup \{-\infty\})^n$, $U \in (\mathbb{R} \cup \{+\infty\})^n$ are lower and upper bounds on the variables, respectively. Since fixed variables can always be eliminated, we assume w.l.o.g. that $L_i < U_i$ for $i = 1, \ldots, n$, i.e., the interior of $[L, U]$ is nonempty. Note that a nonlinear objective function can always be reformulated by introducing one additional constraint and variable, hence form (1) is general.

If all constraint functions g_k are quadratic we call (1) a *mixed-integer quadratically constrained program* (MIQCP). If all constraints are linear we call (1) a *mixed-integer program* (MIP). If $\mathcal{I}$ is empty, we refer to an MINLP, MIQCP, and MIP as a *nonlinear program* (NLP), *quadratically constrained program* (QCP), and *linear program* (LP), respectively.

At the heart of many recently proposed primal MIP heuristics, such as Local Branching [20], RINS [17], DINS [23], and RENS [7], lies large neighborhood search, the paradigm of solving a small sub-MIP that promises to contain good solutions. In this paper, we introduce *Undercover*, a large neighborhood search start heuristic that constructs and solves a sub-MIP of a given MINLP. We demonstrate its effectiveness on a general test set of MIQCPs taken from the MINLPLib [14]. During the design of Undercover, our focus was its application as a start heuristic inside a complete solver such as BARON [34], Couenne [5], Bonmin [11] or SCIP [3].

When primal heuristics are considered as standalone solving procedures, e.g., the RECIPE heuristic [30] or the Feasibility Pump [12, 16], the algorithmic design typically aims at finding feasible solutions for as many instances as possible, even if this takes substantial running time. However, if they are used as supplementary procedures inside a complete solver, the overall solver performance is the main objective. To this end, it is often worth sacrificing success on a small number of instances for a significant saving in average running time. Primal heuristics in modern solvers therefore often follow a "fast fail" strategy: the most crucial decisions are taken in the beginning and in a defensive fashion such that if the procedure aborts, it will not have consumed much running time.

Two major features distinguish Undercover from all mentioned primal heuristics for MINLP. Firstly, unlike most of them [8, 12, 13, 16, 33], Undercover is not an extension of an existing MIP heuristic towards a broader class of problems; moreover, it does not have a counterpart in mixed-integer linear programming. Secondly, Undercover solves two auxiliary MIPs (one for finding a set of variables to be fixed plus the resulting sub-MIP), and at most two NLPs (possibly one to compute initial fixing values, one for postprocessing the sub-MIP solution). To the contrary, most large neighborhood search heuristics [12, 16, 30, 32, 33] for MINLP solve an arbitrarily large series of MIPs, often alternated with a sequence of NLPs, to produce a feasible start solution. The number of iterations is typically not fixed, but depends on the instance at hand.

The paper is organized as follows. Section 2 introduces a first generic version of the Undercover algorithm. In Section 3, we describe how to find variables to fix such that the resulting subproblem is linear. Section 4 explains how to extract useful information, even if the sub-MIP proves to be infeasible. Finally, Section 6 provides computational results that show the effectiveness of Undercover.

2 A generic algorithm

The paradigm of fixing a subset of the variables of a given mixed-integer program in order to obtain subproblems that are easier to solve has proven successful in many primal MIP heuristics such as RINS [17], DINS [23], and RENS [7]. The core difficulty in MIP solving is the integrality constraints. Thus, in MIP context, "easy to solve" usually takes the meaning of few integer variables.

Actually, integrality is a special case of nonlinearity, since it is possible to model the integrality of a bounded integer variable $x_i \in \{L_i, \ldots, U_i\}$ by the nonconvex polynomial constraint $(x_i - L_i) \cdot \ldots \cdot (x_i - U_i) = 0$. This insight matches the practical experience that in MINLP, while integralities do contribute to the complexity of the problem, the specific difficulty is the nonlinearities. Hence, "easy" in an MINLP context can be understood as few nonlinear constraints.
Our heuristic is based on the simple observation that by fixing certain variables (to some value within their bounds) any given mixed-integer *nonlinear* program can be reduced to a mixed-integer *linear* subproblem (sub-MIP). Every feasible solution of this sub-MIP is then a feasible solution of the original MINLP.
Whereas in general it holds that many or even all of the variables might need to be fixed in order to arrive at a linear subproblem, our approach is motivated by the experience that for several practically relevant MINLPs fixing only a comparatively small subset of the variables already suffices to linearize the problem. The computational effort of solving this subproblem compared to solving the original problem, however, is usually greatly reduced since we can apply the full strength of state-of-the-art MIP solving. Before formulating a first generic algorithm for our heuristic, consider the following definitions.

Definition 2.1. *Let P be an MINLP of form (1) and $\mathcal{C} \subseteq \{1, \ldots, n\}$ be a set of variable indices of P. We call $\mathcal{C}$ a* cover *of constraint g_k, $k \in \{1, \ldots, m\}$, if and only if the set*

$$\{(x, g_k(x)) : x \in [L, U], x_i = x_i^* \text{ for all } i \in \mathcal{C}\} \tag{2}$$

is affine for all $x^* \in [L, U]$. We call $\mathcal{C}$ a cover *of P if and only if $\mathcal{C}$ is a cover of all constraints $g_1, \ldots, g_m$.*

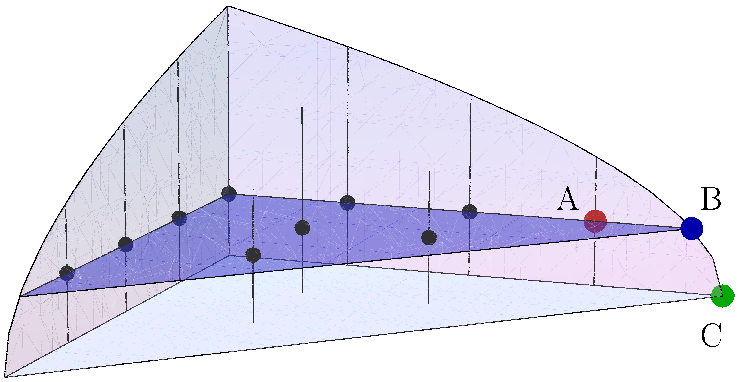

Figure 1: A convex MIQCP and the Undercover sub-MIP induced by the NLP relaxation.

The following example illustrates how covers of an MINLP are used to construct a sub-MIP for finding feasible solutions.

Example 2.2 (the Undercover sub-MIP)**.** Consider the following convex MIQCP:

$$\begin{aligned} \min \quad & -y - z \\ \text{s.t.} \quad & x + y + z^2 - 4 \leqslant 0 \\ & x, y, z \geqslant 0 \\ & x, y \in \mathbb{Z} \end{aligned} \tag{3}$$

The only variable that appears in a nonlinear term is z, hence $\{z\}$ is the minimum cover of (3) w.r.t. the above definition. The (unique) optimal solution of its nonlinear relaxation is $(0, 3.75, 0.5)$ with objective function value -4.25.
Taking that relaxation, the idea of Undercover is to fix $z = 0.5$, which makes (3) an integer linear program. The (unique) optimal solution of this is $(0, 3, 0.5)$ with objective function value -3.5, which is necessarily a feasible solution for the MIQCP (3). Taking the NLP as dual and the Undercover solution as primal bound, this gives an optimality gap of roughly 20%. The actual (unique) optimal solution of (3) is $(0, 4, 0)$.
This example is illustrated in Figure 1. The light shaded region shows the solid corresponding to the NLP relaxation; the parallel lines show the mixed-integer set of feasible solutions of (3). The dark shaded area shows the polytope associated to the Undercover sub-MIP. The blue point B is the optimum of the NLP, the red point A is the optimum of the Undercover sub-MIP, the green point C is the optimum of the MIQCP. The smaller black points indicate further feasible solutions of the Undercover sub-MIP.

A first generic algorithm for our heuristic is given in Figure 2. The clear hinge of the algorithm is found in line 5—finding a suitable cover of the given MINLP. Section 3 elaborates on this in detail.

```
1 input MINLP P as in (1)
2 begin
3 |   compute a solution x* of an approximation or relaxation of P
4 |   round x*_i for i ∈ I
5 |   determine a cover C of P
6 |   solve the sub-MIP of P given by fixing x_i = x*_i for all i ∈ C
7 end
```

Figure 2: Simple generic algorithm.

To obtain suitable fixing values for the selected variables, an approximation or relaxation of the original MINLP is used. For integer variables the approximate values are rounded. Most complete solvers for MINLP are based on branch-and-bound [29]. If the heuristic is embedded within a branch-and-bound solver, using its (linear or nonlinear) relaxation appears as a natural choice for obtaining approximate variable values.
Large neighborhood search heuristics that rely on fixing variables typically have to trade off between eliminating many variables in order to make the sub-MIP tractable and leaving enough degrees of freedom such that the sub-MIP is still feasible and contains good solutions. Often their implementation inside a MIP solver demands a sufficiently large percentage of variables to be fixed to arrive at an easy to solve sub-MIP [6, 7, 17, 23].
For our heuristic, the situation is different since we do not aim to eliminate integrality constraints, but nonlinearities. While it still holds that fixing variables, even only few, results in a smaller search space, the main benefit is that we arrive at a MIP.
In a nutshell: instead of solving an easier problem of the same class, we solve a smaller problem of an easier class.
In order to linearize a given MINLP, in general we may be forced to fix integer and continuous variables. The fixing of continuous variables, especially, in an MINLP can introduce a significant error, even rendering the subproblem infeasible. Thus our heuristic will aim at fixing as *few* variables as possible to obtain as large a linear subproblem as possible, through the utilization of minimum covers.

3 Finding minimum covers

This section describes our method for determining a minimum cover of an MINLP, i.e., a minimal subset of variables to fix in order to linearize each constraint. In this section, we make the standard assumption that the nonlinear functions involved are twice continuously differentiable. However, the idea of Undercover can easily be applied to MINLPs in general, as will be explained in Section 7. Note that the partial derivatives are well-defined since the domain $[L, U]$ has nonempty interior.

As motivation let us first consider *bilinear programs*, i.e., QCPs with a bipartition of their variables, $\{1, \dots, n\} = A \cup B$, $A \cap B = \emptyset$, and each quadratic term of the form $x_i x_j$, $i \in A$, $j \in B$. In this case, holding the variables of either A or B fixed, per definition one obtains a linear program—a simple property that has been used extensively in various solution approaches. The sets A and B each constitutes a cover. In the global optimization literature, the variables corresponding to the smaller of both sets are often called complicating variables. If the partition into A and B is unique, then these complicating variables form a minimum cover.

Any general quadratically constrained program can be converted to a bilinear program by duplication of variables. Hansen and Jaumard [26] showed how to perform this transformation such as to minimize either the number of duplicated or complicating variables. The following definition is a straightforward generalization of their notion of a *co-occurrence graph* to MINLPs.

Definition 3.1 (co-occurrence graph)**.** *Let P be an MINLP of form (1) with $g_1, \dots, g_m$ twice continuously differentiable on $[L, U]$. We call $G_P = (V, E)$ the* co-occurrence graph *of P with node set $V_P = \{1, \dots, n\}$ given by the variable indices of P and edge set*

$$E_P = \big\{ij \mid i, j \in V, \exists k \in \{1, \dots, m\} : \frac{\partial^2}{\partial x_i \partial x_j} g_k(x) \not\equiv 0\big\},$$

i.e., we draw an edge between i and j if and only if the Hessian matrix of some constraint has a structurally nonzero entry (i, j).

Remark 3.2. *Since the Hessian of a twice continuously differentiable function is symmetric, G_P is a well-defined, undirected graph. It may contain loops, e.g., if square terms x_i^2 are present. Trivially, the co-occurrence graph of a bilinear program is bipartite; the co-occurrence graph of a MIP is an edge-free graph.*

Theorem 3.3. *Let P be an MINLP of form (1) with $g_1, \dots, g_m$ twice continuously differentiable on $[L, U]$. Then $\mathcal{C} \subseteq \{1, \dots, n\}$ is a cover of P if and only if it is a vertex cover of the co-occurrence graph G_P.*

Proof. If $h : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, $x_i^* \in \mathbb{R}^n$, $\mathcal{C} \subseteq \{1, \dots, n\}$, then fixing variables $x_i = x_i^*$, $i \in \mathcal{C}$, and projecting to the nonfixed variables yields another twice continuously differentiable function $\bar{h} : \mathbb{R}^{n-|\mathcal{C}|} \to \mathbb{R}$. Let $\pi : \mathbb{R}^n \to \mathbb{R}^{n-|\mathcal{C}|}$ be the projection $x \mapsto (x_i)_{i \notin \mathcal{C}}$. Now the Hessian matrix of $\bar{h}$ is simply obtained from the Hessian of h by taking the columns and rows of nonfixed variables:

$$\nabla^2 \bar{h}_{\pi(x)} = \Big(\frac{\partial^2}{\partial x_i \partial x_j} h(x)\Big)_{i,j \notin \mathcal{C}}$$

for any $x \in \mathbb{R}^n$ with $x_i = x_i^*$, $i \in \mathcal{C}$. A twice continuously differentiable function is affine if and only if its Hessian vanishes on its domain. Hence,

$$\mathcal{C} \text{ is a cover of } h \Leftrightarrow \forall i, j \notin \mathcal{C} : \frac{\partial^2}{\partial x_i \partial x_j} h(x) \equiv 0.$$

For the MINLP P this yields that

$$\begin{aligned}\mathcal{C} \text{ is a cover of } P &\Leftrightarrow \forall i,j \notin \mathcal{C}, k \in \{1,\dots,m\} : \frac{\partial^2}{\partial x_i \partial x_j} g_k(x) \equiv 0 \\ &\Leftrightarrow \forall i,j \notin \mathcal{C} : ij \notin E_P \\ &\Leftrightarrow \forall ij \in E_P d : i \in \mathcal{C} \vee j \in \mathcal{C},\end{aligned}$$

i.e., if and only if $\mathcal{C}$ is a vertex cover of the co-occurrence graph G_P. □

Note that any undirected graph $G = (V, E)$ is the co-occurrence graph of the QCP $\min\{0 : x_i x_j \leqslant 0 \text{ for all } ij \in E\}$. Hence, minimum vertex cover can be transformed to computing a minimum cover of an MINLP. Since minimum vertex cover is $\mathcal{NP}$-hard [22], we have

Corollary 3.4. *Computing a minimum cover of an MINLP is $\mathcal{NP}$-hard.*

There exist, however, many polynomial-time algorithms to approximate a minimum vertex cover within a factor of 2, such as simply taking the vertices of a maximal matching. It is conjectured that 2 is also the optimal approximation factor [28] and it is proven that vertex cover is $\mathcal{NP}$-hard to approximate within a factor smaller than $10\sqrt{5}-21 = 1.3606\dots$ [18], hence no polynomial-time approximation scheme exists. Approximation ratios $2-\epsilon(G)$ are known with $\epsilon(G) > 0$ depending on special properties of the graph such as number of nodes [27] or bounded degree [25].
In this paper we aim at computing minimum covers exactly. For this we use a simple binary programming formulation. For an MINLP of form (1), define auxiliary binary variables α_i, $i = 1,\dots,n$, equal to 1 if and only if the original variable x_i is fixed. Then

$$\mathcal{C}(\alpha) \coloneqq \{i \in \{1,\dots,n\} : \alpha_i = 1\}$$

forms a cover of P if and only if $\alpha_i + \alpha_j \geqslant 1$ for all $ij \in E_P$. For an MIQCP, e.g., this requires all square terms and at least one variable in each bilinear term to be fixed. To obtain as large a linear subproblem as possible, we solve the binary program

$$\min\Big\{\sum_{i=1}^{n} \alpha_i : \alpha_i + \alpha_j \geqslant 1 \text{ for all } ij \in E_P, \alpha \in \{0,1\}^n\Big\} \tag{4}$$

minimizing the sum of auxiliary variables.
Note that for particular classes of MINLPs it is possible to exploit special features of the co-occurrence graph in order to compute a minimum cover exactly in polynomial time—a simple example is the class of bilinear programs mentioned above—or to approximate it within a factor sufficiently close to 1. However, in our experiments the binary program (4) could always be solved by a standard MIP solver within a fraction of a second. In all cases, optimality was proven at the root node, hence without enumeration.

4 Domain propagation and conflict learning

Fixing a variable can have great impact on the original problem and the approximation we use. An important detail, crucial for the success rate of Undercover, is not to fix the variables in the cover simultaneously, but sequentially one by one. This section describes how we use domain propagation, backtracking and conflict analysis to avoid and handle infeasibilities during this process.

Fix-and-propagate. The task of *domain propagation* is to analyze the individual structure of single constraints w.r.t. the current domains of the variables in order to infer additional domain reductions, thereby tightening the search space. For an overview of domain propagation techniques applied in MIP and MINLP solvers, see [3] and [35], respectively.
To prevent obvious infeasibilities, we fix the variables in the cover one after the other applying domain propagation after each fixing in order to further tighten the bounds, in particular of the yet unfixed cover variables. During this process, it might happen that the value a variable takes in the reference solution is no longer contained in its reduced domain. In this case, we instead fix the variable to the closest bound.[1] This fix-and-propagate procedure resembles a method described in [21]. Additionally, we apply it for continuous variables.
Note that by this, the fixing values and hence the created subproblem depend on the fixing order. Different variable orderings lead to different propagations, thereby to different subproblems and different solutions being found.
Of course, it might also happen that a variable domain becomes empty. This means that the subproblem with the currently chosen fixing values is proven to be infeasible without even having started its solution procedure.
In this case, we apply a one-level backtracking, i.e., we undo the last bound change and try alternative fixing values, see Section 5 for details. Note that if we cannot resolve the infeasibility by one-level backtracking, Undercover will terminate. This is a "fast fail" strategy: if we cannot easily resolve the infeasibility, we abort at an early stage of the algorithm without wasting running time.[2]
Even if fix-and-propagate runs into an infeasibility, we can extract useful information for the global solution process. Adding so-called conflict constraints prevents running into the same deadlock twice.

Conflict analysis in MIP. Conflict learning is a technique that analyzes infeasible subproblems encountered during a branch-and-bound search. Whenever a subproblem is infeasible, conflict analysis can be used to learn one (or more) reasons for this infeasibility. This gives rise to so called conflict constraints that can be exploited in the remainder of the search to prune other parts of the tree.
Carefully engineered conflict analysis has lead to a substantial increase in the size of problems modern SAT solvers can deal with [31]. It has recently been generalized to MIP [1, 2]. One main difference between MIP and SAT solving in the context of conflict analysis is that the variables of a MIP do not need to be of binary type. In [1] it is shown how the concept of a conflict graph can be extended to MIPs with general integer and continuous variables.
The most successful SAT learning approaches use so called *first unique implication point (1UIP)* learning, which captures a conflict that is "close" to the infeasibility and can infer new information. Solvers for MIP or MINLP typically take a longer processing time per node and they do not restart during search. That is why MIP solvers with conflict learning such as SCIP potentially generate several conflicts for each infeasibility.

Conflict analysis for Undercover. The fix-and-propagate strategy can be seen as a simulation of a depth-first-search in the branch-and-bound tree, applying one-level backtracking when a fixing results in an infeasible subproblem. Hence, using conflict analysis for these partially fixed, infeasible subproblems enables us to learn constraints that are valid for the global search of the

[1] Alternatively, we could recompute the reference solution to obtain values within the current bounds.

[2] If we want to apply Undercover more aggressively, we can try to recover from infeasibility by reordering the fixing sequence, e.g., such that the variable for which the fixing failed will be the first one in the reordered sequence. This resembles a simple restarting mechanism from SAT solving.

original MINLP. This is done by building up the conflict graph that is implied by the variable fixings and the propagated bound changes. For this, the reason for each propagation, i.e., the bounds of other variables that implied the domain reduction, needs to be stored or reconstructed later on.
Note that the generated conflict constraints will not be limited to the variables in the cover since the conflict graph also contains all variables that have changed their bounds due to domain propagation in the fix-and-propagate procedure.
Valid constraints can be learned even after fix-and-propagate. If the subsequent sub-MIP solution process proves infeasibility and all variables in the cover are integer, we may forbid the assignment made to the cover variables for the global solution process. The same constraint can be learned if the Undercover sub-MIP could be solved to proven optimality, since the search space that is implied by these fixings has been fully explored. In both cases, this is particularly useful for small covers.

5 The complete algorithm

This section outlines the details of the complete Undercover algorithm, cf., Figure 3. In the first step, we construct the covering problem (4) by collecting the edges of the co-occurrence graph, see Section 3. For constraints of simple form such as quadratic ones the sparsity pattern of the Hessian matrix can be read directly from the description of the constraint function. For general nonlinearities, we use *algorithmic differentiation* to automatically compute the sparsity pattern of the Hessian, see, e.g., [24].
To solve the covering problem we employ a standard MIP solver, which in our computational experiments never took more than a fraction of a second to find an optimal cover. Nevertheless, since the covering problem is $\mathcal{NP}$-hard, solving it to optimality may be time-consuming, in general. To safeguard against this, we only solve the root node and proceed with the best solution found. Subsequently, we fix the variables in the computed cover as described in Section 4.[3]
As motivated in the beginning, we designed Undercover to be applied within a complete solver. During fix-and-propagate, we call two routines provided by the solver, domain propagation in line 23 and conflict analysis in line 27. If the former detects infeasibility, we call the latter to learn conflict constraints for the global solution process, see Section 4.
If domain propagation detects infeasibility after fixing variable x_i, $i \in C$, to the (rounded and projected) value X_i in the reference solution, we try to recover by one-level backtracking. The following alternatives will be tried: for binary variables the value $1 - X_i$; for nonbinary variables the lower bound L_i and, if this is also infeasible, the upper bound U_i. In the case of infinite bounds L_i and U_i are replaced by $X_i - |X_i|$ and $X_i + |X_i|$, respectively. If $X_i = 0$, then -1 and $+1$ will be used instead. Of course, if fixing values accidentally coincide, each value is tested only once.
Typically, the sub-MIP solved in the next step incurs the highest computational effort and is controlled by work limits on the number of nodes, LP iterations, etc., see Section 6 for details. Since by construction the sub-MIP should be significantly easier than the original MINLP, we

[3] If we want to apply Undercover aggressively and allow for solving the covering problem multiple times, the following two strategies can be used. First, during the fix-and-propagate routine variables outside the precomputed cover may be fixed simultaneously. In this case, the fixing of some of the yet unfixed variables in the cover might become redundant. Recomputing the cover with $\alpha_i = 1$ for all i with local bounds $\hat{L}_i = \hat{U}_i$ may yield a smaller number of remaining variable fixings. Second, if no feasible fixings for the cover variables are found later on, we can re-solve the covering problem adding a cutoff constraint $\sum_{i \in \mathcal{C}} (1 - \alpha_i) + \sum_{i \notin \mathcal{C}} \alpha_i \geqslant 1$ and try again. However, both techniques appear to be computationally too expensive for the standard setting that we explored in our computational experiments.

input MINLP as in (1), reference point $x^* \in [L, U]$, $n_i \geqslant 0$ alternative fixing values $y^*_{i,1}, \ldots, y^*_{i,n_i} \in [L_i, U_i]$ for all $i \in \{1, \ldots, n\}$
output feasible solution $\hat{x}$ (on success)

begin

`/* Step 1: create covering problem */`

1 $E \leftarrow \emptyset$ `/* edge set of co-occurrence graph */`

2 **foreach** $k \in \{1, \ldots, m\}$ **do**

3 $S_k \leftarrow \{i \in \{1, \ldots, n\} : g_k$ depends on $x_i\}$ `/* variables in` $g_k(x) \leqslant 0$ `*/`

4 **foreach** $i \in S_k$ **do**

5 **if** $\frac{\partial^2}{\partial x_i^2} g_k(x) \not\equiv 0$ **then** $E \leftarrow E \cup \{(i, i)\}$ `/* must fix` x_i `*/`

6 **else**

7 **foreach** $j \in S_k, j > i, \frac{\partial^2}{\partial x_i \partial x_j} g_k(x) \not\equiv 0$ **do**

8 $E \leftarrow E \cup \{(i, j)\}$ `/* must fix` x_i `or` x_j `*/`

`/* Step 2: solve covering problem` (4) `*/`

9 $\alpha^* \leftarrow \arg\min \left\{ \sum_{i=1}^n \alpha_i : \alpha_i + \alpha_j \geqslant 1 \text{ for all } ij \in E, \alpha \in \{0, 1\}^n \right\}$

10 $\mathcal{C} \leftarrow \{i \in \{1, \ldots, n\} : \alpha^*_i = 1\}$

`/* Step 3: fix-and-propagate loop */`

11 $\hat{L} \leftarrow L$, $\hat{U} \leftarrow U$ `/* local bounds */`

12 **foreach** $i \in \mathcal{C}$ **do**

13 $\hat{L}^0 \leftarrow \hat{L}$, $\hat{U}^0 \leftarrow \hat{U}$, $p \leftarrow 0$ `/* store bounds for backtracking */`

14 $\mathcal{X} \leftarrow \emptyset$, *success* $\leftarrow$ false `/* set of failed fixing values */`

15 **while** $\neg$*success* **and** $p \leqslant n_i$ **do**

16 $X_i \leftarrow$ **if** $p = 0$ **then** x^*_i **else** $y^*_{i,p}$

17 **if** $i \in \mathcal{I}$ **then** $X_i \leftarrow [X_i]$ `/* round if variable integer */`

18 $X \leftarrow \min\{\max\{X_i, \hat{L}_i\}, \hat{U}_i\}$ `/* project to bounds if outside */`

19 **if** $X_i \in \mathcal{X}$ **then**

20 $p \leftarrow p + 1$ `/* skip fixing values tried before */`

21 **else**

22 $\hat{L}_i \leftarrow X_i$, $\hat{U}_i \leftarrow X_i$ `/* fix */`

23 call domain propagation on $[\hat{L}, \hat{U}]$ `/* propagate */`

24 **if** $[\hat{L}, \hat{U}] \neq \emptyset$ **then**

25 *success* $\leftarrow$ true `/* accept fixing, go to next variable */`

26 **else**

27 call conflict analysis

28 $\hat{L} \leftarrow \hat{L}^0$, $\hat{U} \leftarrow \hat{U}^0$ `/* infeasible: backtrack */`

29 $\mathcal{X} \leftarrow \mathcal{X} \cup \{X_i\}$, $p \leftarrow p + 1$ `/* try next fixing value */`

30 **if** $\neg$*success* **then return** `/* no feasible fixing found: terminate */`

`/* Step 4: solve sub-MIP */`

31 solve sub-MIP $\min \left\{ d^\mathsf{T} x : g_k(x) \leqslant 0 \text{ for } k = 1, \ldots, m, \; \hat{L}_i \leqslant x_i \leqslant \hat{U}_i \text{ for } i = 1, \ldots, n, \; x_i \in \mathbb{Z} \text{ for } i \in \mathcal{I} \right\}$

32 **if** *sub-MIP solved to optimality or proven infeasible* **and** $\mathcal{C} \subseteq \mathcal{I}$ **then**

33 add constraint $\bigvee_{i \in \mathcal{C}} (x_i \neq X_i)$ to original problem

`/* Step 5: solve sub-NLP */`

34 **if** *feasible sub-MIP solution found* **then**

35 $\hat{x} \leftarrow$ best sub-MIP solution

36 **if** *sub-MIP not solved to optimality* **or** $\mathcal{C} \not\subseteq \mathcal{I}$ **then**

`/* restore global bounds, fix integers, solve locally */`

37 solve sub-NLP $\min \left\{ d^\mathsf{T} x : g_k(x) \leqslant 0 \text{ for } k = 1, \ldots, m, \; L_i \leqslant x_i \leqslant U_i \text{ for } i = 1, \ldots, n, \; x_i = \hat{x}_i \text{ for } i \in \mathcal{I} \right\}$

38 $\hat{x} \leftarrow$ sub-NLP solution `/* update sub-MIP solution */`

39 **return** $\hat{x}$

end

Figure 3: The complete Undercover algorithm.

expect that often it can indeed be solved to optimality or proven infeasible. As described in Section 4, we may then forbid the assignment of fixing values to the cover variables if the latter are all integer, stated in line 33.
Eventually, if a feasible sub-MIP solution $\hat{x}$ has been found, we try to improve it further by fixing all integer variables to their values in $\hat{x}$ and solving the resulting NLP to local optimality. Clearly, if all cover variables are integer and $\hat{x}$ is optimal for the sub-MIP, this step can be skipped. Otherwise, we reoptimize over the continuous variables in the cover and may obtain a better objective value.

6 Computational experiments

Only few solvers exist that handle nonconvex MINLPs, such as BARON [34], COUENNE [5], and LindoGlobal [40]. Others, e.g., BONMIN [11] and SBB [41], guarantee global optimality only for convex problems, but can be used as heuristic solvers for nonconvex problems. Recently, the solver SCIP [2, 3] was extended to solve nonconvex MIQCPs [9] and MINLPs [35] to global optimality. For a comprehensive survey of available MINLP solver software, see [15].

Experimental setup. The target of our computational experiments was to analyze the performance of Undercover as a start heuristic for MINLPs applied at the root node. Therefore, we evaluated the sizes of the actual covers found, the success rate of Undercover, the distribution of running time among different components of the algorithm, and benchmarked against state-of-the-art solvers.
We implemented the algorithm given in Figure 3 within SCIP[4] and used SCIP's LP solution as reference point x^*. To perform the fix-and-propagate procedure, we called the standard domain propagation engine of SCIP. Secondary SCIP instances were used to solve both the covering problem (4) and the Undercover sub-MIP.
We controlled the computational effort for solving the sub-MIP in two ways. First, we imposed a hard limit of 500 nodes and a dynamic stall node limit[5] between 1 and 500 nodes. Second, we adjusted the SCIP settings to find feasible solutions fast: we disabled expensive presolving techniques and used the "primal heuristics emphasis aggressive" and the "emphasis feasibility" settings. Furthermore, if the sub-MIP is infeasible, this is often detected already when solving the root relaxation, hence we deactivated expensive pre-root heuristics so as to not lose time on such instances. Components using sub-MIPs themselves are switched off altogether. For details, please refer to the source code at [38].
In our main experiment, we ran SCIP with all heuristics other than Undercover switched off and cut generation deactivated. We used SCIP 2.1.1 with CPLEX 12.3 [39] as LP solver, IPOPT 3.10 [36] as NLP solver for the postprocessing, and CPPAD 20100101.4 [37] as expression interpreter for evaluating general nonlinear constraints. We refer to this configuration as UC.
We tested against the state-of-the-art MINLP solvers BARON 9.3.1 [34] (commercial license), COUENNE 0.3 [5] (open source), and SCIP 2.1.1 (academic license) with Undercover disabled. In order to investigate how Undercover can enhance the root node performance of complete solvers, we compare UC with their root heuristics. SCIP, for instance, applies eleven primal heuristics at the root node.
As test set we used a selection of 37 MIQCP instances from MINLPLib [14]. We excluded `lop97ic`, `lop97icx`, `pb302035`, `pb351535`, `qap`, and `qapw`, which are linear after the default pre-

[4]The source code is publicly available within SCIP 2.1.1 and can be found at [38].
[5]With a stall node limit we terminate if no improving solutions are found within a certain number of branch-and-bound nodes since the discovery of the current incumbent.

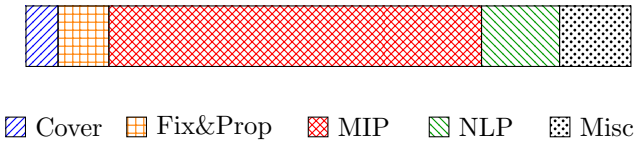

Figure 4: Distribution of running time among different components of Undercover heuristic.

solving of SCIP. On the `nuclear` instances, the root LP relaxation of SCIP is often unbounded due to unbounded variables in nonconvex terms of the constraints. In this case, we cannot apply Undercover since no fixing values are available. Due to this, we only included two of those instances, `nuclear14a` and `nuclear14b`, for which the root LP of SCIP is bounded.
We further tested Undercover on general MINLPs from MINLPLib, excluding those which are MIQCPs, linear after SCIP presolving, or contain expressions that cannot be handled by SCIP, e.g., sin and cos. Additionally, three more instances with unbounded root LP relaxation were removed, leaving 110 instances. We used the same settings and solvers as described above.
All experiments were conducted on a 3.00 GHz Intel Core 2 Extreme CPU X9650 with 6144 KB Cache and 8 GB RAM using openSuse 11.4 with compiler GCC 4.5.1. Hyperthreading and Turboboost were disabled.

Results for MIQCP. The results for the experiments on MIQCPs are shown in Table 1. In columns "% cov" and "% nlcov", we report the relative size of the cover used by UC as percentage of the total number of variables and of the number of variables that appear in at least one nonlinear term, respectively. All numbers are calculated w.r.t. the numbers of variables after preprocessing. Note that a value of 100% in the "% nlcov" column means that the trivial cover consisting of all variables appearing in nonlinear terms is already minimal. For all other instances, the solution of the covering problem gives rise to a smaller cover, hence a larger sub-MIP and potentially more solutions for the MINLP.
Column "UC" shows the objective value of the best solution found by Undercover. For all other solvers, we provide the objective value of the best solution found during root node processing. The best objective value among the four columns is marked bold.
The computational results for MIQCPs seem to confirm our expectation that often a low fixing rate suffices to obtain a linear subproblem: 13 of the instances in our test set allow a cover of at most 5% of the variables, further 13 instances of at most 25%. Only 5 instances were in a medium range of 25%–50%, for another 6 a minimum cover contained more than 80% of the variables.
UC found a feasible solution for 24 test instances: on 11 out of the 13 instances with a cover of at most 5% of the variables, on 8 out of 13 instances with a cover of at most 25%, and on 5 out of the remaining 11 instances. In comparison, BARON found a feasible solution in 18 cases, Couenne in 6, SCIP in 25. UC found an optimal solution for instances `ex1266`, `sep1`, `st_e31`, and `tloss`, and a solution within less than 0.1% gap to the optimal solution value for instances `fac3`, and `util`.
There were 10 instances for which UC found a solution, but BARON did not, 4 times it was the other way around. Comparing UC to Couenne, this ratio is 20 : 2, w.r.t. SCIP it is 8 : 9. We note that on 7 instances UC found a solution, although none of BARON, Couenne, and SCIP did. For 11 instances UC found the single best solution and for 3 further instances it produced the same solution quality as the best of the other solvers.
The time for applying Undercover was always less than 0.3 seconds, except for the instance `waste`,

for which Undercover ran for 2.5 seconds. Figure 4 shows the average distribution of running time spent for solving the covering problem, processing the fix-and-propagate loop, solving the sub-MIP, polishing the solution with an NLP solver and for the remaining parts such as allocating and freeing data structures, constructing the auxiliary problems, computing conflict constraints, and so on. This average has been taken over all instances for which Undercover found a feasible solution, hence all main parts of the algorithm have been executed. The major amount of time, namely 60%, is spent in solving the sub-MIP. Solving the covering problem plus performing fix-and-propagate took only about 15% of the actual running time.
Although the polytope described by (4) is not integral, the covering problem could always be solved to optimality in the root node by SCIP's default heuristics and cutting plane algorithms. In 23 out of 37 cases, the minimal cover was nontrivial, with cover sizes of 8–50% of the nonlinear variables.
We note that in 11 out of the 13 cases for which the resulting sub-MIP was infeasible, the infeasibility was already detected during the fix-and-propagate stage, in the remaining two cases during root node processing of the sub-MIP. Thus in most cases, no time was wasted to try to find a solution for an infeasible subproblem, since the most expensive part, see Figure 4, can be skipped. This confirms that Undercover follows a "fast fail" strategy, a beneficial property of heuristic procedures applied within complete solvers, as argued in Section 1. Also, all feasible sub-MIPs could be solved to optimality within the imposed node limit of 500, which indicates that—with a state-of-the-art MIP solver at hand—the generated subproblems are indeed significantly easier than the full MIQCP.
For 14 out of 24 successful runs, all cover variables were integral. For the remaining 10 instances, NLP postprocessing was applied; 7 times, it could further improve the Undercover solution.
Recall that an arbitrary point $x^* \in [L, U]$ can serve as reference solution for Undercover. A natural alternative to the LP solution is a (locally) optimal solution of the NLP relaxation. An additional experiment showed that, using an NLP solution, Undercover only succeeded in finding a feasible solution for 18 instances of the MIQCP test set, instead of 24. If both versions found a solution, the quality of the one based on the NLP solution was better in eight cases, worse in three. Our interpretation for the lower success rate is that the advantage of the NLP solution, namely being feasible for all nonlinear constraints, is dominated by the fact that an NLP solution typically has a higher fractionality, which leads to a higher chance that infeasibility is introduced in line 17 of the Undercover algorithm in Figure 3.

Results for MINLP. As expected, Undercover is much less powerful for general MINLPs compared to MIQCPs. UC produced feasible solutions for only six out of more than a hundred test problems from MINLPLib. During root node processing, BARON found feasible solutions for 39 instances, COUENNE for 23, SCIP for 35. Table 2 shows results only for those instances on which Undercover succeeded. Although it is clearly outperformed by the other solvers w.r.t. the number of solutions found, we would like to mention that for each other solver there is at least one instance for which UC succeeded, but the solver did not.
Nevertheless, the experiments showed that fixing a small fraction of the variables would often have sufficed to obtain a linear subproblem: for 77 out of the 110 test instances, the minimum cover contained at most 25% of the variables, similar to the MIQCP case, but only 5 MINLPs allowed for a cover size below 5%. Hence, compared to the MIQCP test set, cover sizes are on average larger and very small covers occur rarely, but this alone does not explain the lower success rate. It simply appears to be more difficult to find feasible fixing values due to the higher complexity of the nonlinear constraints, even if we use the solution of an NLP relaxation as the reference point x^*. Curiously enough, Undercover produced feasible solutions for the two instances with smallest and the two instances with largest minimum cover.

Further experiments. We experimented with the following extensions of Undercover: re-ordering the fixing sequence if fix-and-propagate fails, see Footnote 2; re-solving the covering problem if the sub-MIP is infeasible, see Footnote 3; using a weighted version of the covering problem, see Section 7. None of those performed significantly better than our default strategy.
In a complete solver, primal heuristics are applied in concert, hence a feasible solution may be already at hand when starting Undercover. In our implementation, this is exploited in two ways. First, we use values from the incumbent solution as fixing alternatives during fix-and-propagate. Fixing the variables to values in the incumbent has the advantage that the resulting sub-MIP is guaranteed to be feasible, compare, e.g., [17]. Second, we add a primal cutoff to the sub-MIP to only look for improving solutions.[6] On four instances of the MIQCP testset, SCIP 2.1.1 with default heuristics including Undercover produced a primal solution that was significantly better than the best solution found by either SCIP or Undercover alone; a worse solution was produced only for one instance.

7 Variants

We experimented with a few more variants of the Undercover heuristic. Some of them proved beneficial for specific problem classes. For the standard setting presented in our computational results, however, they showed no significant impact. As they might prove useful for future applications of Undercover, we will give a brief description.
Our initial motivation for using a minimum cardinality cover was to minimize the impact on the original MINLP. Instead of measuring the impact of fixing variables uniformly, we could solve a weighted version of the covering problem (4). To better reflect the problem structure, the objective coefficients of the auxiliary variables α_i could be computed from characteristics of the original variables x_i such as the domain size, variable type, or appearance in nonlinear terms or constraints violated by the reference solution.
Instead of fixing the variables in a cover, we could also merely reduce their domains to a small neighborhood around the reference solution. Especially for continuous variables this leaves more freedom to the subproblem explored and can lead to better solutions found. Of course, the difficulty of solving the subproblem is increased. Nevertheless, small domains may allow for sufficiently tight underestimators for an MINLP solver to tackle the subproblem.
The main idea of Undercover is to reduce the computational effort by switching to a problem class that is easier to address. While we have focused on exploring a linear subproblem, for nonconvex MINLPs, convex subproblems may provide a larger neighborhood to be searched and still be sufficiently easy to solve.

8 Conclusion

In this paper, we have introduced Undercover, a primal MINLP heuristic exploring large linear subproblems induced by a minimum vertex cover. It differs from other recently proposed MINLP heuristics in that it is neither an extension of an existing MIP heuristic, nor solves an entire sequence of MIPs.
We defined the notion of a minimum cover of an MINLP and proved that it can be computed by solving a vertex covering problem on the co-occurrence graph induced by the sparsity patterns of the Hessians of the nonlinear constraint functions. Although $\mathcal{NP}$-hard, in our experiments

[6] A primal cutoff is an upper bound on the objective function that results in branch-and-bound nodes with worse dual bound not being explored.

covering problems could be solved rapidly. Several extensions and algorithmic details have been discussed.
Undercover exploits the fact that small covers correspond to large sub-MIPs. We showed that most instances of the MINLPLib [14] allow for covers consisting of at most 25% of their variables. In particular for MIQCPs, Undercover proved to be a fast start heuristic, that often produces feasible solutions of reasonable quality. The computational results indicate, that it complements nicely with existing root node heuristics in different solvers. Undercover is now one of the default heuristics applied in SCIP.

Acknowledgements

Many thanks to Tobias Achterberg, Christina Burt, and Stefan Vigerske for their valuable comments. This research has been supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin, `http://www.matheon.de`.

References

[1] Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization **4**(1), 4–20 (2007)

[2] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007)

[3] Achterberg, T.: SCIP: Solving Constraint Integer Programs. Mathematical Programming Computation **1**(1), 1–41 (2009)

[4] Achterberg, T., Berthold, T.: Improving the feasibility pump. Discrete Optimization **4**(1), 77–86 (2007)

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

[6] Berthold, T.: Primal heuristics for mixed integer programs. Diploma thesis, Technische Universität Berlin (2006)

[7] Berthold, T.: RENS – Relaxation Enforced Neighborhood Search. ZIB-Report 07-28, Zuse Institute Berlin (2007)

[8] Berthold, T., Heinz, S., Pfetsch, M.E., Vigerske, S.: Large neighborhood search beyond MIP. In: L.D. Gaspero, A. Schaerf, T. Stützle (eds.) Proceedings of the 9th Metaheuristics International Conference (MIC 2011), pp. 51–60 (2011)

[9] Berthold, T., Heinz, S., Vigerske, S.: Extending a CIP framework to solve MIQCPs. In: J. Lee, S. Leyffer (eds.) Mixed Integer Nonlinear Programming, *The IMA Volumes in Mathematics and its Applications*, vol. 154, pp. 427–444. Springer (2011)

[10] Bixby, R., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice – closing the gap. In: M. Powell, S. Scholtes (eds.) Systems Modelling and Optimization: Methods, Theory, and Applications, pp. 19–49. Kluwer Academic Publisher (2000)

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

[12] Bonami, P., Cornuéjols, G., Lodi, A., Margot, F.: A feasibility pump for mixed integer nonlinear programs. Mathematical Programming **119**(2), 331–352 (2009)

[13] Bonami, P., Gonçalves, J.: Heuristics for convex mixed integer nonlinear programs. Computational Optimization and Applications pp. 1–19 (2010)

[14] Bussieck, M., Drud, A., Meeraus, A.: MINLPLib – a collection of test models for mixed-integer nonlinear programming. INFORMS Journal on Computing **15**(1), 114–119 (2003)

[15] Bussieck, M.R., Vigerske, S.: MINLP solver software. In: J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh, J.C. Smith (eds.) Wiley Encyclopedia of Operations Research and Management Science. Wiley and Sons, Inc. (2010)

[16] D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: Experiments with a feasibility pump approach for nonconvex MINLPs. In: P. Festa (ed.) Experimental Algorithms, *Lecture Notes in Computer Science*, vol. 6049, pp. 350–360. Springer Berlin / Heidelberg (2010)

[17] Danna, E., Rothberg, E., Pape, C.L.: Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming **102**(1), 71–90 (2004)

[18] Dinur, I., Safra, S.: On the hardness of approximating vertex cover. Annals of Mathematics **162**, 439–485 (2005)

[19] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming **104**(1), 91–104 (2005)

[20] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**(1-3), 23–47 (2003)

[21] Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Mathematical Programming Computation **1**, 201–222 (2009)

[22] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)

[23] Ghosh, S.: DINS, a MIP improvement heuristic. In: Proceedings of the 12th IPCO, pp. 310–323 (2007)

[24] Griewank, A., Walther, A.: Evaluating derivatives: principles and techniques of algorithmic differentiation. Society for Industrial and Applied Mathematics (SIAM) (2008)

[25] Halperin, E.: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. SIAM Journal on Computation **31**, 1608–1623 (2002)

[26] Hansen, P., Jaumard, B.: Reduction of indefinite quadratic programs to bilinear programs. Journal of Global Optimization **2**(1), 41–60 (1992)

[27] Karakostas, G.: A better approximation ratio for the vertex cover problem. ACM Transactions on Algorithms **5**, 41:1–41:8 (2009)

[28] Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2-\epsilon$. Journal of Computer and System Sciences **74**(3), 335–349 (2008). Computational Complexity 2003

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

[30] Liberti, L., Mladenović, N., Nannicini, G.: A recipe for finding good solutions to MINLPs. Mathematical Programming Computation **3**, 349–390 (2011)

[31] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of DAC'01, pp. 530–535 (2001)

[32] Nannicini, G., Belotti, P.: Rounding-based heuristics for nonconvex MINLPs. Mathematical Programming Computation (2011)

[33] Nannicini, G., Belotti, P., Liberti, L.: A local branching heuristic for MINLPs. ArXiv e-prints (2008)

[34] Tawarmalani, M., Sahinidis, N.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**, 563–591 (2004)

[35] Vigerske, S.: Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming. Ph.D. thesis, Humboldt-Universität zu Berlin (2012). Submitted

[36] Wächter, A., Biegler, L.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1), 25–57 (2006)

[37] CppAD. A Package for Differentiation of C++ Algorithms. `http://www.coin-or.org/CppAD/`

[38] SCIP. Solving Constraint Integer Programs. `http://scip.zib.de/`

[39] IBM ILOG CPLEX Optimizer. `http://www.cplex.com/`

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

[41] SBB. ARKI Consulting & Development A/S and GAMS Inc. `http://www.gams.com/solvers/solvers.htm#SBB`

Table 1: Computational results on MIQCP instances.

instance	% cov	% nlcov	UC	BARON	COUENNE	SCIP
du-opt5	94.74	100.00	546.280975	–	157.082231	**15.62177**
du-opt	95.24	100.00	632.891424	108.331477	**3.905155**	4.904642
elf	5.56	50.00	1.675	1.675	–	**0.328**
ex1263	4.40	20.00	**30.1**	–	–	–
ex1264	4.88	20.00	**11.1**	–	–	–
ex1265	4.10	16.67	**15.1**	–	–	–
ex1266	3.57	14.29	**16.3**	–	–	–
fac3	80.60	100.00	**31995143.5**	72826487.6	–	32039523.2
feedtray2	4.67	50.00	–	–	–	**0**
meanvarx	23.33	100.00	14.824808	14.369232	14.404062	**14.369212**
netmod_dol1	0.30	100.00	0	-0.250023	–	**-0.372303**
netmod_dol2	0.38	100.00	**0**	0.0571641	–	**0**
netmod_kar1	0.88	100.00	0	**-0.364809**	–	0
netmod_kar2	0.88	100.00	0	**-0.364809**	–	0
nous1	29.79	36.84	–	–	**1.567072**	–
nous2	30.43	36.84	–	**0.625967**	**0.625967**	1.384316
nuclear14a	12.24	48.98	–	**-1.129079**	–	-1.100766
nuclear14b	12.24	48.98	–	–	–	**-1.097686**
nvs19	88.89	100.00	–	**-1098**	–	-1097.8
nvs23	90.00	100.00	484.2	**-1124.8**	–	-1122.2
nvs24	90.91	100.00	–	-1027.4	–	**-1028.8**
product2	23.80	100.00	–	–	–	**-2102.377**
product	36.22	100.00	–	–	–	**-2094.688**
sep1	10.53	40.00	**-510.081**	**-510.081**	**-510.081**	-470.13
space25a	5.84	41.86	–	–	–	–
space25	1.04	30.77	–	–	–	–
space960	27.74	43.43	–	–	–	**17130000**
spectra2	44.12	100.00	306.3343	119.8743	–	**13.97830**
st_e31	3.39	40.00	**-2**	**-2**	–	–
tln12	6.67	8.33	–	–	–	–
tln5	14.29	16.67	**15.1**	–	–	15.5
tln6	12.50	14.29	**32.3**	–	–	–
tln7	11.11	12.50	**30.3**	–	–	–
tloss	13.04	14.29	**16.3**	–	–	–
tltr	16.07	33.33	**61.133333**	–	–	83.475
util	3.12	16.67	999.690564	**999.57875**	–	1005.26814
waste	2.50	13.78	**661.337258**	684.087647	–	692.983766

Table 2: Computational results on selected MINLP instances.

instance	% cov	% nlcov	UC	BARON	COUENNE	SCIP
mbtd	0.18	0.18	**9.00006**	–	9.6667	–
nvs09	85.00	87.18	28.865663	–	**-43.134337**	-9.451041
nvs20	96.97	100.00	3276194408	**230.922165**	258.96067	–
stockcycle	9.98	100.00	357714.332	433304.376	–	**306163.247**
synthes1	33.33	100.00	**6.009759**	**6.009759**	7.092732	**6.009759**
johnall	0.14	0.16	**-224.73017**	-222.373032	**-224.73017**	**-224.73017**