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) revised version February 2013

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†

06/Jan/2013
(revised version)

Abstract

We present Undercover, a primal heuristic for nonconvex mixed-integer nonlinear programs (MINLPs) that explores a mixed-integer *linear* subproblem (sub-MIP) of a given MINLP. We solve a vertex covering problem to identify a smallest set of variables to fix, a so-called *cover*, such that each constraint is linearized. Subsequently, these variables are fixed to values obtained from a reference point, e.g., an optimal solution of a linear relaxation. Each feasible solution of the sub-MIP corresponds to a feasible solution of the original problem. We apply domain propagation to try to avoid infeasibilities, and conflict analysis to learn additional constraints from infeasibilities that are nonetheless encountered.

We present computational results on a test se t of mixed-integer quadratically constrained programs (MIQCPs) and MINLPs. It turns out that the majority of these instances allows for small covers. Although general in nature, we show that the heuristic is most successful on MIQCPs. It nicely complements existing root-node heuristics in different state-of-the-art solvers and helps to significantly improve the overall performance of the MINLP solver SCIP.

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 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, research in the last five years has shown an increasing interest in general-purpose primal heuristics [8, 9, 12, 13, 16, 31, 36, 37].

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., that 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 linear subproblem of a given MINLP. We evaluate its effectiveness on publicly available test sets and show that the heuristic performs well on MIQCPs, but has a low success rate on general MINLPs from MINLPLib [14].

During the design of Undercover, our focus was its application as a start heuristic inside a complete solver such as Baron [38], Bonmin [11], Couenne [5], GloMIQO [33, 34], Minotaur [45], or SCIP [3].

When primal heuristics are considered as standalone solving procedures, e.g., the RECIPE heuristic [31] 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 the above mentioned primal heuristics for MINLP. Firstly, unlike most of them [8, 12, 13, 16, 37], 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 and one for postprocessing the sub-MIP solution). On the contrary, most large neighborhood search heuristics [12, 16, 31, 36, 37] for MINLP solve a 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.

This 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. A summary of the complete algorithm is given in Section 5. Section 6 contains the results of our computational experiments. In Section 7, we briefly discuss further variants of the Undercover heuristic that we have experimented with, and Section 8 presents concluding remarks.

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 presence of integrality constraints. Thus, in a MIP context, "easy to solve" usually means that there are few integer variables.

Actually, integrality is a special case of (nonconvex) nonlinearity, since it is possible to model the integrality of a bounded integer variable $x_i \in \{L_i, \dots, U_i\}$ by the nonconvex polynomial constraint $(x_i - U_i)(x_i - U_i + 1) \cdots (x_i - L_i) = 0$. This insight matches the practical experience that in MINLP, while integralities do contribute to the complexity of the problem, the specific difficulty is the presence of nonlinearities. Hence, "easy" in an MINLP context can be understood as having 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). Certainly, the sub-MIP may be infeasible, but if not then each of its solutions is 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 suffices to obtain a sub-MIP. 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 definition.

Definition 2.1. *Let P be an MINLP of form (1) and $\mathcal{C} \subseteq \{1, \dots, n\}$ be a set of variable indices of P. We call $\mathcal{C}$ a* cover *of function g_k, $k \in \{1, \dots, m\}$, if and only if for all $x^* \in [L, U]$ the set*

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

is an affine set intersected with $[L, U]$. We call $\mathcal{C}$ a cover *of P if and only if $\mathcal{C}$ is a cover of all constraint functions $g_1, \dots, g_m$.*

Note that this definition refers to the functions g_k in the problem formulation and not to the feasible set of (1) directly. It does not exploit, for example, the fact that some of the functions might be redundant. Compare also, how a *convex* MINLP is typically understood as all g_k being convex rather than the feasible region being convex because only the first ensures the general validity of gradient cuts.

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:

$$\begin{aligned} \min \quad & -x_2 - x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3^2 - 4 \leqslant 0, \\ & x_1, x_2, x_3 \geqslant 0, \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned} \tag{3}$$

The only variable that appears in a nonlinear term is x_3, hence $\{3\}$ is a (smallest) cover of (3) w.r.t. the above definition. The (unique) optimal solution of its nonlinear relaxation is $(0, 3.75, 0.5)$ with an objective function value of -4.25.

Taking that relaxation, the idea of Undercover is to fix x_3 to 0.5, which renders (3) an integer linear program. The (unique) optimal solution of this is $(0, 3, 0.5)$ with an objective function value of -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 $(0, 4, 0)$.

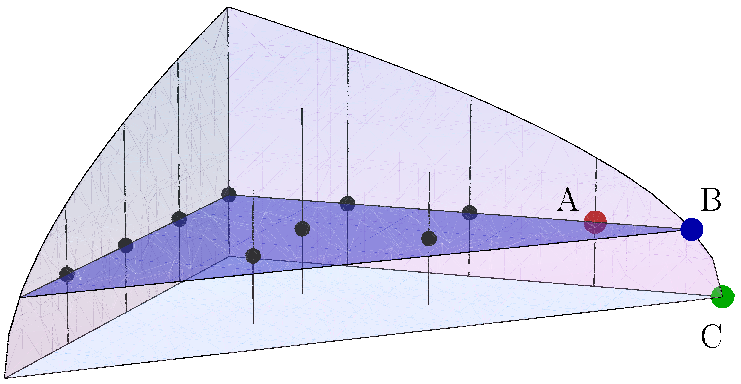

Figure 1: A convex MIQCP and the Undercover sub-MIP induced by the NLP relaxation.

This example is illustrated in Figure 1. The lightly shaded region shows the NLP relaxation; the parallel lines show the mixed-integer set of feasible solutions of (3). The darkly shaded area shows the polytope associated with 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, and 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 crucial step of the algorithm is found in line 5: finding a suitable cover of the given MINLP. Section 3 elaborates on this aspect of the algorithm 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 [30]. 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 that a sufficiently large percentage of variables 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, we may be forced to fix integer and continuous variables. The fixing of continuous variables, especially, can introduce a significant restriction, even rendering the subproblem infeasible. Thus, our heuristic aims 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 smallest 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. This allows for the following definition, which is a generalization of Hansen and Jaumard's notion of a *co-occurrence graph* for quadratically constrained quadratic programs [26].

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

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

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 are present. Trivially, the co-occurrence graph of a MIP is edge-free.*

The Undercover algorithm rests on the following observation.

Theorem 3.3. *Let P be an MINLP of form* (1) *with $g_1, \dots, g_m$ twice continuously differentiable on the interior of $[L, U]$. 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, $x^* \in \mathbb{R}^n$, $\mathcal{C} \subseteq \{1, \dots, n\}$, 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

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

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 $\min\{0 : x_i x_j \leqslant 0 \text{ for all } (i,j) \in E\}$. Hence, minimum vertex cover can be transformed to computing a minimum cover of an MINLP, or even a QCP. 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. This holds even when restricted to quadratic constraints.*

There are, however, many polynomial-time algorithms that approximate a minimum vertex cover within a factor of 2, such as 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 $10\sqrt{5} - 21 = 1.3606\ldots$ [18], hence no polynomial-time approximation scheme exists. Approximation ratios $2 - \epsilon(G)$ are known with $\epsilon(G) > 0$ depending on particular properties of the graph such as number of nodes [27] or bounded degree [25].

The following example shows that even for well-structured problems with obvious choices for small covers, computing a minimum cover may yield additional insight.

Example 3.5 (bilinear programming)**.** *A* bilinear program *is a QCP with a bipartition of its variables, $\{1,\dots,n\} = \mathcal{S} \cup \mathcal{T}$, $\mathcal{S} \cap \mathcal{T} = \emptyset$, and each quadratic term of the form $x_i x_j$, $i \in \mathcal{S}$, $j \in \mathcal{T}$. In this case, holding the variables of either $\mathcal{S}$ or $\mathcal{T}$ fixed, by definition one obtains a linear program—a simple property that has been used in solution approaches such as the cutting-plane algorithm of Konno [29]. The co-occurrence graph of a bilinear program is bipartite, and the sets $\mathcal{S}$ and $\mathcal{T}$ each constitutes a cover. See Figure 3 for an example. In global optimization, the variables in the smaller set are sometimes called complicating variables.*

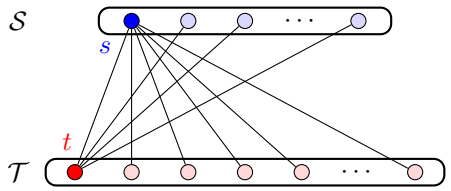

Figure 3: Example of a co-occurrence graph for a bilinear program: the cover $\mathcal{S}$ of complicating variables may be arbitrarily larger than the minimum cover $\{s, t\}$.

However, even in this well-structured case, a minimum cover is in general not given by the smaller of the partitions. The minimum of $|\mathcal{S}|$ and $|\mathcal{T}|$ cannot be bounded from above by a constant times the size of a minimum cover. As an example, consider the situation depicted in Figure 3. An arbitrary number of nodes connecting to t can be added to $\mathcal{S}$—and nodes connecting to s to $\mathcal{T}$—while the minimum cover remains the two variables s and t.

Note that here each of the two partitions is a minimal *cover in the sense that one cannot obtain a smaller cover by simply discarding an element. This shows that a simple greedy heuristic for generating small covers may fail.*

The above example motivates why, despite Corollary 3.4, 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 $(i, j) \in E_P$. For an MIQCP, for example, this requires all square terms and at least one variable in each bilinear term to be fixed. For a given general MINLP, to obtain as large a linear subproblem as possible, we solve the binary program

$$\min \left\{ \sum_{i=1}^{n} \alpha_i : \alpha_i + \alpha_j \geqslant 1 \text{ for all } (i, j) \in E_P, \alpha \in \{0, 1\}^n \right\} \tag{4}$$

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 exactly compute a minimum cover 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, despite the problem being $\mathcal{NP}$-hard in general, as shown above.

4 Domain propagation and conflict learning

Fixing a variable can have a great impact on the original problem and the approximation we use. An important detail, that is 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 structure of individual 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 [39], respectively.

To prevent obvious infeasibilities, we fix the variables in the cover one after the other and apply domain propagation after each fixing in order to further tighten the bounds, in particular those 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 fix the variable to the closest bound instead.[1] This fix-and-propagate procedure resembles

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

a method described by Fischetti and Salvagnin [21]. Additionally, we apply it to continuous variables.

In the above scheme, the fixed values of variables depend on the fixing order. Different variable orderings may lead to different propagations, thereby to different subproblems and different solutions being found.

It is also possible that a variable domain becomes empty. Then 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 the so-called conflict constraints prevents us from reaching the same deadlock again.

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 the 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 led to a substantial increase in the size of problems that modern SAT solvers can solve [35]. 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. Achterberg [1] has 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 have a longer processing time per node compared to SAT or CP solvers and they do not restart during search. As a consequence, the overhead for further exploring the conflict graph is often negligible compared to the potential savings. That is why MIP solvers with conflict learning such as SCIP [3] 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 original MINLP. This is done by building up the conflict graph that is implied by the variable fixings and the propagated bound changes. Therefore, 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

[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 is a simple version of a restarting mechanism. Restarting techniques are commonly used in solving SAT problems [35].

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 can 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, see Figure 4. 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* (AD) to automatically compute the sparsity pattern of the Hessian, (see, e.g., Griewank and Walther[24]).

As a consequence, the computational graphs of the constraint functions must be readily available. Oracle-based evaluations of the functions and their derivatives are not sufficient. Furthermore, the sparsity pattern computed by an AD code depends on the formulation of the function. Consider, for example, the linear expression $x^3 + 3x^2 - (x+1)^3$ for which an AD code might return unnecessary structural nonzeros in the Hessian. Therefore, the constraint functions should be reformulated and simplified in advance. In our implementation, this happens during the preprocessing phase of the SCIP solver. Although it does not explicitly take cover sizes into account, it helps to avoid simple cases of redundant variables in the cover.

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. Besides other primal heuristics, the MIP solver that we use applies a greedy algorithm as a start heuristic such that there is always an incumbent solution for covering problems available after root-node processing. 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. If fixing values accidentally coincide, each value is tested only once.

[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 in $\mathcal{C}$ are found, we can solve the covering problem again with an additional cutoff constraint $\sum_{i \in \mathcal{C}}(1 - \alpha_i) + \sum_{i \notin \mathcal{C}} \alpha_i \geqslant 1$ and try once more. 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 $\alpha^* \leftarrow \arg\min \left\{ \sum_{i=1}^n \alpha_i : \alpha_i + \alpha_j \geqslant 1 \text{ for all } (i, j) \in E, \alpha \in \{0, 1\}^n \right\}$

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 $X_i \leftarrow \min\{\max\{X_i, \hat{L}_i\}, \hat{U}_i\}$ `/* 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 $\min \{ d^\mathsf{T} x : g_k(x) \leqslant 0$ for $k = 1, \ldots, m$, $\hat{L}_i \leqslant x_i \leqslant \hat{U}_i$ for $i = 1, \ldots, n$, $x_i \in \mathbb{Z}$ for $i \in \mathcal{I} \}$

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 $\min \{ d^\mathsf{T} x : g_k(x) \leqslant 0$ for $k = 1, \ldots, m$, $L_i \leqslant x_i \leqslant U_i$ for $i = 1, \ldots, n$, $x_i = \hat{x}_i$ for $i \in \mathcal{I} \}$

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

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

end

Figure 4: The complete Undercover algorithm.

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 expect that it can often 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, as stated in line 33. Note that the two learning steps described in lines 27 and 33 are only relevant for the overall solution process of the complete solver within which Undercover is called. They do not alter the behavior of the Undercover algorithm itself.

For several design decisions, we considered alternatives which were ruled out in preliminary experiments. We tried using alternative objectives for the covering problem, an NLP solution instead of an LP solution for the fixing values, different variable orders in the fix-and-propagate loop, and so on. Details can be found in Section 7. These variants either altered the performance only slightly or they were inferior in the sense that they failed on a significant number of instances for which the default settings succeeded, but did not typically succeed on any instance for which the default failed. All of these choices have been made user parameters in our implementation of Undercover.

If a feasible sub-MIP solution $\hat{x}$ is 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 re-optimize 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 [38], Couenne [5], and LindoGlobal [32, 44]. Others, e.g., Bonmin [11] and SBB [46], 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 [39] to global optimality. For a comprehensive survey of available MINLP solver software, see Bussieck and Vigerske [15].

Experimental setup. The goal of our computational experiments was to analyze the performance of Undercover as a start heuristic for MINLPs, applied at the root node. To this end, we benchmarked against state-of-the-art solvers and measured its impact on the overall performance of an MINLP solver. We evaluated the sizes of the actual covers found, the success rate of Undercover, and the distribution of running time among different components of the algorithm.

We implemented the algorithm given in Figure 4 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

[4]The source code is publicly available within SCIP 2.1.1 and can be found at [47].

[5]With a stall node limit, we terminate if no improving solutions are found within a certain number of branch-and-bound nodes after the discovery of the current incumbent.

such instances. Components using sub-MIPs themselves are switched off altogether. For more details, we refer to the source code at [47].

We performed two main experiments to evaluate the Undercover algorithm. In order to investigate how Undercover can enhance the root-node performance of complete solvers, we compare UC with the root heuristics of four different MINLP solvers. SCIP, for instance, applies eleven primal heuristics at the root node: three rounding heuristics, three propagation heuristics, a trivial one, a feasibility pump, a local search, a repair heuristic and an improvement heuristic. Our second experiment measures the impact of using Undercover as a subroutine inside a complete solver on the overall performance.

In our first 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 [43] as LP solver, Ipopt 3.10 [40] as NLP solver for the postprocessing, and CppAD 20100101.4 [41] as expression interpreter for evaluating general nonlinear constraints. We refer to this Undercover standalone configuration as UC.

We tested against three state-of-the-art solvers for nonconvex MINLPs: Baron 9.3.1 [38] (commercial license), Couenne 0.3 [5] (open source), and SCIP 2.1.1 (academic license) with Undercover disabled. Additionally, we included Bonmin 1.6 [11] (open source), which is a solver for convex MINLP, but can be used as a heuristic for nonconvex problems. All solvers were run in their default configuration. In particular, the algorithm "B-BB" was used for Bonmin. We compare the primal bound obtained after the solution of the root node. Therefore, all solvers, including UC, were started with a node limit of one. We further imposed a large time limit of six hours to enforce termination and a memory limit of 8 GB.

Our test set is based on the 172 MIQCPs from the test suite of Misener and Floudas [34, 42], a broad selection of publically available MIQCP and QCP instances. From this test set we removed all instances for which we knew a-priori that Undercover would never be called. These were seven very easy instances, mainly of the `st_test` type, that are solved by SCIP presolving or the solution of the root LP, i.e., before Undercover would be executed, and eleven instances that have an unbounded root LP, all of the `nuclear` type. For the first experiment, we further excluded three of the `LeeCrudeOil` instances for which Bonmin did not terminate within nine hours (given a time limit of six hours). This left 147 instances.

We also tested Undercover on general MINLPs from MINLPLib [14], 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.

Our second experiment analyzes the impact of Undercover on the overall solution process of a complete solver. Therefore, we ran SCIP in its default settings, with and without Undercover, using a time limit of one hour and a memory limit of 40 GB. For this test, we included the three `LeeCrudeOil` instances, and excluded `ruiz_flowbased_pw4`, for which SCIP terminates with an error. This gives a test set of 149 instances.

The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSUSE 11.4 with a GCC 4.5.1 compiler. Hyperthreading and Turboboost were disabled. For the latter experiment, we ran only one job per node to avoid random noise in the measured running time that might be caused by cache-misses if multiple processes share common resources.

Results for MIQCP. The results for the experiments on MIQCPs are shown in Table 2. In columns "% cov" and "% nlcov", we report the relative size of the cover used by UC as a percentage of the total number of variables and of the number of variables that appear in at least one nonlinear term, respectively. A value of 100% in the "% nlcov" column means that the

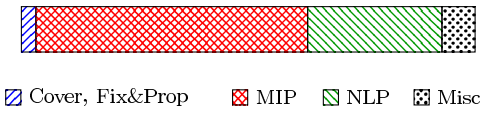

Figure 5: Distribution of running time among different components of Undercover heuristic.

trivial cover consisting of all variables appearing in nonlinear terms is the minimum cover. 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. All numbers are calculated w.r.t. the numbers of variables after preprocessing.

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 computational results for MIQCPs seem to confirm our expectation that a low fixing rate often suffices to obtain a linear subproblem: 25 of the instances in our test set allow a cover of at most 5% of the variables, a further 40 instances of at most 10% and 46 instances of at most 25%. Eighteen instances were in a medium range of 25%–50%; for another 18, a minimum cover contained more than half of the variables.

UC found a feasible solution for 76 test instances. Interestingly, it worked similarly well with small and large covers. For 15 out of 25 instances with a cover of at most 5% of the variables, UC found a solution, but also for 30 out of the 36 instances with a cover of at least 25%. In comparison, BARON found a feasible solution in 65 cases, COUENNE in 55, SCIP and BONMIN in 98 each.

There were 32 instances for which UC found a better solution than BARON, 20 for which it was better than SCIP, 36 for COUENNE, and 32 for BONMIN. We note that for six instances UC found the single best solution compared to *all* other solvers and for 27 further instances it produced the same solution quality as the best of the other solvers.

Out of 147 instances, the time for applying Undercover was less than 0.1 seconds in 131 cases, 14 times it was between 0.1 and 0.5 seconds; the two outliers are `waste` (1.31 seconds) and `Sarawak_Scenario81` (2.53 seconds). Figure 5 shows the average distribution of running time 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 were executed. The major amount of time, namely 65%, is spent in solving the sub-MIP. Solving the covering problem plus performing fix-and-propagate took only about 3% of the actual running time.

Although the polytope described by (4) is not integral, the covering problem could always be solved to optimality at the root node by SCIP's default heuristics and cutting plane algorithms. In 89 out of 147 cases, the minimum cover was nontrivial, with cover sizes of 8% to 60% of the nonlinear variables.

We note that in 55 out of the 70 cases for which the resulting sub-MIP was infeasible, the infeasibility was detected during the fix-and-propagate stage and in ten of the remaining fifteen cases during root-node processing of the sub-MIP.[6] Thus in most cases, no time was wasted trying to find a solution for an infeasible subproblem, since the most expensive part (see Figure 5) can be skipped. This confirms that Undercover follows a "fast fail" strategy, a beneficial

[6]For one instance, the feasibility status had not been decided within 500 nodes.

property of heuristic procedures applied within complete solvers, as argued in Section 1. Also, all except one feasible sub-MIP could be solved to optimality within the imposed node limit of 500. This indicates that—with a state-of-the-art MIP solver at hand—the generated subproblems are indeed significantly easier than the full MIQCP, as can also be seen when compared to the running times and the number of nodes in Table 3.

For 31 out of 76 successful runs, all cover variables were integral. For the remaining 35 instances, NLP postprocessing was applied; 21 times, it further improved the Undercover solution.

Recall that an arbitrary point $x^* \in [L, U]$ can serve as a 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 52 instances of the MIQCP test set, instead of 76. If both versions found a solution, the quality of the one based on the NLP solution was better in 23 cases, worse in eleven. 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 4. We also tried using an NLP relaxation for those eleven instances that were excluded because of an unbounded root LP: in no case was a feasible solution found.

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: `mbtd`, `nvs09`, `nvs20`, `stockcycle`, `synthes1`, and `johnall`. During root-node processing, Baron found feasible solutions for 39 instances, Couenne for 23, SCIP for 35, and Bonmin for 56. Although Undercover is clearly outperformed by the other solvers w.r.t. the number of solutions found, 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 five MINLPs allowed for a cover size below 5%. The extreme values were 0.18% for `mbtd` and 96.97% for `nvs20`.

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^*. Surprisingly, Undercover produced feasible solutions for the two instances with the smallest and the two instances with the largest minimum covers.

Undercover inside a complete solver. The previous experiments showed that for general MIQCP instances, Undercover nicely complements the existing root-node heuristics of Baron, Bonmin, Couenne and SCIP. The question remains as to whether this is beneficial for the overall solution process.

For this experiment, interactions of different primal heuristics with each other and with other solver components come into play. Obviously, a primal heuristic called prior to Undercover might already have found a solution which is better than the optimal solution of the Undercover sub-MIP, or in an extreme case, the solution process might have terminated before Undercover is called. Further, any solution found before Undercover is called might change the solution path. It might trigger variable fixings by dual reductions, which lead to a different LP and hence to a different initial situation for Undercover. Because of this, Undercover might succeed for

Table 1: Comparison of SCIP with and without Undercover (aggregated results).

	both solved (117)		time > 10 s (31)	
	nodes	time [s]	nodes	time [s]
SCIP − UC	702	10.8	21 097	98.2
SCIP + UC	610	9.4	15 619	74.5
shifted geom. mean	−15%	−15%	−35%	−32%

problems where it failed in the first experiment and vice versa. Note that even in the case of failure, Undercover might produce conflict clauses (see Section 4) and thereby be beneficial for the overall solution process.

In this experiment, our main criteria for measuring performance are the running time and the number of branch-and-bound nodes needed to prove optimality. To average values over all instances of the test set, we use a *shifted geometric mean*. The shifted geometric mean of values $t_1, \ldots, t_n$ with shift s is defined as $\sqrt[n]{\prod(t_i + s)} - s$. We use a shift of $s = 10$ for time and $s = 100$ for nodes in order to reduce the effect of very easy instances in the mean values. Further, using a *geometric* mean prevents hard instances at or close to the time limit from overly dominating the measures. Thus, the shifted geometric mean has the advantage that it reduces the influence of outliers in both directions.

The results of running SCIP once with and once without Undercover are shown in Table 3, and a summary can be found in Table 1. Both versions solved nearly the same set of instances within the given time limit; there was only one instance, `SLay10H`, which needed more than an hour when using SCIP with Undercover, but solved within 35 minutes and about 150 000 nodes otherwise. However, for those 117 instances which could be solved by both variants, the SCIP version that included Undercover needed 15% fewer nodes and 15% less running time in shifted geometric mean. If we ignore very easy instances that both versions solved in less than ten seconds, the improvement is even more significant: for the 31 instances which fall into this category, SCIP without Undercover is 32% slower and needs 35% more nodes in shifted geometric mean.

Even though the running times for Undercover are moderate (see above) for instances that solve within a fraction of a second, it sometimes consumes a significant amount of the running time. However, when we consider the 33 problems that need between one and ten seconds of running time, on average, Undercover only requires 1.5% of the overall running time; for the 58 instances that need more than ten seconds the ratio is 0.04%.

Further experiments. We experimented with the following extensions of Undercover: reordering 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., Danna et al. [17]. Second, we add a primal cutoff to the sub-MIP to only look for improving solutions.[7] On four instances of the MIQCP testset,

[7] 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.

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 provide a brief description here.

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 (integer or continuous), or appearance in nonlinear terms or in 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 for exploration of the subproblem and can lead to better solutions found. Of course, the difficulty of solving the subproblem is increased. Nevertheless, small domains may allow for a sufficiently tight relaxation 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 nonlinear 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 not an extension of an existing MIP heuristic, nor does it solve 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, covering problems were solved rapidly in our experiments. Several extensions and algorithmic details have been discussed.

Undercover exploits the fact that small covers correspond to large sub-MIPs. We showed that the majority of MIQCPs from the GloMIQO test set [42] and MINLPs from the MINLPLib [14] allow for covers consisting of at most 25% of their variables.

For MIQCPs, Undercover proved to be a fast start heuristic that often produces feasible solutions of reasonable quality. The computational results indicate that it nicely complements existing root-node heuristics in different solvers.

We further showed that for MIQCPs, applying Undercover at the root node significantly improved the overall performance of SCIP, in particular for hard instances. Undercover is now one of the default heuristics applied in SCIP.

Acknowledgements

Many thanks to Tobias Achterberg, J. Christopher Beck, Christina Burt, Angela Glover, Stefan Vigerske, the associate editor, and two anonymous reviewers 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). doi:10.1016/j.disopt.2006.10.006

[2] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007). http://vs24.kobv.de/opus4-zib/frontdoor/index/index/docId/1018

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

[4] Achterberg, T., Berthold, T.: Improving the feasibility pump. Discrete Optimization **4**(1), 77–86 (2007). doi:10.1016/j.disopt.2006.10.004

[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). doi:10.1080/10556780903087124

[6] Berthold, T.: Primal heuristics for mixed integer programs. Diploma thesis, Technische Universität Berlin (2006). http://vs24.kobv.de/opus4-zib/frontdoor/index/index/docId/1029

[7] Berthold, T.: RENS – the optimal rounding. ZIB-Report 12-17, Zuse Institute Berlin (2012). http://vs24.kobv.de/opus4-zib/frontdoor/index/index/docId/1520

[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). Available as Matheon Preprint #856. urn:nbn:de:0296-matheon-9752

[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). doi:10.1007/978-1-4614-1927-3_15

[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.T., Conn, A.R., Cornuéjols, G., Grossmann, I.E., Laird, C.D., 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). doi:10.1016/j.disopt.2006.10.011

[12] Bonami, P., Cornuéjols, G., Lodi, A., Margot, F.: A feasibility pump for mixed integer nonlinear programs. Mathematical Programming **119**(2), 331–352 (2009). doi:10.1007/s10107-008-0212-2

[13] Bonami, P., Gonçalves, J.: Heuristics for convex mixed integer nonlinear programs. Computational Optimization and Applications **51**, 729–747 (2012). doi:10.1007/s10589-010-9350-6

[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). doi:10.1287/ijoc.15.1.114.15159

[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. John Wiley & Sons, Inc. (2010). Online publication, doi:10.1002/9780470400531.eorms0527

[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). doi:10.1007/978-3-642-13193-6_30

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

[18] Dinur, I., Safra, S.: On the hardness of approximating vertex cover. Annals of Mathematics **162**, 439–485 (2005). doi:10.4007/annals.2005.162.439

[19] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming **104**(1), 91–104 (2005). doi:10.1007/s10107-004-0570-3

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

[21] Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Mathematical Programming Computation **1**(2–3), 201–222 (2009). doi:10.1007/s12532-009-0007-3

[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: M. Fischetti, D.P. Williamson (eds.) Integer Programming and Combinatorial Optimization, Proceedings of the 12th International IPCO Conference, *LNCS*, vol. 4513, pp. 310–323. Springer (2007). doi:10.1007/978-3-540-72792-7_24

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

[25] Halperin, E.: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. SIAM Journal on Computing **31**, 1608–1623 (2002). doi:10.1137/S0097539700381097

[26] Hansen, P., Jaumard, B.: Reduction of indefinite quadratic programs to bilinear programs. Journal of Global Optimization **2**(1), 41–60 (1992). doi:10.1007/BF00121301

[27] Karakostas, G.: A better approximation ratio for the vertex cover problem. ACM Transactions on Algorithms **5**, 41:1–41:8 (2009). doi:10.1145/1597036.1597045

[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). doi:10.1016/j.jcss.2007.06.019

[29] Konno, H.: A cutting plane algorithm for solving bilinear programs. Mathematical Programming **11**, 14–27 (1976). doi:10.1007/BF01580367

[30] Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960). http://www.jstor.org/stable/1910129

[31] Liberti, L., Mladenović, N., Nannicini, G.: A recipe for finding good solutions to MINLPs. Mathematical Programming Computation **3**, 349–390 (2011). doi:10.1007/s12532-011-0031-y

[32] Lin, Y., Schrage, L.: The global solver in the LINDO API. Optimization Methods and Software **24**(4–5), 657–668 (2009). doi:10.1080/10556780902753221

[33] Misener, R., Floudas, C.A.: Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. Mathematical Programming (2012). Online publication, doi:10.1007/s10107-012-0555-6

[34] Misener, R., Floudas, C.A.: GloMIQO: Global mixed-integer quadratic optimizer. Journal of Global Optimization (2012). Online publication, doi:10.1007/s10898-012-9874-7

[35] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th annual Design Automation Conference (DAC '01), pp. 530–535 (2001). doi:10.1145/378239.379017

[36] Nannicini, G., Belotti, P.: Rounding-based heuristics for nonconvex MINLPs. Mathematical Programming Computation **4**(1), 1–31 (2012). doi:10.1007/s12532-011-0032-x

[37] Nannicini, G., Belotti, P., Liberti, L.: A local branching heuristic for MINLPs. ArXiv e-print 0812.2188, Cornell University (2008). arXiv:0812.2188v1

[38] Tawarmalani, M., Sahinidis, N.V.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**, 563–591 (2004). doi:10.1007/s10107-003-0467-6

[39] 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

[40] Wächter, A., Biegler, L.T.: 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). doi:10.1007/s10107-004-0559-y

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

[42] GloMIQO 2.0. http://helios.princeton.edu/GloMIQO/

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

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

[45] MINOTAUR: A toolkit for MINLP. http://wiki.mcs.anl.gov/minotaur

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

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

Table 2: Cover sizes and primal solution values attained by UC and MINLP solvers on MIQCP test set.

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
CLay0203M	22.22	100.00	—	41573.262	—	54581.749	41986.253
CLay0204M	16.67	100.00	—	9199.9953	—	—	55601.563
CLay0205M	13.33	100.00	—	81612.088	—	—	8688.4286
CLay0303M	19.35	100.00	—	—	—	—	56141.526
CLay0304M	14.81	100.00	—	78552.626	—	—	—
CLay0305M	12.35	100.00	—	70332.768	—	—	11136.861
SLay04H	5.67	100.00	14395.62	9975.6616	13338.483	12013.906	12013.906
SLay04M	17.78	100.00	14395.62	12544.861	13241.081	9859.6597	12013.906
SLay05H	4.33	100.00	56836.232	24998.521	30419.804	—	30158.377
SLay05M	14.08	100.00	56836.232	27119.518	30286.966	—	35512.884
SLay06H	3.50	100.00	78418.037	135525.52	40761.755	—	44660.49
SLay06M	11.65	100.00	99393.821	42920.398	40225.601	—	46473.124
SLay07H	2.94	100.00	—	266528.05	105472.68	—	100329.71
SLay07M	9.93	100.00	157243.3	99366.754	105417.85	96289.867	112362.73
SLay08H	2.53	100.00	—	370075	131525.48	—	178522.98
SLay08M	8.65	100.00	458483.51	102746.46	143095.48	—	190574.08
SLay09H	2.22	100.00	—	152428.01	162397.71	—	188685.63
SLay09M	7.66	100.00	—	135783.77	184450.78	344702.34	205678.96
SLay10H	1.98	100.00	—	577942.5	216914.82	—	462524.34
SLay10M	6.87	100.00	—	144233.42	259159.88	—	352953.66
LeeCrudeOil1_05	8.25	25.00	—	—	—	—	-79.75
LeeCrudeOil1_06	7.97	25.00	—	—	—	—	-78.75
LeeCrudeOil1_07	7.79	25.00	—	—	—	—	—
LeeCrudeOil1_08	7.67	25.00	—	—	—	—	—
LeeCrudeOil1_09	7.58	25.00	—	—	—	—	-79.75
LeeCrudeOil1_10	7.52	25.00	—	—	—	—	—
LeeCrudeOil2_05	7.81	22.41	—	—	—	—	—
LeeCrudeOil2_07	7.29	22.47	—	—	—	—	—
LeeCrudeOil2_09	7.10	22.50	—	—	—	—	—
LeeCrudeOil2_10	7.04	22.51	—	—	—	—	—
LeeCrudeOil3_05	11.79	23.85	—	—	—	—	-81.314018
LeeCrudeOil3_06	11.66	23.37	—	—	—	—	—
LeeCrudeOil3_07	11.57	23.08	—	—	—	—	—
LeeCrudeOil3_08	11.51	22.88	—	—	—	—	-78.5
LeeCrudeOil3_09	11.45	22.75	—	—	—	—	—
LeeCrudeOil3_10	11.41	22.64	—	—	—	—	—
LeeCrudeOil4_06	6.57	21.62	—	-132.53069	—	—	—
LeeCrudeOil4_07	6.37	21.52	—	-132.16	—	—	—
LeeCrudeOil4_08	6.24	21.46	—	-132.47831	—	—	—
LeeCrudeOil4_09	6.15	21.40	—	-131.96016	—	—	—
LeeCrudeOil4_10	6.08	21.36	—	—	—	—	—
LiCrudeOil_ex01	9.69	46.67	—	—	—	—	5102.1808
LiCrudeOil_ex02	0.95	29.41	—	64752228	—	—	32923042
LiCrudeOil_ex03	5.72	33.33	—	—	—	—	—
LiCrudeOil_ex05	6.73	33.33	—	—	—	—	—

Table 2 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
LiCrudeOil_ex06	5.72	33.33	—	—	—	—	2979.375
LiCrudeOil_ex11	5.24	33.33	—	—	—	—	—
LiCrudeOil_ex21	4.70	33.33	—	—	—	—	—
alan	33.33	100.00	3.6	3	2.925	2.925	3.6
du-opt5	94.74	100.00	546.27998	15.621774	157.08224	—	1531.95
du-opt	95.24	100.00	632.89142	4.9046426	3.9051553	3725.6208	1758.1078
elf	5.56	50.00	1.675	0.32799999	—	1.675	1.9771427
ex1223a	33.33	100.00	6.5574613	4.5795817	4.5795824	4.5795824	4.5795824
ex1263a	17.39	20.00	30.1	29.6	—	—	—
ex1263	4.40	20.00	30.1	—	—	—	—
ex1264a	17.39	20.00	11.1	11.1	—	—	—
ex1264	4.88	20.00	11.1	—	—	—	—
ex1265a	14.71	16.67	15.1	15.1	—	—	—
ex1265	4.10	16.67	15.1	—	—	—	—
ex1266a	13.04	14.29	16.3	—	—	—	—
ex1266	3.57	14.29	16.3	—	—	—	—
ex4	13.51	100.00	-7.7891291	-8.0641362	289199.88	-7.7891291	-7.3132691
fac3	80.60	100.00	31995144	32039523	—	—	1.3065386e+08
feedtray2	4.67	50.00	—	0	—	0	1.5447167e-09
fuel	46.15	100.00	10286.143	11925	—	8566.119	8566.119
meanvarx	23.33	100.00	14.824808	14.369212	14.404062	14.369232	21.110398
netmod_dol1	0.30	100.00	0	-0.372303	—	-4.4408921e-15	4.9805856e-09
netmod_dol2	0.38	100.00	0	-1.012723e-09	—	0.057164114	4.6888223e-08
netmod_kar1	0.88	100.00	0	-1.0012411e-09	—	1.6653345e-15	2.6219098e-09
netmod_kar2	0.88	100.00	0	-1.0012411e-09	—	1.6653345e-15	2.6219098e-09
nous1	29.79	36.84	—	—	1.567072	1.567072	1.567072
nous2	30.43	36.84	—	1.3843163	0.62596741	0.62596741	0.62596741
nuclear14a	12.24	48.98	—	-1.1007655	—	—	-1.1286438
nuclear14b	12.24	48.98	—	-1.0976863	—	—	—
nuclear24a	12.24	48.98	—	-1.1007655	—	—	-1.1286438
nuclear24b	12.24	48.98	—	-1.0976863	—	—	—
nuclear25a	11.88	49.02	—	-1.0571308	—	—	-1.0890967
nuclear25b	11.88	49.02	—	—	—	—	—
nvs03	66.67	100.00	16	16	16	68	—
nvs10	66.67	100.00	-252	-310.8	—	-310.8	-310.8
nvs11	75.00	100.00	-270.8	-431	-431	-431	-431
nvs12	80.00	100.00	-358.4	-481.2	—	-481.2	-481.2
nvs13	83.33	100.00	-172	-580.4	—	-582.8	-585.2
nvs14	33.33	60.00	-39886.664	-40358.155	-40358.155	—	—
nvs15	40.00	100.00	1	1	1	1	3
nvs17	87.50	100.00	0	-1098.6	—	-1098.6	-1100.4
nvs18	85.71	100.00	-106.8	-777	-678.4	-776.6	-778.4
nvs19	88.89	100.00	0	-1097.8	—	-1098	—
nvs23	90.00	100.00	484.2	-1122.2	—	-1118	-1113.8
nvs24	90.91	100.00	—	-1028.8	—	-1025.8	-1031.8
prob02	16.67	16.67	792000	112235	112235	112235	112235
prob03	50.00	50.00	10	10	10	10	10
product2	23.80	100.00	—	-2102.3771	—	—	-2093.4479

Table 2 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
product	36.22	100.00	—	-2094.688	—	—	-1868.5446
eniplac_reformulated	19.51	100.00	—	—	—	-128473.32	-129658.81
fo7_2_reformulated	8.54	50.00	—	—	—	—	31.980873
fo7_reformulated	8.54	50.00	—	—	—	—	28.335082
fo8_reformulated	7.84	50.00	—	—	—	—	40.789936
fo9_reformulated	7.26	50.00	—	—	—	—	59.305615
m3_reformulated	13.64	50.00	37.8	46.306314	67.8	37.8	49.8
m6_reformulated	9.68	50.00	—	—	—	—	106.25688
m7_reformulated	8.75	50.00	—	—	—	—	130.75688
o7_2_reformulated	7.78	50.00	—	—	—	—	167.68058
o7_reformulated	7.78	50.00	—	—	—	—	189.66429
sep1	10.53	40.00	-510.08098	-470.13009	-510.08098	-510.08098	-510.08098
space25a	5.84	41.86	—	—	—	—	487.07433
space25	1.04	30.77	—	—	—	—	487.07433
space960	27.74	43.43	—	17130000	—	42155000	—
spectra2	44.12	100.00	306.3343	13.978303	—	28.143456	306.3343
st_e13	50.00	100.00	2	2	2	2	2
st_e27	40.00	100.00	9	2	2	2	2
st_e31	3.39	40.00	-2.0000015	—	—	—	-2
st_miqp2	40.00	100.00	2	2	—	2	—
st_miqp3	50.00	100.00	-6	-6	-6	-6	-6
st_miqp4	50.00	100.00	-4574	-4574	-4574	-4574	-4574
st_miqp5	25.00	100.00	-333.88891	-333.88891	-333.88889	-333.88889	-333.88889
st_test4	28.57	100.00	-7	-7	-7	-7	—
st_test8	96.00	100.00	-26041	-29605	-29575	-29605	-29605
st_testgr1	90.91	100.00	-6.688	-12.79955	-12.7842	-12.7392	-7.713
st_testgr3	95.24	100.00	-20.27475	-20.5795	-20.49105	-20.47635	-20.0796
st_testph4	75.00	100.00	-56	-80.5	-80.5	-80.5	-80.5
tln12	6.67	8.33	—	—	—	—	—
tln2	33.33	33.33	17.3	5.3	5.3	—	—
tln4	16.67	20.00	11.1	12.4	—	—	—
tln5	14.29	16.67	15.1	15.5	—	—	11
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	999.69056	1005.2681	—	1000.0498	999.57875
waste	2.50	13.78	626.89124	692.98377	—	710.352	1011.5257
Sarawak_Scenario16	4.97	19.05	-31479.405	-31868.099	-31921.57	-31409.405	—
Sarawak_Scenario1	4.04	19.05	-32435.405	-31115.543	-27840.759	-32399.405	-22605.074
Sarawak_Scenario81	5.03	19.05	-31479.405	-31865.355	-30515.443	-31409.405	—
lee1	16.33	40.00	—	—	—	-4640.0824	-4296.9511
lee2	22.64	50.00	—	—	—	—	—
meyer04	10.17	42.86	—	—	—	—	1422175.2
meyer10	7.61	23.08	—	—	—	3698168.3	—
meyer15	6.13	16.67	—	—	—	—	1008046.4
ahmetovic1_pw4	2.40	35.00	—	—	—	—	606466.42
ahmetovic2_pw4	2.09	28.57	—	—	—	—	1217509.3

Table 2 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
karuppiah1	27.59	34.78	—	139.32508	—	117.05263	117.45263
karuppiah2_pw4	14.74	26.42	—	381396.6	490999.74	480435.29	—
karuppiah3_pw4	17.72	28.57	—	1753698.3	1753698.3	—	—
karuppiah4_pw4	18.57	27.96	—	1430067.5	2376436.2	—	1046406.2
ruiz_concbased_pw4	13.33	36.36	—	414748.31	—	—	—
ruiz_flowbased_pw4	8.33	45.45	—	—	—	346345.39	—

Table 3: Comparison of overall performance of SCIP 2.1.1 with and without Undercover on MIQCP test set. Columns "nodes" and "time" show the number of branch-and-bound nodes and the running time needed to solve an instance to proven optimality, respectively. Column "pb root" depicts the primal bound after the root node.

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
CLay0203M	48	0.1	41572.98	48	0.2	41572.98
CLay0204M	661	0.7	9199.995	721	0.7	9199.995
CLay0205M	10 690	4.2	81611.33	9 655	3.9	81611.33
CLay0303M	87	0.1	–	87	0.1	–
CLay0304M	316	0.6	78552.09	298	0.6	78552.09
CLay0305M	9 205	3.9	70332.47	8 969	4.1	70332.47
SLay04H	31	0.6	14395.62	31	0.3	9975.662
SLay04M	71	0.6	11676.06	132	0.8	12544.86
SLay05H	288	2.5	24998.52	286	2.3	24998.52
SLay05M	24	0.6	25589.95	56	0.7	27119.52
SLay06H	992	5.1	135525.5	1 670	8.6	135525.5
SLay06M	266	1.4	41921.2	618	2.0	42920.4
SLay07H	5 406	66.9	266528.1	5 895	71.9	266528.1
SLay07M	730	3.8	71077.43	1 430	9.6	99366.75
SLay08H	4 769	61.2	370075	32 232	310.3	370075
SLay08M	1 079	5.8	102746.5	1 493	7.0	102746.5
SLay09H	6 971	107.8	152428	31 680	383.1	152428
SLay09M	3 561	24.8	136774.1	1 453	22.4	135783.8
SLay10H	>212 055	>3600.0	577942.5	144 350	2144.1	577942.5
SLay10M	27 922	181.9	144233.4	170 975	1034.2	144233.4
LeeCrudeOil1_05	25	1.0	–	13	0.8	–
LeeCrudeOil1_06	14	1.3	–	27	1.5	–
LeeCrudeOil1_07	29	1.5	–	29	1.4	–
LeeCrudeOil1_08	40	4.3	–	39	4.3	–
LeeCrudeOil1_09	62	3.9	–	108	4.4	–
LeeCrudeOil1_10	141	7.4	–	179	8.6	–
LeeCrudeOil2_05	32	2.1	–	80	2.1	–
LeeCrudeOil2_06	21	3.5	-101.1746	46	3.3	-101.1746
LeeCrudeOil2_07	397	6.5	–	384	6.2	–
LeeCrudeOil2_08	261	7.3	-101.1738	371	8.0	-101.1738
LeeCrudeOil2_09	713	17.5	–	517	16.5	–
LeeCrudeOil2_10	672	20.5	–	682	30.0	–
LeeCrudeOil3_05	2 141	7.3	–	6 821	18.3	–
LeeCrudeOil3_06	14 851	53.1	–	21 515	65.0	–
LeeCrudeOil3_07	20 341	77.2	–	28 851	95.2	–
LeeCrudeOil3_08	52 781	259.0	–	32 411	160.6	–
LeeCrudeOil3_09	48 118	289.9	–	51 121	270.9	–
LeeCrudeOil3_10	41 941	308.5	–	37 141	264.0	–
LeeCrudeOil4_05	106	2.5	–	23	3.3	–
LeeCrudeOil4_06	20	3.5	-132.5307	16	4.5	-132.5307
LeeCrudeOil4_07	118	5.7	-132.16	21	5.8	-132.16
LeeCrudeOil4_08	67	8.8	-132.4783	212	15.0	-132.4783
LeeCrudeOil4_09	43	15.5	-131.9602	28	11.9	-131.9602
LeeCrudeOil4_10	419	20.8	–	157	21.9	–
LiCrudeOil_ex01	>1 318 676	>3600.0	–	>1 178 319	>3600.0	–
LiCrudeOil_ex02	>1 096 681	>3600.0	64752230	>1 074 998	>3600.0	64752230
LiCrudeOil_ex03	>285 396	>3600.0	–	>307 358	>3600.0	–
LiCrudeOil_ex05	>375 180	>3600.0	–	>408 032	>3600.0	–
LiCrudeOil_ex06	19 790	313.8	–	60 296	805.1	–
LiCrudeOil_ex11	>269 067	>3600.0	–	>271 844	>3600.0	–
LiCrudeOil_ex21	>232 969	>3600.0	–	>242 431	>3600.0	–
alan	6	0.1	2.924996	6	0.1	2.924996
du-opt5	80	0.5	13.60875	58	0.4	15.62177
du-opt	238	0.7	7.246512	162	0.6	4.904643
elf	293	0.3	0.328	293	0.4	0.328
ex1223a	1	0.1	4.579582	1	0.0	4.579582
ex1263a	229	0.2	29.3	126	0.2	29.6
ex1263	596	0.7	30.1	194	0.4	–
ex1264a	176	0.2	10.3	128	0.1	11.1
ex1264	86	0.2	11.1	179	0.2	–
ex1265a	72	0.1	14.3	70	0.1	15.1
ex1265	69	0.3	11.3	186	0.4	–

Table 3 continued

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
ex1266a	1	0.0	16.3	397	0.5	–
ex1266	1	0.1	16.3	209	0.6	–
ex4	11	0.7	-8.064135	11	0.8	-8.064136
fac3	8	0.2	31995140	12	0.1	32039520
feedtray2	1	0.1	0	1	0.1	0
fuel	3	0.1	10286.14	5	0.1	11925
gbd	1	0.0	2.2	1	0.0	2.2
meanvarx	4	0.1	14.36921	4	0.1	14.36921
netmod_dol1	>42 355	>3600.0	-0.3740157	>40 552	>3600.0	-0.372303
netmod_dol2	793	71.7	0	80	33.4	0
netmod_kar1	315	4.5	-0.3717949	279	3.7	0
netmod_kar2	315	4.5	-0.3717949	279	3.7	0
nous1	>2 196 718	>3600.0	–	>2 174 442	>3600.0	–
nous2	3 311	2.9	1.384316	4 764	3.4	1.384316
nuclear104	>62 874	>3600.0	–	>66 256	>3600.0	–
nuclear10a	>49	>3600.0	–	>43	>3600.0	–
nuclear10b	>1	>3600.0	–	>1	>3600.0	–
nuclear14a	>62 466	>3600.0	-1.111458	>57 760	>3600.0	-1.100766
nuclear14b	>47 549	>3600.0	-1.097686	>47 568	>3600.0	-1.097686
nuclear14	>1 473 004	>3600.0	–	>1 471 465	>3600.0	–
nuclear24a	>62 343	>3600.0	-1.111458	>57 760	>3600.0	-1.100766
nuclear24b	>47 556	>3600.0	-1.097686	>47 482	>3600.0	-1.097686
nuclear24	>1 474 515	>3600.0	–	>1 463 971	>3600.0	–
nuclear25a	>55 186	>3600.0	-1.057131	>49 835	>3600.0	-1.057131
nuclear25b	>33 525	>3600.0	–	>35 924	>3600.0	–
nuclear25	>1 380 060	>3600.0	–	>1 374 983	>3600.0	–
nuclear49a	>5 883	>3600.0	–	>6 729	>3600.0	–
nuclear49b	>2 920	>3600.0	–	>3 032	>3600.0	–
nuclear49	>379 308	>3600.0	–	>378 649	>3600.0	–
nuclearva	>2 988 726	>3600.0	–	>2 978 098	>3600.0	–
nuclearvb	>3 004 258	>3600.0	–	>3 005 785	>3600.0	–
nuclearvc	>2 983 650	>3600.0	–	>3 002 087	>3600.0	–
nuclearvd	>2 719 871	>3600.0	–	>2 715 102	>3600.0	–
nuclearve	>2 735 734	>3600.0	–	>2 732 253	>3600.0	–
nuclearvf	>2 740 055	>3600.0	–	>2 743 820	>3600.0	–
nvs03	1	0.0	16	1	0.0	16
nvs10	1	0.0	-310.8	1	0.0	-310.8
nvs11	3	0.0	-431	3	0.0	-431
nvs12	6	0.1	-481.2	5	0.0	-481.2
nvs13	12	0.1	-580.4	9	0.1	-580.4
nvs14	1	0.0	-40358.15	1	0.0	-40358.15
nvs15	4	0.1	1	5	0.0	1
nvs17	51	0.1	-1098.6	45	0.1	-1098.6
nvs18	23	0.1	-777	20	0.1	-777
nvs19	89	0.2	-1097.8	84	0.2	-1097.8
nvs23	106	0.3	-1124.2	103	0.3	-1122.2
nvs24	104	0.3	-1028.8	103	0.3	-1028.8
prob02	1	0.0	112235	1	0.0	112235
prob03	1	0.0	10	1	0.0	10
product2	>3 258 756	>3600.0	-2099.124	>3 311 596	>3600.0	-2102.377
product	7 317	21.1	-2094.688	7 989	23.3	-2094.688
iplac_reformulated	282	0.7	–	282	0.8	–
fo7_2_reformulated	57 203	33.5	–	62 818	35.6	–
fo7_reformulated	185 976	108.2	–	210 033	125.2	–
fo8_reformulated	364 808	230.3	–	426 135	250.6	–
fo9_reformulated	1 689 569	1130.6	–	2 775 675	1831.0	–
m3_reformulated	14	0.2	37.8	21	0.1	46.30631
m6_reformulated	8 958	4.1	–	1 601	1.5	–
m7_reformulated	4 635	3.6	–	5 988	4.2	–
o7_2_reformulated	1 461 823	773.7	–	1 501 419	790.7	–
o7_reformulated	3 647 967	2124.1	–	3 838 657	2191.2	–
sep1	37	0.3	-510.081	47	0.2	-470.1301
space25a	>339 329	>3600.0	–	>188 127	>3600.0	–
space25	>6 611	>3600.0	–	>70 227	>3600.0	–
space960	>3 760	>3600.0	17130000	>3 479	>3600.0	17130000
spectra2	19	0.8	13.9783	23	0.6	13.9783
st_e13	1	0.0	0	1	0.0	0
st_e27	1	0.0	2	1	0.0	2

Table 3 continued

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
st_e31	1 647	1.0	-2.000001	2 038	1.0	–
st_miqp1	1	0.0	281	1	0.0	281
st_miqp2	1	0.0	2	1	0.0	2
st_miqp3	1	0.0	-6	1	0.0	-6
st_miqp4	1	0.1	-4574	1	0.0	-4574
st_miqp5	1	0.1	-333.8889	1	0.0	-333.8889
st_test1	1	0.0	0	1	0.0	0
st_test2	1	0.0	-9.25	1	0.0	-9.25
st_test3	1	0.0	-7	1	0.0	-7
st_test4	1	0.0	-7	1	0.0	-7
st_test5	1	0.0	-110	1	0.0	-110
st_test6	1	0.0	471	1	0.0	471
st_test8	1	0.0	-29605	1	0.0	-29605
st_testgr1	48	0.1	-12.79955	20	0.1	-12.79955
st_testgr3	28	0.1	-20.5795	23	0.1	-20.5795
st_testph4	1	0.0	-80.5	1	0.0	-80.5
tln12	>1 549 104	>3600.0	–	>1 481 337	>3600.0	–
tln2	1	0.0	5.3	1	0.0	5.3
tln4	2 658	1.5	11.1	2 784	1.5	12.4
tln5	171 037	105.1	15.1	104 002	63.1	15.5
tln6	>5 322 038	>3600.0	32.3	>5 432 291	>3600.0	–
tln7	>3 010 846	>3600.0	30.3	>3 179 741	>3600.0	–
tloss	1	0.0	16.3	145	0.2	–
tltr	38	0.2	61.13333	94	0.2	83.475
util	7	0.3	999.6906	213	0.4	1005.268
waste	>2 080 248	>3600.0	621.8648	>2 068 008	>3600.0	692.9838
Sarawak_Scenario16	>706 322	>3600.0	-31868.1	>668 385	>3600.0	-31868.1
Sarawak_Scenario1	502	1.1	-32435.4	541	1.4	-31115.54
Sarawak_Scenario81	>152 074	>3600.0	-31865.36	>153 286	>3600.0	-31865.36
lee1	2 828	2.2	–	21 451	19.3	–
lee2	37 584	44.8	–	31 580	36.9	–
meyer04	>3 106 891	>3600.0	–	>3 047 664	>3600.0	–
meyer10	>1 391 651	>3600.0	–	>1 338 247	>3600.0	–
meyer15	>183 963	>3600.0	–	>358 139	>3600.0	–
ahmetovic1_pw4	42 842	36.1	–	63 195	58.2	–
ahmetovic2_pw4	>684 192	>3600.0	–	>640 108	>3600.0	–
karuppiah1	1 941	1.8	139.3251	1 031	1.1	139.3251
karuppiah2_pw4	>4 650 797	>3600.0	381396.6	>4 299 998	>3600.0	381396.6
karuppiah3_pw4	34 191	23.0	1753698	61 191	33.9	1753698
karuppiah4_pw4	>1 198 193	>3600.0	1430067	>1 110 591	>3600.0	1430067
ruiz_concbased_pw4	8 511	8.3	414748.3	27 271	22.7	414748.3