Available online at www.sciencedirect.com
ScienceDirect
Discrete Optimization 4 (2007) 4–20
DISCRETE OPTIMIZATION
www.elsevier.com/locate/disopt

Conflict analysis in mixed integer programming

Tobias Achterberg*

Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany

Received 15 September 2005; received in revised form 24 August 2006; accepted 6 October 2006
Available online 21 December 2006

Abstract

Conflict analysis for infeasible subproblems is one of the key ingredients in modern SAT solvers. In contrast, it is common practice for today's mixed integer programming solvers to discard infeasible subproblems and the information they reveal. In this paper, we try to remedy this situation by generalizing SAT infeasibility analysis to mixed integer programming.

We present heuristics for branch-and-cut solvers to generate valid inequalities from the current infeasible subproblem and the associated branching information. SAT techniques can then be used to strengthen the resulting constraints. Extensive computational experiments show the potential of our method. Conflict analysis greatly improves the performance on particular models, while it does not interfere with the solving process on the other instances. In total, the number of required branching nodes on general MIP instances was reduced by 18% in the geometric mean, and the solving time was reduced by 11%. On infeasible MIPs arising in the context of chip verification and on a model for a particular combinatorial game, the number of nodes was reduced by 80%, thereby reducing the solving time by 50%.

Keywords: Mixed integer programming; Branch-and-cut; Conflict analysis; SAT; Infeasible mixed integer programming

1. Introduction

A well-known approach to solve mixed integer programs (MIPs) is to apply branch-and-bound type algorithms: the given problem instance is divided into smaller subproblems, and this decomposition is continued recursively until an optimal solution of the respective subproblem can be identified or the subproblem is detected to be infeasible or to exceed the primal bound. It seems that current state-of-the-art MIP solvers like CPLEX [17], LINGO [19], SIP [21], or XPRESS [12], simply discard infeasible and bound-exceeding subproblems without paying further attention to them.

For solving the satisfiability problem (SAT), a similar branching scheme to decompose the problem into smaller subproblems is usually applied, which was proposed by Davis, Putnam, Logemann, and Loveland [13,14]. Modern SAT solvers, however, try to learn from infeasible subproblems, an idea due to Marques-Silva and Sakallah [20]. The infeasibilities are analyzed in order to generate so-called *conflict clauses*. These are implied clauses that help to prune the search tree.

The idea of conflict analysis is to identify a reason for the infeasibility of the current subproblem and exploit this knowledge. In SAT solving, such a reason is a subset of the current variable fixings that already suffices to trigger

* Corresponding address: Zuse Institute Berlin, Optimization, Takustr. 7, 14195 Berlin, Germany.
E-mail address: achterberg@zib.de.

1572-5286/$ - see front matter
doi:10.1016/j.disopt.2006.10.006

a chain of logical deductions that ends in a contradiction. This means, the fixing of the variables of this *conflict set* renders the problem instance infeasible. The *conflict clause*, which can be learned from this conflict, states that at least one of the variables in the conflict set has to take the opposite truth value. This clause is added to the clause database and can then be used at other subproblems to find additional implications, thereby pruning the search tree.

A similar idea of conflict analysis are the so-called *no-goods* which emerged from the constraint programming community, see, e.g. Stallman and Sussman [28], Ginsberg [15], and Jiang, Richards and Richards [18]. They can be seen as a generalization of conflict clauses to the domain of constraint programming.

In this paper, we propose a generalization of conflict analysis to branch-and-bound-based mixed integer programming. The same generalization was independently developed by Sandholm and Shields [26]. We consider a mixed integer program of the form:

$$\text{(MIP)} \quad \max\{c^T x \mid Ax \leq b, l \leq x \leq u, x_j \in \mathbb{Z} \text{ for all } j \in I\}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c, l, u \in \mathbb{R}^n$, and $I \subseteq N = \{1, \ldots, n\}$. Suppose a subproblem in the branch-and-bound search tree is detected to be infeasible or to exceed the primal bound. We will show that this situation can be analyzed with similar techniques as in SAT solving: a *conflict graph* is constructed from which *conflict constraints* can be extracted. These constraints can be used as cutting planes to strengthen the relaxations of other subproblems in the tree.

Note that the term "*conflict graph*" is used differently in the SAT and MIP communities. In MIP solving, the conflict graph consists of implications between two binary variables each, see e.g. Atamtürk, Nemhauser, and Savelsbergh [5]. It represents a vertex-packing relaxation of the MIP instance and can, for instance, be used to derive cutting planes like clique cuts. In SAT solving, however, the conflict graph arises from the *implication graph* which is a hyper-graph containing all implications between *any* number of variables. For each infeasible subproblem, a specific conflict graph is constructed to represent the implications from which the infeasibility follows. In this paper we adopt the nomenclature of the SAT community.

There are two main differences of MIP and SAT solving in the context of conflict analysis. First, the variables of an MIP need not to be of binary type; we also have to deal with general integer and continuous variables. Furthermore, the infeasibility of a subproblem in the MIP search tree usually has its sources in the linear programming (LP) relaxation of the subproblem. In this case, we first have to find a (preferably simple) reason for the LP's infeasibility before we can apply the SAT conflict analysis techniques for generating conflict constraints.

The rest of the paper is organized as follows. Section 2 reviews conflict graph analysis of SAT solvers. For an infeasible subproblem, it is shown how to generate the conflict graph and how to extract valid conflict clauses out of this graph. Section 3 deals with the generalization of these techniques to mixed integer programming. We explain how infeasible and bound-exceeding linear programs can be analyzed in order to detect a conflict in the local bounds of the variables. This conflict is used as starting point to construct the conflict graph from which conflict constraints can be extracted with the techniques explained in Section 2. Additionally, we discuss how we generalize the notion of the conflict graph in the presence of nonbinary variables. Experimental results in Section 4 demonstrate that conflict analysis can lead to moderate savings in the number of branching nodes and the time needed to solve an MIP. For the examined infeasible MIPs, the savings due to conflict analysis are substantial.

2. Conflict analysis in SAT solving

In this section we review the conflict analysis techniques used in SAT solving, see e.g. Marques-Silva and Sakallah [20] or Zhang et al. [32]. The Boolean satisfiability problem (SAT) is defined as follows. The Boolean truth values *false* and *true* are identified with the values 0 and 1, and Boolean formulas are evaluated correspondingly.

Definition 2.1 (*SAT*). Let $C = C_1 \wedge \cdots \wedge C_m$ be a logic formula in conjunctive normal form (CNF) on Boolean variables $x_1, \ldots, x_n$. Each clause $C_i = \ell_1^i \vee \cdots \vee \ell_{k_i}^i$ is a disjunction of literals. A literal $\ell \in L = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ is either a variable x_j or the negation of a variable $\bar{x}_j$. The task is to either find an assignment $x^\star \in \{0, 1\}^n$, such that the formula C is satisfied, i.e. each clause C_i evaluates to 1, or to conclude that C is unsatisfiable, i.e. for all $x \in \{0, 1\}^n$ at least one C_i evaluates to 0.

SAT was the first problem shown to be $\mathcal{NP}$-complete by Cook [10]. Besides its theoretical relevance, it has many practical applications, e.g. in the design and verification of integrated circuits or in the design of logic-based intelligent

systems. We refer to Biere and Kunz [7] for an overview of SAT techniques in chip verification and to Truemper [29] for details on logic-based intelligent systems.

Modern SAT solvers like CHAFF [23] or BERKMIN [16] rely on the following techniques:

- using a branching scheme (the *DPLL-algorithm* of Davis, Putnam, Logemann, and Loveland [13,14]) to split the problem into smaller subproblems,
- applying *Boolean Constraint Propagation* (BCP) [31] on the subproblems, which is a simple node preprocessing, and
- analyzing infeasible subproblems to produce conflict clauses [20], which help to prune the search tree later on.

The DPLL-algorithm creates two subproblems at each node of the search tree by fixing a single variable to zero and one, respectively. The nodes are processed in a depth first fashion. At each node, BCP is applied to derive further deductions by substituting the fixed variables in the clauses. It may happen that a still unsatisfied clause is thereby reduced to a single literal, a so-called *unit clause*. In this case, the remaining literal can be fixed to 1. BCP is applied iteratively until no more deductions can be found or a clause becomes empty, i.e. all its literals are fixed to 0. The latter case is called a conflict, and conflict analysis can be performed to produce a *conflict clause*, which is explained in the following.

The deductions performed in BCP can be visualized in a *conflict graph* $G = (V, A)$. The vertices $V = \{\ell_1, \ldots, \ell_k, \lambda\} \subset L \cup \{\lambda\}$ of this directed graph represent the current value assignments of the variables, with the special vertex λ representing the conflict. The arcs can be partitioned into $A = A_1 \cup \cdots \cup A_D \cup A_\lambda$. Each subset A_d, $d = 1, \ldots, D$, represents one deduction: whenever a clause $C_i = \ell^i_1 \vee \cdots \vee \ell^i_{k_i} \vee \ell^i_r$ became a unit clause in BCP with remaining unfixed literal ℓ^i_r, a set of arcs $A_d = \{(\bar{\ell}^i_1, \ell^i_r), \ldots, (\bar{\ell}^i_{k_i}, \ell^i_r)\}$ is created in order to represent the deduction $\bar{\ell}^i_1 \wedge \cdots \wedge \bar{\ell}^i_{k_i} \to \ell^i_r$ that is implied by C_i. The additional set of arcs $A_\lambda = \{(\bar{\ell}^\lambda_1, \lambda), \ldots, (\bar{\ell}^\lambda_{k_\lambda}, \lambda)\}$ represents clause C_λ that detected the conflict (i.e. the clause that became empty in BCP).

Example 2.2. Consider the CNF $C = C_1 \wedge \cdots \wedge C_{18}$ with the following clauses:

$$
\begin{array}{lll}
C_1 : x_1 \vee x_2 & C_7 : \bar{x}_{10} \vee x_{11} & C_{13} : x_{16} \vee x_{18} \\
C_2 : \bar{x}_2 \vee \bar{x}_3 & C_8 : \bar{x}_8 \vee x_{12} \vee x_{13} & C_{14} : \bar{x}_{17} \vee \bar{x}_{18} \\
C_3 : \bar{x}_2 \vee \bar{x}_4 \vee \bar{x}_5 & C_9 : x_{12} \vee x_{14} & C_{15} : \bar{x}_{12} \vee x_{19} \\
C_4 : x_6 \vee \bar{x}_7 & C_{10} : \bar{x}_8 \vee \bar{x}_{13} \vee \bar{x}_{14} \vee x_{15} & C_{16} : x_7 \vee \bar{x}_{19} \vee x_{20} \\
C_5 : x_3 \vee x_5 \vee x_7 \vee x_8 & C_{11} : \bar{x}_8 \vee \bar{x}_9 \vee \bar{x}_{15} \vee \bar{x}_{16} & C_{17} : x_{15} \vee \bar{x}_{20} \vee x_{21} \\
C_6 : x_3 \vee \bar{x}_8 \vee x_9 & C_{12} : \bar{x}_{15} \vee x_{17} & C_{18} : \bar{x}_8 \vee \bar{x}_{20} \vee \bar{x}_{21}.
\end{array}
$$

Suppose the fixings $x_1 = 0$, $x_4 = 1$, $x_6 = 0$, $x_{10} = 1$, and $x_{12} = 0$ were applied in the branching steps of the DPLL procedure. This leads to a conflict generated by constraint C_{14}. The corresponding conflict graph is shown in Fig. 1. □

In the conflict graph, we distinguish between branching vertices V_B and deduced vertices $V \setminus V_B$. Branching vertices are those without incoming arcs. Each cut separating the branching vertices V_B from the conflict vertex λ gives rise to one distinct *conflict clause* (see Fig. 1), which is obtained as follows.

Let $V = V_r \cup V_c$, $V_r \cap V_c = \emptyset$, be a partition of the vertices arising from a cut with $V_B \subseteq V_r$ and $\lambda \in V_c$. V_r is called *reason side*, and V_c is called *conflict side*. The reason side's frontier $V_f := \{\ell_p \in V_r \mid \exists(\ell_p, \ell_q) \in A, \ell_q \in V_c\}$ is called *conflict set*. Fixing the literals in V_f to 1 suffices to produce the conflict. Therefore, the *conflict clause* $C_f = \bigvee_{\ell_j \in V_f} \bar{\ell}_j$ is valid for all feasible solutions of the SAT instance at hand.

Fig. 1 shows different types of cuts (labeled 'A' to 'D'), leading to different conflict clauses. The cut labelled 'A' produces clause $C_A = x_1 \vee \bar{x}_4 \vee x_6 \vee \bar{x}_{10} \vee x_{12}$ consisting of all branching variables. This clause would not help to prune the search tree, because the current subproblem is the only one where all branching variables are fixed to these specific values. The clause would never be violated again. Cut 'D' is not useful either, because clause $C_D = \bar{x}_{17} \vee \bar{x}_{18}$ is equal to the conflict-detecting clause $C_\lambda = C_{14}$ and already present in the clause database. Therefore, useful cuts must be located somewhere "in between".

There are several methods for generating useful cuts. Two of them are the so-called All-FUIP and 1-FUIP schemes which proved to be successful for SAT solving. These are explained in the following. We refer to [32] for a more detailed discussion.

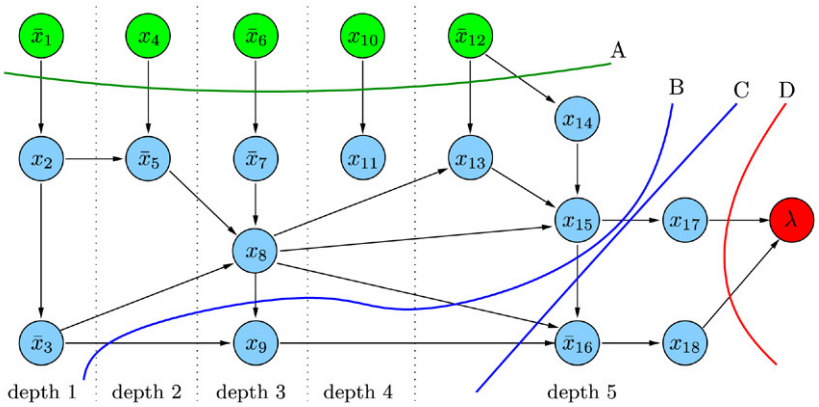

Fig. 1. Conflict graph of Example 2.2. The vertices in the top row are branching decisions, the ones below are deductions. Each cut separating the branching vertices and the conflict vertex (λ) yields a *conflict clause*.

Each vertex in the conflict graph represents a fixing of a variable that was applied in one of the nodes on the path from the root node to the current node in the search tree. The *depth level* of a vertex is the depth of the node in the search tree at which the variable was fixed. In each depth level, the first fixing corresponds to a branching vertex while all subsequent fixings are deductions. In the example shown in Fig. 1, there are five depth levels (excluding the root node) which are defined by the successive branching vertices $\bar{x}_1$, x_4, $\bar{x}_6$, x_{10}, and $\bar{x}_{12}$.

Definition 2.3 (*Unique Implication Point*). A *unique implication point* (UIP) of depth level d is a vertex $\ell_u^d \in V$ representing a fixing in depth level d, such that every path from the branching vertex of depth level d to the conflict vertex λ goes through ℓ_u^d or through a UIP $\ell_{u'}^{d'}$ of higher depth level $d' > d$. The *first unique implication point* (FUIP) of a depth level d is the UIP $\ell_u^d \neq \lambda$ that was fixed last, i.e. that is closest to the conflict vertex λ.

Note that the UIPs of the different depth levels are defined recursively, starting at the last depth level, i.e. the level of the conflict. UIPs can be identified in linear time by a single scan through the conflict graph. In the example, the FUIPs of the individual depth levels are x_{15}, x_{11}, x_8, $\bar{x}_5$, and $\bar{x}_3$, respectively.

The 1-FUIP scheme finds the first UIP in the last depth level. All literals that were fixed after this FUIP are put to the conflict side. The remaining literals and the FUIP are put to the reason side. In the example shown in Fig. 1, the FUIP of the last depth level is x_{15}. The 1-FUIP cut is the one labeled 'C'. It corresponds to the conflict clause $C_C = \bar{x}_8 \vee \bar{x}_9 \vee \bar{x}_{15}$.

The All-FUIP scheme finds the first UIP of every single depth level. From each depth level, the literals fixed after their corresponding FUIP are put to the conflict side. The remaining literals and the FUIPs are put to the reason side. In the example, this results in cut 'B' and the conflict clause $C_B = x_3 \vee \bar{x}_8 \vee \bar{x}_{15}$.

3. Conflict analysis in MIP

In this section we describe the generalization of conflict analysis of Section 2 to mixed integer programming. Recall that we consider a mixed integer program of the form

$$\text{(MIP)} \quad \max\{c^T x \mid Ax \leq b, l \leq x \leq u, x_j \in \mathbb{Z} \text{ for all } j \in I\}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c, l, u \in \mathbb{R}^n$, and $I \subseteq N = \{1, \ldots, n\}$. A branch-and-bound-based MIP solver decomposes the problem instance into subproblems typically by modifying the bounds l and u of the variables. These branching decisions may entail further deductions on the bounds of other variables. The deductions can be generated by bound-strengthening rules on linear constraints (see, e.g. Savelsbergh [27]) and may imply further deductions due to iterative application of the bound-strengthening rules.

Suppose we detected a subproblem in the branch-and-bound search tree to be infeasible, either because a deduction leads to a variable with empty domain or because the LP relaxation is infeasible. To analyze this conflict, we proceed

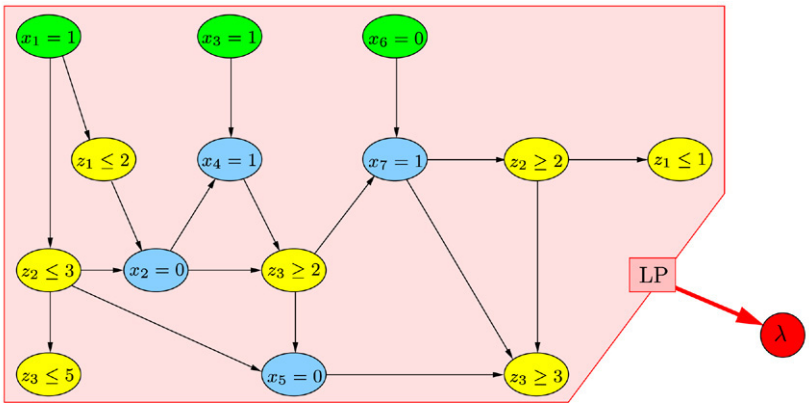

Fig. 2. Conflict graph of Example 3.1. After applying the branching decisions $x_1 = 1$, $x_3 = 1$, $x_6 = 0$, and all inferred bound changes, the LP relaxation becomes infeasible. The implications on variables z_4 and z_5 are not included in the figure.

in the same fashion as in SAT solving: we construct a conflict graph, choose a cut in this graph, and produce a conflict constraint which consists of the variables in the conflict set, i.e. in the cut's frontier. Because an MIP may contain non-binary variables, we have to extend the concept of the conflict graph: it has to represent bound changes instead of fixings of variables. This generalization is described in Section 3.1.

A conflict in SAT solving is always detected due to a single clause that became empty during the Boolean constraint propagation process (see Section 2). This conflict-detecting clause provides the links from the vertices in the conflict graph that represent fixings of variables to the conflict vertex λ. In contrast, in an LP-based branch-and-bound algorithm to solve mixed integer programs, infeasibility of a subproblem is almost always detected due to the infeasibility of its LP relaxation or due to the LP exceeding the primal bound. In this case the LP relaxation as a whole is responsible for the infeasibility. There is no single conflict-detecting constraint that defines the predecessors of the conflict vertex in the conflict graph. To cope with this situation, we have to analyze the LP in order to identify a subset of the bound changes that suffices to render the LP infeasible or bound-exceeding. The conflict vertex can then be connected to the vertices of this subset. Section 3.2 explains how to analyze infeasible LPs and how to identify an appropriate subset of the bound changes. The case of LPs having exceeded the objective bound is treated in Section 3.3.

Note that the LP analysis is related to the separation of *Dantzig cuts* [8,25], which are known to be computationally ineffective. However, the latter include *all* non-basic variables of a fractional LP solution, while the LP analysis selects only a (hopefully very small) subset of the variables in an infeasible or bound-exceeding solution as starting point for the conflict graph analysis.

After the conflict graph has been constructed, we have to choose a cut in the graph in order to define the conflict set and the resulting conflict constraint. In the case of a binary program, i.e. $I = N$, $l = 0$, $u = 1$, the conflict graph can be analyzed by the same algorithms as described in Section 2 to produce a conflict clause $C_f = \bigvee_{\ell_j \in V_f} \bar{\ell}_j$. This clause can be linearized by the set-covering constraint

$$\sum_{j:x_j \in V_f} (1 - x_j) + \sum_{j:\bar{x}_j \in V_f} x_j \geq 1, \tag{1}$$

and added to the MIP's constraint set. However, in the presence of non-binary variables, the analysis of the conflict graph may produce a conflict set that contains bound changes on non-binary variables. In this case the conflict constraint can not be linearized by the set-covering constraint (1). Section 3.4 shows how nonbinary variables can be incorporated into the conflict constraints.

3.1. Generalized conflict graph

If general integer or continuous variables are present in the problem, a bound on a specific variable could have been changed more than once on the path from the root node to the current subproblem in the search tree. A local bound change on a non-binary variable can be both reason and consequence of a deduction, similar to a fixing of a

binary variable. Therefore, we generalize the concept of the conflict graph: the vertices now represent bound changes instead of fixings. Note that there can now exist multiple vertices corresponding to the same non-binary variable in the conflict graph, each vertex representing one change of the variable's bounds.

Example 3.1. Consider the following constraints of an integer program with variables $x_1, \ldots, x_7 \in \{0, 1\}$ and $z_1, \ldots, z_5 \in \mathbb{Z}_{\geq 0}$.

$$
\begin{aligned}
2x_1 \qquad\qquad\qquad\qquad\qquad\qquad + 3z_1 + 2z_2 \qquad\qquad &\leq 9 && (2)\\
+9x_2 \qquad\qquad\qquad\qquad\qquad\qquad - \ z_1 - 2z_2 \qquad\qquad &\leq 0 && (3)\\
-3x_2 + 5x_3 - 3x_4 \qquad\qquad\qquad\qquad\qquad\qquad &\leq 4 && (4)\\
-3x_2 \qquad + 9x_4 \qquad\qquad\qquad\qquad\qquad - 2z_3 \qquad &\leq 6 && (5)\\
+9x_5 \qquad\qquad\qquad - \ z_2 + 2z_3 \qquad &\leq 8 && (6)\\
-4x_6 - 7x_7 \qquad\qquad + 2z_3 \qquad &\leq 3 && (7)\\
+5x_7 \qquad - 2z_2 \qquad\qquad &\leq 2 && (8)\\
- \ x_5 \qquad + 5x_7 \qquad + 4z_2 - 5z_3 \qquad &\leq 2 && (9)\\
x_1 - \ x_2 + \ x_3 \qquad - 2x_5 + \ x_6 \qquad - \ z_1 - 2z_2 + \ z_3 - 2z_4 + 4z_5 &\leq 1 && (10)\\
+2x_2 \qquad - \ x_4 + 3x_5 - 2x_6 \qquad - \ z_1 + 5z_2 + \ z_3 + 2z_4 - 6z_5 &\leq 2 && (11)\\
-2x_1 \qquad - 2x_3 + \ x_4 + \ x_5 \qquad + \ z_1 + 2z_2 - 2z_3 + 2z_4 - 2z_5 &\leq 1. && (12)
\end{aligned}
$$

By the basic bound-strengthening techniques of Savelsbergh [27], we can deduce upper bounds $z_1 \leq 3$, $z_2 \leq 4$, $z_3 \leq 6$, $z_4 \leq 23$, and $z_5 \leq 15$ on the general integer variables. Assume we branched on $x_1 = 1$. By applying bound-strengthening on constraint (2) we can deduce $z_1 \leq 2$ and $z_2 \leq 3$ (see Fig. 2). Using constraint (3) and the new bounds on z_1 and z_2 it follows $x_2 = 0$. By inserting the bound on z_2 into constraint (6) we can also infer $z_3 \leq 5$. After branching on $x_3 = 1$ and $x_6 = 0$ and applying the deductions that follow from these branching decisions we arrive at the situation depicted in Fig. 2 with the LP relaxation being infeasible. Note that the nonbinary variables z_i appear more than once in the conflict graph. For example, the upper bound of z_3 was changed once and the lower bound was changed twice. The implications on variables z_4 and z_5 are not included in the figure. They can be tightened to $7 \leq z_4 \leq 11$ and $4 \leq z_5 \leq 6$. □

We use the following notation in the rest of the paper. Let $\mathcal{B}_L = \{B_1, \ldots, B_K\}$ with hyperplanes $B_k = L_{j_k}^{\mu_k} := \{x \mid x_{j_k} \geq \mu_k\}$ or $B_k = U_{j_k}^{\mu_k} := \{x \mid x_{j_k} \leq \mu_k\}$, $1 \leq j_k \leq n$, $l_{j_k} \leq \mu_k \leq u_{j_k}$, $k = 1, \ldots, K$. The set $\mathcal{B}_L$ corresponds to the additional bounds imposed on the variables in the local subproblem. Thus, the subproblem is defined as:

$$
\text{(MIP}') \quad \max \left\{ c^T x \mid Ax \leq b, l \leq x \leq u, x_j \in \mathbb{Z} \text{ for all } j \in I, x \in \bigcap_{B \in \mathcal{B}_L} B \right\}.
$$

The vertices of the conflict graph correspond to the local bound changes $\mathcal{B}_L$. As before, the arcs of the graph represent the implications.

3.2. Analyzing infeasible LPs

In order to analyze the conflict expressed by an infeasible LP, we have to find a subset $\mathcal{B}_C \subseteq \mathcal{B}_L$ of the local bound changes that suffice to render the LP (together with the global bounds and rows[1]) infeasible. If all these remaining bound changes are fixings of binary variables, this already leads to a valid inequality of type (1). Furthermore, even if bound changes on nonbinary variables are present, such a subset can be used like the conflict-detecting clause in SAT to represent the conflict in the conflict graph. Analysis of this conflict graph may also lead to a valid inequality.

A reasonable heuristic to select $\mathcal{B}_C \subseteq \mathcal{B}_L$ is to try to make $|\mathcal{B}_C|$ as small as possible. This would produce a conflict graph with the least possible number of predecessors of the conflict vertex and thus (hopefully) a small

[1] In a branch-and-cut framework, we have to either remove local cuts from the LP or mark the resulting conflict constraint being only locally valid at the depth level of the last local cut remaining in the LP. Removing local rows can of course render the LP feasible again, thus making conflict analysis impossible.

conflict constraint. Unfortunately, the problem of finding the smallest subset of $\mathcal{B}_L$ with the LP still being infeasible is $\mathcal{NP}$-hard:

Definition 3.2. Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $F = \{x \mid Ax \leq b\}$. Let $\mathcal{B}_L = \{B_1, \ldots, B_K\}$ be additional bounds with $B_k = \{x \mid x_{j_k} \geq \mu_k\}$ or $B_k = \{x \mid x_{j_k} \leq \mu_k\}$, $1 \leq j_k \leq n$, for all $k = 1, \ldots, K$, such that $F \cap (\bigcap_{B\in\mathcal{B}_L} B) = \emptyset$. Then, the *minimal cardinality bound-IIS*[2] *problem* is to find a subset $\mathcal{B}_C \subseteq \mathcal{B}_L$ with

$$F \cap \left(\bigcap_{B\in\mathcal{B}_C} B\right) = \emptyset, \quad \text{and} \quad |\mathcal{B}_C| = \min_{\mathcal{B}\subseteq\mathcal{B}_L} \left\{|\mathcal{B}| \mid F \cap \left(\bigcap_{B\in\mathcal{B}} B\right) = \emptyset\right\}.$$

Proposition 3.3. *The minimal cardinality bound-IIS problem is $\mathcal{NP}$-hard.*

Proof. We provide a reduction from the *minimal cardinality IIS* problem, which is $\mathcal{NP}$-hard [4]. Given an instance $F' = (A, b)$ of the *minimal cardinality IIS* problem with $\{x \mid Ax \leq b\} = \emptyset$, the task is to find a minimal cardinality subset of the rows of $Ax \leq b$ that still defines an infeasible subsystem. Consider now the *minimal cardinality bound-IIS* problem instance $F = \{(x, s) \in \mathbb{R}^{n\times m} \mid Ax + s = b\}$ and $\mathcal{B}_L = \{B_1, \ldots, B_m\}$ with $B_i = \{(x, s) \mid s_i \geq 0\}$ for $i = 1, \ldots, m$. Then, for each subset $\mathcal{B} \subseteq \mathcal{B}_L$, the index set $I_\mathcal{B} = \{i \mid B_i \in \mathcal{B}\}$ defines an infeasible subsystem of F' if and only if $F \cap (\bigcap_{B\in\mathcal{B}} B) = \emptyset$. Hence, there exists a one-to-one correspondence between the set of solutions of $(F, \mathcal{B}_L)$ and the one of F'. Because $|I_\mathcal{B}| = |\mathcal{B}|$, the optimal solution of $(F, \mathcal{B}_L)$ defines an optimal solution of F'. □

There are various heuristics for *minimal cardinality IIS* (see [24]). These can easily be specialized to the *minimal cardinality bound-IIS* problem. We implemented a preliminary version of a heuristic based on one of these methods which applies the Farkas lemma, but the overhead in running time was very large. Therefore, we employ very simple heuristics using the LP information at hand, which are described in the following.

First, we will only consider the case with the global lower bounds l and local lower bounds $\tilde{l}$ being equal to $l = \tilde{l} = 0$. We further assume that each component of the global upper bounds u was tightened at most once to obtain the local upper bounds $\tilde{u} \leq u$. Thus, the set of local bound changes $\mathcal{B}_L$ consists of at most one bound change for each variable.

Suppose the local LP relaxation

$$\text{(P)} \quad \max\{c^T x \mid Ax \leq b, 0 \leq x \leq \tilde{u}\}$$

is infeasible. Then its dual:

$$\text{(D)} \quad \min\{b^T y + \tilde{u}^T r \mid A^T y + r \geq c, (y, r) \geq 0\}$$

has an unbounded ray, i.e. $(\bar{y}, \bar{r}) \geq 0$ with $A^T \bar{y} + \bar{r} \geq 0$ and $b^T \bar{y} + \tilde{u}^T \bar{r} < 0$. Note that the dual LP does not need to be feasible.

We can aggregate the rows and bounds of the primal LP with the non-negative weights $(\bar{y}, \bar{r})$ to get the following proof of infeasibility:

$$0 \leq (\bar{y}^T A + \bar{r}^T)x \leq \bar{y}^T b + \bar{r}^T \tilde{u} < 0. \tag{13}$$

Now we try to relax the bounds as much as possible without loosing infeasibility. Note that the left-hand side of (13) does not depend on $\tilde{u}$. Relaxing $\tilde{u}$ to some $\hat{u}$ with $\tilde{u} \leq \hat{u} \leq u$ increases the right- hand side of (13), but as long as $\bar{y}^T b + \bar{r}^T \hat{u} < 0$, the relaxed LP

$$(\hat{P}) \quad \min\{c^T x \mid Ax \leq b, 0 \leq x \leq \hat{u}\}$$

is still infeasible with the same infeasibility proof $(\bar{y}, \bar{r})$. This leads to the following heuristic to produce a relaxed upper bound vector $\hat{u}$ with the corresponding LP still being infeasible.

[2] IIS: irreducible inconsistent subsystem (an infeasible subsystem all of whose proper subsystems are feasible).

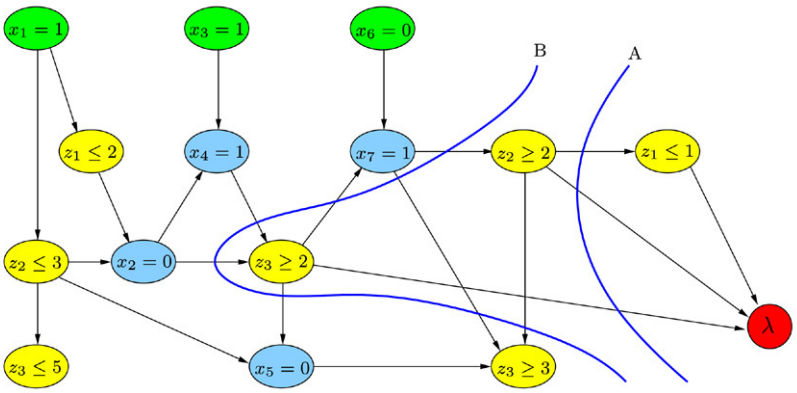

Fig. 3. Conflict graph of Example 3.1 after the infeasible LP was analyzed. Cut 'A' is the 1-FUIP cut. Cut 'B' was constructed by moving the nonbinary variables of the conflict set of cut 'A' to the conflict side.

Algorithm 3.4. Let $\max\{c^T x \mid Ax \leq b, 0 \leq x \leq \tilde{u} \leq u\}$ be an infeasible LP with dual ray $(\bar{y}, \bar{r})$.

1. Set $\hat{u} := \tilde{u}$, and calculate the infeasibility measure $d := \bar{y}^T b + \bar{r}^T \hat{u} < 0$.
2. Select a variable j with $\hat{u}_j < u_j$ and $d_j := d + \bar{r}_j(u_j - \tilde{u}_j) < 0$. If no such variable exists, stop.
3. Set $\hat{u}_j := u_j$, update $d := d_j$, and go to 2.

In the general case of multiple bound changes on a single variable, we have to process these bound changes step by step, always relaxing to the previously active bound. In the presence of nonzero lower bounds the reduced costs r may also be negative. In this case, we can split up the reduced cost values into $r = r^u - r^l$ with $r^u, r^l \geq 0$. It follows from the Farkas lemma that $r^u \cdot r^l = 0$. The infeasibility measure d of the dual ray has to be defined in Step 1 as $d := \bar{y}^T b + (\bar{r}^u)^T \hat{u} + (\bar{r}^l)^T \hat{l}$. A local lower bound $\tilde{l}$ can be relaxed in the same way as an upper bound $\tilde{u}$, where u has to be replaced by l in the formulas of Steps 2 and 3.

Example 3.5 (*Continued from Example* 3.1). After applying the deductions on the bounds of the variables in Example 3.1, the LP relaxation is infeasible. Let $y_{(i)}$ denote the dual variable of constraint (i) and r_j the reduced cost value of variable j. Then the dual ray $\bar{y}_{(10)} = 2$, $\bar{y}_{(11)} = 1$, $\bar{y}_{(12)} = 1$, $\bar{r}_{z_1} = 2$, $\bar{r}_{z_2} = -3$, $\bar{r}_{z_3} = -1$, and the remaining coefficients set to zero proves the infeasibility of the LP. In Step 1 of Algorithm 3.4 the infeasibility measure is calculated as

$$\begin{array}{rcccccccccccl} d & = & \bar{y}_{(10)} b_{(10)} & + & \bar{y}_{(11)} b_{(11)} & + & \bar{y}_{(12)} b_{(12)} & + & \bar{r}^u_{z_1} \tilde{u}_{z_1} & - & \bar{r}^l_{z_2} \tilde{l}_{z_2} & - & \bar{r}^l_{z_3} \tilde{l}_{z_3} \\ & = & 2 \cdot 1 & + & 1 \cdot 2 & + & 1 \cdot 1 & + & 2 \cdot 1 & - & 3 \cdot 2 & - & 1 \cdot 3 \quad = -2. \end{array}$$

In Step 2, all local bounds except the upper bound of z_1 and the lower bounds of z_2 and z_3 can be relaxed to the corresponding global bounds, because their reduced cost values in the dual ray are zero. Additionally, the lower bound of z_3 can be relaxed from 3 to 2, which was the lower bound before $z_3 \geq 3$ was deduced. This relaxation increases d by 1 to $d = -1$. No further relaxations are possible without increasing d to $d \geq 0$. Thus, the local bounds $z_1 \leq 1$, $z_2 \geq 2$, and $z_3 \geq 2$ are identified as initial reason for the conflict, and the "global" arc from the LP to the conflict vertex in Fig. 2 can be replaced by three arcs as shown in Fig. 3. The 1-FUIP scheme applied to the resulting conflict graph yields the cut labeled 'A' and the conflict constraint

$$(z_2 \leq 1) \vee (z_3 \leq 1).$$

Note that the involved variables z_2 and z_3 are nonbinary. Section 3.4 shows how to proceed in this situation. □

3.3. Analyzing LPs exceeding the primal bound

In principle, the case of an LP exceeding the primal bound can be handled as in the previous section by adding an appropriate objective bound inequality to the constraint system. In the implementation, however, we use the dual solution directly as a proof of objective bound excess. Then, we relax the bounds of the variables as long as the dual

solution's objective value stays below the primal bound. Again, we describe the case with $l = \tilde{l} = 0$ and with at most one upper bound change per variable on the path from the root node to the local subproblem.

Suppose, the local LP relaxation

$$\text{(P)} \quad \max\{c^T x \mid Ax \leq b, 0 \leq x \leq \tilde{u}\}$$

exceeds (i.e. falls below) the primal objective bound $\bar{z}$. Then the dual

$$\text{(D)} \quad \min\{b^T y + \tilde{u}^T r \mid A^T y + r \geq c, (y, r) \geq 0\}$$

has an optimal solution $(\bar{y}, \bar{r})$ with $b^T \bar{y} + \tilde{u}^T \bar{r} \leq \bar{z}$. Note that the variables' upper bounds $\tilde{u}$ do not affect dual feasibility. Thus, after relaxing the upper bounds $\tilde{u}$ to a vector $\hat{u}$ with $\tilde{u} \leq \hat{u} \leq u$ that also satisfies $b^T \bar{y} + \hat{u}^T \bar{r} \leq \bar{z}$, the LP's objective value stays below the primal objective bound.

After relaxing the bounds, the vector $(\bar{y}, \bar{r})$ is still feasible, but not necessarily optimal for the dual LP. We may resolve the dual LP in order to get a stronger dual bound which can be used to relax further local upper bounds. The following algorithm summarizes this procedure.

Algorithm 3.6. Let $\max\{c^T x \mid Ax \leq b, 0 \leq x \leq \tilde{u} \leq u\}$ be an LP, $\bar{z}$ a primal objective bound, and $(\bar{y}, \bar{r})$ a dual feasible solution with $b^T \bar{y} + \tilde{u}^T \bar{r} \leq \bar{z}$.

1. Set $\hat{u} := \tilde{u}$.
2. Calculate the bound excess measure $d := b^T \bar{y} + \hat{u}^T \bar{r} - \bar{z} \leq 0$.
3. Select a variable j with $\hat{u}_j < u_j$ and $d_j := d + \bar{r}_j(u_j - \tilde{u}_j) \leq 0$. If no such variable exists, go to 5.
4. Set $\hat{u}_j := u_j$, update $d := d_j$, and go to 3.
5. (optional) If at least one upper bound was relaxed in the last iteration, resolve the dual LP to get the new dual solution $(\bar{y}, \bar{r})$, and go to 2.

3.4. Conflict constraints with non-binary variables

Despite the technical issue of dealing with bound changes instead of fixings in the conflict graph, there is also a principle obstacle in the presence of non-binary variables, which is the construction of the conflict constraint if nonbinary variables appear in the conflict set.

The conflict graph analysis yields a conflict set, which is a subset $\mathcal{B}_f \subseteq \mathcal{B}_L$ that together with the global bounds l and u suffices to render the current subproblem infeasible. This conflict set leads to the conflict constraint

$$\bigvee_{L_j^\mu \in \mathcal{B}_f} (x_j < \mu) \vee \bigvee_{U_j^\mu \in \mathcal{B}_f} (x_j > \mu).$$

Bounds on continuous variables x_j, $j \notin I$, would remain strict inequalities which cannot be handled using floating point arithmetics and feasibility tolerances. Therefore, we have to relax the bounds on continuous variables by allowing equality in the conflict constraint. This leads to the conflict constraint:

$$\bigvee_{\substack{L_j^\mu \in \mathcal{B}_f \\ j \in I}} (x_j \leq \mu - 1) \vee \bigvee_{\substack{U_j^\mu \in \mathcal{B}_f \\ j \in I}} (x_j \geq \mu + 1) \vee \bigvee_{\substack{L_j^\mu \in \mathcal{B}_f \\ j \notin I}} (x_j \leq \mu) \vee \bigvee_{\substack{U_j^\mu \in \mathcal{B}_f \\ j \notin I}} (x_j \geq \mu). \tag{14}$$

As shown in the introduction of Section 3, this constraint can be linearized by the set-covering constraint (1) if all conflict variables are binary. However, if a non-binary variable is involved in the conflict, we cannot use such a simple linearization. In this case, (14) can be modelled with the help of auxiliary variables: $y_j^\mu, z_j^\mu \in \{0, 1\}$:

$$\begin{aligned}
&\sum_{L_j^\mu \in \mathcal{B}_f} y_j^\mu + \sum_{U_j^\mu \in \mathcal{B}_f} z_j^\mu \geq 1 \\
&x_j - (\mu - 1) y_j^\mu \leq 0 && \text{for all } L_j^\mu \in \mathcal{B}_f, j \in I \\
&x_j - (\mu + 1) z_j^\mu \geq 0 && \text{for all } U_j^\mu \in \mathcal{B}_f, j \in I \\
&x_j - \mu y_j^\mu \leq 0 && \text{for all } L_j^\mu \in \mathcal{B}_f, j \notin I \\
&x_j - \mu z_j^\mu \geq 0 && \text{for all } U_j^\mu \in \mathcal{B}_f, j \notin I
\end{aligned} \tag{15}$$

The question arises, whether the potential gain in the dual bound justifies the expenses in adding system (15) to the LP. Many fractional points violating conflict constraint (14) cannot even be separated by (15) if the integrality restrictions on the auxiliary variables are not enforced through other cutting planes or branching. This suggests that system (15) is probably very weak, although we did not verify this hypothesis by computational studies.

We have the following two possibilities to avoid adding system (15) to the LP. Either we use conflict constraints involving non-binary variables only for domain propagation but not for cutting plane separation, or we prevent the generation of conflict constraints with non-binary variables. The former demands the possibility of including non-linear constraints into the underlying MIP framework. Our implementation is based on SCIP [1] which provides support for arbitrary constraints. For the latter option we have to modify the cut selection rules in the conflict graph analysis such that the non-binary variables are not involved in the resulting conflict set. This can be achieved by moving the bound changes on non-binary variables from the reason side's frontier to the conflict side of the cut. The following example illustrates this idea.

Example 3.7 (*Continued from Examples* 3.1 *and* 3.5). Fig. 3 shows the conflict graph of Example 3.1 after branching on $x_1 = 1$, $x_3 = 1$, and $x_6 = 0$. The analysis of the LP relaxation identified $z_1 \leq 1$, $z_2 \geq 2$, and $z_3 \geq 2$ as sufficient to cause an infeasibility in the LP (see Example 3.5). The 1-FUIP cut selection scheme leads to the cut labelled 'A' in the figure. The corresponding conflict constraint is:

$$(z_2 \leq 1) \vee (z_3 \leq 1).$$

Because there are non-binary variables involved in the conflict constraint, it cannot be linearized by the set-covering constraint (1). To avoid the introduction of the auxiliary variables of System (15), the bound changes $z_2 \geq 2$ and $z_3 \geq 2$ are put to the conflict side, resulting in cut 'B'. Thus, the conflict constraint that is added to the constraint database is:

$$(x_2 = 1) \vee (x_4 = 0) \vee (x_7 = 0),$$

which can be written as

$$x_2 + (1 - x_4) + (1 - x_7) \geq 1$$

in terms of a set-covering constraint. □

Since branching vertices must be located on the reason side, the bound changes representing branching decisions on non-binary variables cannot be moved to the conflict side. In this case, we just remove the bound change from the conflict set, thereby destroying the global validity of the resulting conflict clause. The clause can therefore only be added to the local subtree which is rooted at the node where the bound change on the non-binary variable was applied.

4. Computational results

In this section we examine the computational effectiveness of conflict analysis. All calculations were performed on a 3.8 GHz Pentium-4 workstation with 2 GB RAM. In all runs we used a time limit of 3600 CPU seconds and a memory limit of 1.5 GB.

The conflict analysis techniques described in Sections 2 and 3 were implemented into the constraint and mixed integer programming framework SCIP version 0.81f [1] using CPLEX 10.0 to solve the LP relaxations. The conflict graph analysis produces one conflict constraint for each depth level with the FUIP scheme. This includes the constraints of the 1-FUIP and All-FUIP schemes as extreme cases (see Section 2). For LP conflicts, the initial conflict set generated by the LP analysis is also used to create a conflict constraint. Conflict constraints that are dominated by others produced for the same conflict are deleted.

We only separate cutting planes in the root node, which seems to yield the best performance on most MIP instances using SCIP.[3] The generated conflict constraints take part in the propagation process inside the subproblem solving loop. If they consist of only binary variables they may also be used as cutting planes.

[3] In the current version, SCIP separates Gomory MIR cuts, strong CG cuts, c-MIR cuts, lifted knapsack cover cuts, implied bound cuts, and clique cuts (see Wolter [30]).

Because the recorded conflict constraints increase the costs for processing the subproblems, we try to only keep the "useful" conflict clauses. We implemented a constraint-aging mechanism to identify useless conflict constraints. Constraints are deleted, if they did not help to tighten the LP relaxation during propagation or separation in a number of consecutive iterations. This iteration limit is adjusted dynamically. For instance, longer constraints are discarded earlier than constraints with fewer variables.

4.1. Test set

Our first two test sets consist of instances from MIPLIB 2003 [2,3], instances collected by Mittelmann [22], and instances described in Danna, Rothberg, and Le Pape [11]. From this set of 121 feasible MIP instances, we selected all problems that could be solved in one hour CPU time by at least one of CPLEX 10.0,[4] SCIP in default settings, or SCIP with conflict analysis. These 76 instances are divided into the first test set consisting of the 25 binary programs (i.e. where all variables after SCIP's preprocessing are binary) and the second set consisting of the remaining 51 general mixed integer programs.

As a third test set, we use IP models of the so-called *property checking problem* which arises in chip design verification (see, e.g. [9]). The task is to prove the validity of a certain property for a given chip design. The problem can be modelled as an IP, where each feasible solution represents a *counter-example* of the property. Hence, in order to prove that the property holds, one has to show the *infeasibility* of the IP.

The data are for a very simple arithmetic logical unit (ALU) of different register widths ranging from 4 to 8 bits and with different properties to be checked. All of the ALU instances investigated here correspond to valid properties, i.e. all the IP instances are infeasible. The instances, the ALU model, and the property definitions can be found in the *contributed instances* section of MIPLIB 2003.

The considered ALU design includes signed and unsigned multiplication of the two input registers. The internal calculations for multiplying the n-bit input registers operate on $2n$-bit variables. Therefore, the IP instances include integer variables and matrix coefficients that are in the range of 2^{2n}. In order to overcome the numerical difficulties arising from such large values, we had to set the feasibility and integrality tolerances of the solvers for this second test set to 10^{-9}.

A fourth test set consists of the 'Enlight' instances, again from the *contributed instances* section of MIPLIB 2003. They describe a combinatorial game by FeejoSoft,[5] which is played on an $n \times n$ board. The model contains n^2 binary and n^2 integer variables, the latter with domains $\{0, 1, 2, 3\}$, and n^2 constraints. Some of the instances are feasible, some of them are not.

4.2. Description of the results

We present detailed results on the four test sets comparing CPLEX, SCIP in default settings, and SCIP with conflict analysis using the following strategy which turned out to produce the best results on our test set:

- We only generate constraints for conflicting propagations and infeasible LP relaxations, but not for bound-exceeding LPs.
- Conflict constraints with non-binary variables are kept as they are instead of enforcing pure binary constraints.
- We do not add conflict constraints as cutting planes to the LPs, even if they consist of binary variables only.

In a second computational experiment we compare this strategy with other parameter choices for conflict analysis.

Tables 1 and 2 show the results on the first test set of feasible MIP instances. Table 1 contains the pure binary instances while the instances of Table 2 are general MIPs. Columns 'Nodes' and 'Time' show the number of branching nodes and the total time in CPU seconds needed to solve the instances. Values marked with a '>' denote that the instance could not be solved within the time or memory limit. The additional columns 'Confs' and '∅Vars' show the number of conflict constraints generated for each instance and the average number of variables per conflict constraint.

[4] CPLEX was run with default settings, except that 'absolute mipgap' was set to 10^{-9} and 'relative mipgap' to 0.0, which are the corresponding values in SCIP.

[5] 'Enlight' is available as freeware for PDA from http://www.feejo.com.

Table 1
Results for the binary programming test set

Name	Vars	CPLEX 10.0		SCIP 0.81f (default)		SCIP 0.81f (conflict)			
		Nodes	Time	Nodes	Time	Nodes	Time	Confs	∅Vars
10teams	1 600	22	4.4	697	33.6	**358**	**31.4**	92	225.8
air04	7 370	128	14.8	134	173.2	134	**169.9**	13	380.8
air05	6 120	288	12.0	228	122.8	228	**119.9**	4	887.8
cap6000	4 109	4 565	15.9	2 615	28.9	**2 598**	28.3	299	72.3
disctom	9 991	50	189.9	1	67.3	1	67.2	0	–
fast0507	62 999	13 997	3082.7	>735	>3600.0	>743	>3600.0	0	–
fiber	975	72	0.3	63	5.2	63	5.2	0	–
harp2	1 034	1 071 588	1113.3	598 447	1048.3	**481 835**	**976.7**	14 583	65.2
misc07	232	11 049	9.6	30 934	47.3	**25 528**	**41.9**	3 431	39.9
nw04	76 309	126	20.1	7	164.4	7	**158.7**	0	–
p2756	2 635	16	0.4	64	4.4	64	4.3	2	4.5
l152lav	1 989	227	1.2	46	5.6	46	5.4	7	109.4
stein45	45	51 041	19.7	52 400	48.0	52 400	**46.5**	3	25.3
eilD76	1 823	122	11.2	3 838	92.7	3 838	**89.1**	0	–
irp	19 941	15	8.4	52	27.5	52	27.0	0	–
neos1	1 728	1	1.2	1	6.3	1	6.0	0	–
nug08	1 632	54	17.2	3	74.9	3	74.9	0	–
qap10	4 150	10	160.9	5	**412.2**	5	420.4	0	–
acc-0	1 620	1	0.1	1	15.4	1	14.9	0	–
acc-1	1 620	1	5.0	1	30.2	1	29.7	0	–
acc-2	1 620	1	7.5	77	90.4	77	**87.7**	0	–
acc-3	1 570	29	50.8	**75**	**172.9**	247	283.0	107	74.6
acc-4	1 570	37	72.1	**236**	**403.8**	445	448.6	475	135.9
acc-5	1 017	86	94.4	8 592	2145.1	**5 068**	**1595.0**	6 560	70.5
acc-6	1 018	453	335.4	**18**	77.9	19	**75.7**	18	45.9
Total (25 instances)		1 153 979	5248.9	699 270	8899.6	573 762	8415.8		
Geom. mean		113	18.6	126	74.3	127	73.4		
Shifted geom. mean		1 173	54.4	1 314	120.2	1 244	119.5		

The winner between SCIP default and SCIP with conflict analysis is marked in bold face (differences below one second are ignored).

The summary at the bottom gives the total number of nodes and seconds to process the whole test set, the *geometric mean*

$$\gamma(x_1, \ldots, x_k) := \left(\prod_{i=1}^{k} \max\{x_i, 1\} \right)^{\frac{1}{k}},$$

and the *shifted geometric mean*

$$\gamma_s(x_1, \ldots, x_k) := \left(\prod_{i=1}^{k} (x_i + s) \right)^{\frac{1}{k}} - s$$

over all instances, respectively. The influence of the very easy instances is more and more reduced for increasing parameter s in the latter measure. We select $s = 1000$ for the nodes and $s = 60$ for the time in order to focus on the harder instances and to disregard small absolute differences.

The comparison with CPLEX indicates that SCIP's performance (using CPLEX as LP solver) is not strictly competitive, but not far away from a state-of-the-art MIP solver. On the binary instances, SCIP is a factor of about 4.0 slower than CPLEX in the geometric mean, which reduces to a factor of 2.2 in the shifted geometric mean. On the general MIPs, the factors are 2.3, and 1.5. SCIP with default settings could not solve 3 instances while CPLEX with default settings could not solve 5 instances in the time and memory limit, although `manna81` can also be solved by CPLEX in the root node if aggressive cut separation is applied.

The results of SCIP with and without conflict analysis are very similar on the binary instances. Minor improvements due to conflict analysis can be observed on `10teams`, `harp2`, `misc07`, and `acc-5`, while slightly worse results were

Table 2
Results for the mixed integer programming test set

Name	Vars	CPLEX 10.0		SCIP 0.81f (default)		SCIP 0.81f (conflict)			
		Nodes	Time	Nodes	Time	Nodes	Time	Confs	∅Vars
aflow30a	842	11 832	30.5	**7 904**	**52.9**	8 800	58.9	52	3.6
fixnet6	877	132	1.0	19	0.8	19	0.8	0	–
gesa2-o	1 224	3 380	4.8	**386**	9.7	392	10.0	7	4.6
gesa2	1 224	39	0.5	148	5.1	**147**	5.1	3	4.3
manna81	3 321	>1 129 905	>3600.0	1	3.3	1	3.4	0	–
mas74	150	4 451 916	1254.1	4 431 632	**1432.2**	**4 298 895**	1485.3	15 190	23.8
mas76	150	660 320	122.2	**341 871**	**106.8**	351 121	120.5	2 050	25.2
mod011	6 764	45	69.5	2 433	**219.6**	2 433	221.0	0	–
modglob	387	245	0.3	1 098	2.0	1 098	2.0	0	–
mzzv11	8 878	1 873	217.8	**3 778**	**1175.8**	4 742	1471.8	308	49.6
mzzv42z	10 390	183	52.5	1 222	584.1	**996**	**512.3**	36	57.6
noswot	120	4 717 721	1569.0	6 940 311	2431.6	**1 093 715**	**524.7**	337 363	11.1
pk1	86	338 108	82.8	**247 450**	**98.7**	247 473	107.4	1 727	26.4
pp08a	240	618	1.1	2 604	4.9	2 604	4.9	0	–
pp08aCUTS	240	1 372	2.1	1 197	3.5	1 197	3.5	0	–
qiu	840	2 371	31.5	12 153	226.3	12 153	227.1	2	15.0
rout	555	41 345	88.1	21 020	50.8	**18 393**	**47.4**	1 074	16.6
set1ch	666	326	0.5	215	2.3	215	2.2	0	–
vpm2	181	3 269	1.6	**9 994**	9.5	11 281	9.9	18	10.3
bell3a	110	27 344	5.7	44 453	**40.0**	**44 258**	42.3	1 516	27.5
bell5	94	1 076	0.3	**5 996**	3.3	6 196	3.7	375	20.4
gesa3	1 128	45	0.7	224	6.2	224	6.2	1	5.0
gesa3_o	1 128	55	0.9	437	11.4	**339**	10.6	2	3.5
ran8x32	512	8 993	19.0	16 019	37.3	**16 017**	37.5	10	8.1
ran10x26	520	22 086	52.6	**29 904**	80.2	29 906	80.1	3	14.3
ran12x21	504	53 156	125.9	110 612	208.1	**110 496**	207.7	7	16.9
ran13x13	338	14 535	22.4	62 051	91.4	**61 903**	91.2	8	12.1
binkar10_1	1 444	7 992	57.6	>474 018	>2345.4	>384 333	>2103.6	13 271	5.1
mas284	150	24 079	13.3	**18 059**	**29.6**	18 924	34.2	412	57.6
prod1	213	61 292	54.4	57 643	**47.8**	**57 076**	51.6	12 817	11.2
bc1	1 002	5 834	155.6	18 314	831.2	**17 412**	**788.5**	0	–
bienst1	505	7 918	325.7	**9 224**	50.6	10 479	51.5	23	8.0
bienst2	505	>92 887	>3600.0	90 815	567.3	**88 272**	**565.4**	225	7.6
dano3_3	13 873	15	129.4	19	**192.2**	19	199.2	0	–
dano3_4	13 873	27	128.4	41	**250.1**	41	259.5	0	–
dano3_5	13 873	367	552.8	203	**577.1**	203	580.6	0	–
mkc1	5 314	14 265	77.0	>408 554	>3600.0	>519 586	>3600.0	12 615	2.9
neos2	1 516	951	6.3	44 078	188.8	**30 344**	**142.7**	292	19.3
neos3	2 582	4 104	21.0	397 610	2470.5	**385 629**	**2221.4**	3 302	21.3
neos4	18 799	55	5.0	13	**142.1**	13	146.4	0	–
neos5	19 229	55	4.9	15	107.0	15	106.5	0	–
neos6	8 563	>36 365	>3600.0	3 747	473.5	**1 077**	**244.6**	87	224.5
neos7	1 538	14 159	53.1	54 745	557.6	**40 697**	**404.7**	176	11.4
seymour1	1 255	6 419	750.2	4 255	**895.2**	4 255	898.6	0	–
swath1	6 320	5 069	16.1	**499**	67.9	591	67.7	100	127.3
swath2	6 320	26 492	70.5	5 323	120.8	**3 664**	**108.1**	662	186.0
ic97_tension	469	234 803	264.2	143 196	206.8	**48 944**	**97.2**	2 833	5.0
nh97_potential	1 180	>202 212	>3600.0	286 079	1060.0	**206 145**	**1049.7**	102 818	10.5
nh97_tension	726	>1 702 946	>3600.0	187 222	449.0	**6 626**	**33.7**	1 811	4.3
neos10	793	28	8.7	7	**183.7**	7	190.9	0	–
neos20	613	2 587	27.9	4 048	37.4	**1 176**	**20.9**	753	12.5
Total (51 instances)		13 943 211	24 479.3	14 502 859	22 349.4	8 150 542	19 264.6		
Geom. mean		4 668	39.6	5 444	92.1	4 477	81.6		
Shifted geom. mean		10 142	110.9	12 685	171.2	10 512	149.7		

The winner between SCIP default and SCIP with conflict analysis is marked in bold face (differences below one second are ignored). Note that `binkar10_1` could not be solved by SCIP due to the memory limit.

Table 3
Results for the ALU test set

Name	Vars	CPLEX 10.0		SCIP 0.81f (default)		SCIP 0.81f (conflict)			
		Nodes	Time	Nodes	Time	Nodes	Time	Confs	∅Vars
alu4_1	225	1 982	2.9	10 761	17.4	**295**	**5.6**	217	6.9
alu4_2	150	54	0.2	3	0.7	3	0.7	0	–
alu4_6	168	2 467	1.2	445	2.4	**32**	1.7	24	3.8
alu4_7	154	1 933	1.5	2 027	4.6	**1 451**	4.4	1 313	7.9
alu4_8	176	7 563	4.0	5 449	7.8	**2 315**	**5.3**	2 103	9.7
alu5_1	230	36 983	27.0	50 453	63.9	**440**	**5.7**	335	8.0
alu5_2	155	741	0.5	1	0.1	1	0.1	0	–
alu5_6	168	357	0.4	8	1.6	8	1.6	1	3.0
alu5_7	160	7 497	4.0	22 067	18.8	**11 387**	**14.7**	11 164	9.0
alu5_8	184	10 385	5.7	121 187	113.3	**45 621**	**55.4**	46 547	11.8
alu6_1	235	43 663	29.5	8 851	16.3	**315**	**6.0**	288	7.6
alu6_2	160	2 549	1.4	1	0.1	1	0.1	0	–
alu6_6	174	26	0.3	13	1.5	13	1.5	6	3.8
alu6_7	166	42 702	25.6	67 941	50.7	**23 985**	**26.4**	23 790	10.4
alu6_8	190	23 715	15.1	235 817	213.1	**92 539**	**112.2**	101 720	12.9
alu7_1	240	392 658	258.0	10 339	21.1	**340**	**6.3**	215	7.1
alu7_2	165	14 005	7.0	1	0.1	1	0.1	0	–
alu7_6	178	24 389	8.3	16	1.3	**14**	1.2	4	3.5
alu7_7	182	79 701	65.1	2 992 365	1972.0	**245 790**	**368.7**	215 426	11.4
alu7_8	200	63 204	70.5	780 043	620.4	**64 161**	**120.8**	83 757	15.5
alu8_1	245	1 885 899	1526.9	558 637	731.9	**30**	**3.6**	15	6.3
alu8_2	170	78 941	48.7	1	0.1	1	0.1	0	–
alu8_6	183	377 529	175.9	24	1.4	**14**	1.3	5	3.4
alu8_7	188	320 789	184.3	>4170 573	>3600.0	**436 860**	**1382.5**	407 775	12.4
alu8_8	207	1 392 568	1751.4	2 547 685	**2199.4**	>1 980 526	>3600.0	2 183 290	15.2
Total (25 instances)		4 812 300	4215.3	11 584 708	9660.0	2 906 143	5725.9		
Geom. mean		14 675	14.2	1 948	18.0	384	8.8		
Shifted geom. mean		21 738	50.1	15 018	82.4	4 129	42.4		

All instances are infeasible.

achieved on `acc-3` and `acc-4`. For general MIPs, conflict analysis yielded a reduction in the number of nodes of 18% in the geometric mean, which lead to a speed-up of 11%. Conflict analysis performed very well on `noswot`, `neos6`, `ic97_tension`, `nh97_tension`, and `neos20`. Small enhancements can be seen for `mzzv42z`, `rout`, `bc1`, `neos2`, `neos3`, `neos7`, `swath2`, and `nh97_potential`. There seem to be no major disadvantages in using conflict analysis, although the performance decreased slightly on `aflow30a`, `mas76`, `mzzv11`, and `pk1`.

Table 3 shows the results for the infeasible ALU instances. About 80% of the variables in these instances are binary, and the remaining 20% are of general integer type. The width of the input and output registers range from 4 to 8 bits, reflected by the name of the instance. For each width, eight different properties were checked. Properties 3–5 are trivial for all of the three solvers, and they are not listed in the table.

On this test set, conflict analysis clearly outperforms the default MIP settings in both branching nodes and solving time. In the geometric mean, it reduced the number of nodes by 80% and the solving time by 50%. SCIP with conflict analysis was faster and needed fewer nodes than SCIP with default settings on all instances except the easy properties 2 and 6 and instance `alu8_8`. Most notably, property 1 can be solved with conflict analysis quite easily in a few branching nodes even for 8-bit input registers, while SCIP with default settings needs over half a million nodes to prove the property, i.e. to show the infeasibility of the 8-bit instance. In contrast to the conflict constraints generated for the feasible MIP instances in Tables 1 and 2, the constraints found for the ALU instances contain only very few variables. Therefore, they cut off a much larger part of the search tree, which is a possible explanation for the success of conflict analysis on these instances. However, the sizes of the conflict constraints and the performance of conflict analysis for the instances of Tables 1 and 2 do not seem to be correlated.

The results for the 'Enlight' IP instances are given in Table 4. Again, conflict analysis yielded substantial improvements over the default settings. The largest improvement was achieved on the infeasible instance `enlight9`.

Table 4
Results for the 'Enlight' test set

Name	Vars	CPLEX 10.0		SCIP 0.81f (default)		SCIP 0.81f (conflict)			
		Nodes	Time	Nodes	Time	Nodes	Time	Confs	∅Vars
enlight4	32	66	0.0	1	0.0	1	0.0	0	–
enlight5	50	940	0.2	1 289	1.3	**666**	1.3	582	7.4
enlight6	72	3 628	0.7	7 734	2.9	**2 038**	**1.8**	1 577	8.0
enlight7	98	2 951	1.0	8 570	4.0	**2 619**	**2.4**	1 853	8.1
enlight8	128	80 928	25.1	360 397	127.7	**80 556**	**43.7**	62 959	9.2
enlight9	162	3 871 450	1280.5	504 943	182.7	**41 519**	**24.8**	49 338	12.5
enlight10	200	>5 061 019	>3600.0	7 708 326	3171.4	**1 295 582**	**893.3**	990 765	9.6
Total (7 instances)		9 020 982	4907.6	8 591 260	3489.9	1 422 981	967.3		
Geom. mean		19 450	14.3	14 258	19.6	3 964	9.1		
Shifted geom. mean		33 712	117.7	41 957	95.3	12 516	42.5		

Instances `enlight4`, `enlight5`, and `enlight9` are infeasible, the remaining instances are feasible.

Table 5
Geometric means of results for various parameter settings of conflict analysis

Settings							BPs			MIPs			ALU			Enlight		
prop	inf	bdex	sb	opt	bin	cuts	Nodes	Time	Fails	Nodes	Time	Fails	Nodes	Time	Fails	Nodes	Time	Fails
✓	✓						127	73.4	1	4 477	81.6	2	384	8.8	1	3 964	9.1	0
✓	✓					✓	129	74.0	2	4 980	89.4	2	352	8.8	0	5 199	12.1	0
✓	✓				✓		127	73.4	1	4 680	83.2	2	409	8.2	0	7 364	11.7	0
✓	✓				✓	✓	129	73.9	2	4 386	81.3	2	425	8.5	0	6 673	12.1	0
✓	✓	✓					125	75.2	1	4 562	90.8	3	370	8.5	0	3 857	9.1	0
✓	✓	✓		✓			121	82.5	1	4 352	101.1	3	378	8.6	0	3 911	9.2	0
✓	✓	✓	✓				135	80.7	1	4 774	95.2	3	246	7.6	0	4 361	9.9	0
✓	✓	✓	✓	✓			123	91.5	1	4 686	113.7	3	237	7.3	0	4 400	10.1	0
SCIP 0.81f (default settings)							126	74.3	1	5 444	92.1	2	1 948	18.0	1	14 258	19.6	0
CPLEX 10.0							113	18.6	0	4 668	39.6	5	14 675	14.2	0	19 450	14.3	1

Like for the ALU test set, the conflict constraints are rather small. The run on `enlight9`, however, produced the largest conflict constraints.

In addition to the discussed conflict analysis settings, we experimented with various other settings. Table 5 presents a summary of our results. It shows the geometric means of node numbers and seconds for all four test sets and different parameter settings.

The first row corresponds to the settings which yield the results of Tables 1–4. Here, conflict analysis is only applied on propagation conflicts (column 'prop') and infeasible LPs (column 'inf'). Conflict constraints are not forced to contain only binary variables (column 'bin'), and they are not added as cutting planes to the LP relaxations (column 'cuts').

For the second line, we activated the separation of pure binary conflict constraints. One can see that this results in slight performance reductions, even in the number of nodes. This is a bit surprising since one would expect that tighter LP relaxations lead to smaller branching trees.

In the third line we used the other strategy to treat non-binary conflicts presented in Section 3.4. In this case, the cut selection rule in the conflict graph analysis is modified such that only binary variables remain in the final conflict set. Naturally, the algorithm is identical to the settings in the first line on the pure binary programming instances. On the MIP and ALU instances, the different treatment of non-binary conflicts does not make a large difference. However, the 'Enlight' instances behave worse with these settings.

For the fourth line, we produced only conflict constraints with binary variables and added them as cutting planes to the LP relaxations of the subproblems. In comparison to the second line, there is now a larger potential to find violated conflict constraints that can be added to the LP, since now all conflict constraints are clauses that can be

linearized. Indeed, these settings give a small reduction in the number of nodes for the general MIP instances, but the performance on the 'Enlight' instances is still worse than that of the settings in the first line.

In the next four lines, we report on experiments with also analyzing bound-exceeding LPs to produce conflict constraints (column 'bdex'). Note that LPs can exceed the primal bound even for infeasible instances, because a trivial primal bound can always be derived if the bounds of the variables are finite. With the exception of the ALU test set, the results degraded in general. Column 'opt' denotes the activation of the optional LP resolving (Step 5 of Algorithm 3.6). As expected, this procedure reduces the number of branching nodes on the BP and MIP test sets. However, it turns out to be too expensive in terms of solving time. Analyzing even the infeasible and bound-exceeding LPs of the strong branching calls (column 'sb') consumes additional time and does not even reduce the number of nodes. The only exception are the ALU instances where the best results could be achieved by analyzing all infeasible and bound-exceeding LPs and solving them to optimality inside the loop of Algorithm 3.6. This shows that the effort spent on conflict analysis can easily be adjusted to further improve the performance on specific problem classes.

We conclude from our computational experiments that conflict analysis is a useful tool for mixed integer programming. Although it has no significant impact on the majority of the feasible instances, it greatly improves the performance on some specific problem classes and instances, especially infeasible ones. With the proposed settings, which apply conflict analysis only moderately, this reduction in branching nodes and solving time can be achieved without major drawbacks on other instances. A more aggressive use of conflict analysis yields even larger improvements on infeasible instances. However, a more appropriate way of dealing with huge amounts of conflict constraints and a better identification of irrelevant constraints would then be needed to avoid overheads on the majority of the instances. Ideas on this topic can be found, for instance, in Bayardo and Schrag [6] or Goldberg and Novikov [16].

References

[1] T. Achterberg, SCIP — a framework to integrate constraint and mixed integer programming, Technical Report 04-19, Zuse Institute Berlin. http://www.zib.de/Publications/abstracts/ZR-04-19/, 2004.

[2] T. Achterberg, T. Koch, A. Martin, The mixed integer programming library: MIPLIB 2003. http://miplib.zib.de.

[3] T. Achterberg, T. Koch, A. Martin, MIPLIB 2003, Operations Research Letters 34 (4) (2006) 1–12.

[4] E. Amaldi, M.E. Pfetsch, L.E. Trotter Jr., On the maximum feasible subsystem problem, IISs, and IIS-hypergraphs, Mathematical Programming 95 (3) (2003) 533–554.

[5] A. Atamtürk, G.L. Nemhauser, M.W. Savelsbergh, Conflict graphs in integer programming, European Journal of Operations Research 121 (2000) 40–55.

[6] R.J. Bayardo, R.C. Schrag, Using CSP look-back techniques to solve real-world SAT instances, in: Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI'97, Providence, Rhode Island, 1997, pp. 203–208.

[7] A. Biere, W. Kunz, SAT and ATPG: Boolean engines for formal hardware verification, in: ACM/IEEE Intl. Conf. on Computer-Aided Design, ICCAD, San Jose, November 2002.

[8] V. Bowman, G.L. Nemhauser, A finiteness proof for the modified Dantzig cuts in integer programming, Naval Research Logistics Quarterly 17 (1970) 309–313.

[9] R. Brinkmann, R. Drechsler, RTL-datapath verification using integer linear programming, in: Proceedings of the IEEE VLSI Design Conference, 2002, pp. 741–746.

[10] S.A. Cook, The complexity of theorem proving procedures, in: Proceedings of 3rd Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–158.

[11] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, Mathematical Programming 102 (1) (2005) 71–90.

[12] Dash Optimization. XPRESS-MP. http://www.dashoptimization.com.

[13] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Communications of the ACM 5 (1962) 394–397.

[14] M. Davis, H. Putnam, A computing procedure for quantification theory, Journal of the Association for Computing Machinery 7 (1960) 201–215.

[15] M.L. Ginsberg, Dynamic backtracking, Journal of Artificial Intelligence Research 1 (1993) 25–46.

[16] E. Goldberg, Y. Novikov, Berkmin: A fast and robust SAT solver, in: Design Automation and Test in Europe, DATE, 2002, pp. 142–149.

[17] ILOG. CPLEX. http://www.ilog.com/products/cplex.

[18] Y. Jiang, T. Richards, B. Richards, No-good backmarking with min-conflict repair in constraint satisfaction and optimization, in: Principles and Practice of Constraint Programming, in: Lecture Notes in Computer Science, vol. 874, 1994, pp. 21–39.

[19] I. LINDO Systems, LINGO. http://www.lindo.com.

[20] J.P. Marques-Silva, K.A. Sakallah, GRASP: A search algorithm for propositional satisfiability, IEEE Transactions of Computers 48 (1999) 506–521.

[21] A. Martin, Integer programs with block structure. Habilitations-Schrift, Technische Universität Berlin. http://www.zib.de/Publications/abstracts/SC-99-03/, 1998.

[22] H. Mittelmann, Decision tree for optimization software: Benchmarks for optimization software. http://plato.asu.edu/bench.html, 2003.
[23] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of the Design Automation Conference, July 2001.
[24] M.E. Pfetsch, The maximum feasible subsystem problem and vertex-facet incidences of polyhedra, Ph.D. Thesis, Technische Universität Berlin, 2002.
[25] D. Rubin, R. Graves, Strengthened Dantzig cuts for integer programming, ORSA 20 (1972) 178–182.
[26] T. Sandholm, R. Shields, Nogood learning for mixed integer programming, Technical Report CMU-CS-06-155, Carnegie Mellon University, Computer Science Department, September 2006.
[27] M.W. Savelsbergh, Preprocessing and probing techniques for mixed integer programming problems, ORSA Journal on Computing 6 (1994) 445–454.
[28] R.M. Stallman, G.J. Sussman, Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis, Artificial Intelligence 9 (1977) 135–196.
[29] K. Truemper, Design of Logic-Based Intelligent Systems, John Wiley & Sons, Hoboken, NJ, 2004.
[30] K. Wolter, Implementation of cutting plane separators for mixed integer programs, Master's Thesis, Technische Universität Berlin, 2006.
[31] R. Zabih, D.A. McAllester, A rearrangement search strategy for determining propositional satisfiability, in: Proceedings of the National Conference on Artificial Intelligence, 1988, pp. 155–160.
[32] L. Zhang, C.F. Madigan, M.W. Moskewicz, S. Malik, Efficient conflict driven learning in Boolean satisfiability solver, in: ICCAD, 2001, pp. 279–285.