Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TOBIAS ACHTERBERG

Conflict Analysis in Mixed Integer Programming

URL: http://www.zib.de/projects/integer-optimization/MIP

ZIB-Report 05-19 (August 2005)

Conflict Analysis in Mixed Integer Programming

Tobias Achterberg*

August 3, 2005

Abstract

Conflict analysis for infeasible subproblems is one of the key ingredients in modern SAT solvers to cope with large real-world instances. In contrast, it is common practice for today's mixed integer programming solvers to just 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 cuts. We performed computational experiments which show the potential of our method: On feasible MIP instances, the number of required branching nodes was reduced by 50% in the geometric mean. However, the total solving time increased by 15%. On infeasible MIPs arising in the context of chip verification, the number of nodes was reduced by 90%, thereby reducing the solving time by 60%.

Keywords: mixed integer programming, branch-and-cut, conflict analysis, SAT, infeasible MIP

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 [13], LINDO [14], SIP [16], or XPRESS [9], 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 applied, which was proposed by Davis, Putnam, Longemann, and Loveland [10, 11]. Modern SAT solvers, however, try to learn from infeasible subproblems, an idea due to Marques-Silva and Sakallah [15]. 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 of the current subproblem's infeasibility and exploit this knowledge. In SAT solving, such a reason is a subset of the current variable fixings which already suffices to trigger a chain of logical deductions that ends in a contradiction. That means, the fixing of the variables of

*Konrad-Zuse-Zentrum für Informationstechnik Berlin, achterberg@zib.de

this *conflict set* renders the problem instance infeasible. The *conflict clause* that 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.

In this paper, we propose a generalization of conflict analysis to branch-and-bound based mixed integer programming. 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, e.g., 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 a 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 variables' local bounds. 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 have to generalize the notion of the conflict graph in the presence of non-binary variables. Experimental results in Section 4 demonstrate that conflict analysis can lead to substantial savings in the number of branching nodes to solve a MIP. However, time measurements indicate that further enhancements to the given algorithms are needed in order to produce an overall speed-up on the investigated feasible instances. For infeasible MIPs, conflict analysis can yield major performance improvements in both, the number of branching nodes and the time needed to prove the infeasibility.

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 [15] or Zhang et al. [23]. The Boolean satisfiability problem (SAT) is defined as follows. The Boolean truth values *false* and *true* are identified with the values 0 and 1, respectively, and Boolean formulas are evaluated correspondingly.

Definition 2.1 (SAT) *Let $C = C_1 \wedge \ldots \wedge C_m$ be a logic formula in conjunctive normal form (CNF) on Boolean variables $x_1, \ldots, x_n$. Each clause $C_i = \ell_1^i \vee \ldots \vee \ell_{k_i}^i$ 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 [8]. 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 [6] for an overview of SAT techniques in chip verification and to Truemper [21] for details on logic-based intelligent systems.

Modern SAT solvers like Chaff [18] or BerkMin [12] rely on the following techniques:

- using a branching scheme (the *DPLL-algorithm* of Davis, Putnam, Longemann, and Loveland [10, 11]) to split the problem into smaller subproblems,
- applying *Boolean Constraint Propagation* (BCP) [22] on the subproblems, which is a simple node preprocessing, and
- analyzing infeasible subproblems to produce conflict clauses [15], 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 gets 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.

2.1 Conflict Graph Analysis

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 $A = A_1 \cup \ldots A_D \cup A_\lambda$. Each subset A_d, $d = 1, \ldots, D$, represents one deduction: whenever a clause $C_i = \ell_1^i \vee \ldots \vee \ell_{k_i}^i \vee \ell_r^i$ became a unit clause in BCP with remaining unfixed literal ℓ_r^i, a set of arcs $A_d = \{(\bar{\ell}_1^i, \ell_r^i), \ldots, (\bar{\ell}_{k_i}^i, \ell_r^i)\}$ is created in order to represent the deduction $\bar{\ell}_1^i \wedge \ldots \wedge \bar{\ell}_{k_i}^i \rightarrow \ell_r^i$ that is implied by C_i. The additional set of arcs $A_\lambda = \{(\bar{\ell}_1^\lambda, \lambda), \ldots, (\bar{\ell}_{k_\lambda}^\lambda, \lambda)\}$

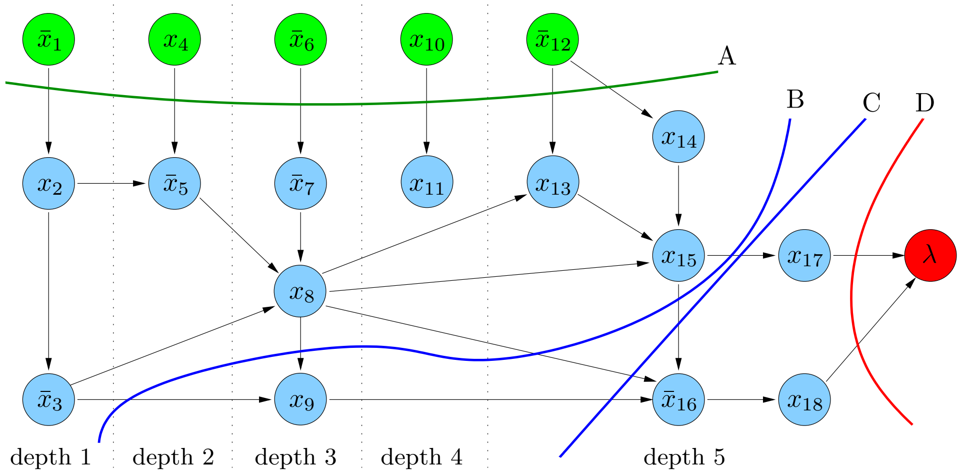

Figure 1. Conflict graph of Example 2.2. The vertices in the top row are branching decisions, the ones below are deductions. Each cut separating the branching vertices and the conflict vertex (λ) yields a *conflict clause*.

represents clause C_λ that detected the conflict (i.e., the clause that got empty in BCP).

Example 2.2 Consider the CNF $C = C_1 \wedge \ldots \wedge C_{18}$ 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 Figure 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 Figure 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.

Figure 1 shows different types of cuts (labeled 'A' to 'D'), leading to different conflict clauses. The cut labeled '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 [23] for a more detailed discussion.

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 Figure 1, there are 5 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) *An* 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 Figure 1, the *FUIP* of the last depth level is x_{15}. The *1-FUIP* cut is the one labeled 'C'. It corresponds to the 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' which corresponds to the clause $C_B = x_3 \vee \bar{x}_8 \vee \bar{x}_{15}$.

2.2 Reconvergence Cuts

In the previous section it was shown that each cut separating the branching vertices from the conflict vertex gives rise to a conflict clause, which contains the literals of the reason side's frontier. By dropping the requirement that the cut must separate the conflict vertex from the branching vertices, we get a different class of cuts which are called *cuts not involving conflicts* (see [23]). These cuts can also be used to derive valid clauses from the conflict graph. In order to apply *non-chronological backtracking*, which is explained in Section 2.3, one has to generate some of these cuts, in particular the *UIP reconvergence cuts* of the last depth level (see below).

Figure 2 gives an example of a cut not involving conflicts. In conflict graph analysis, the conflict vertex λ is substituted by an arbitrary vertex ℓ_u representing a literal. In the example, $\ell_u = x_{15}$ was chosen, which is the first unique implication point of the last depth level.

Each cut separating the branching vertices V_B from the vertex ℓ_u by partitioning the vertices V into $V_r \supseteq V_B$ and $V_c \ni \ell_u$ gives rise to the clause $C_u = (\bigvee_{\ell_i \in V_f} \bar{\ell}_i) \vee \ell_u$.

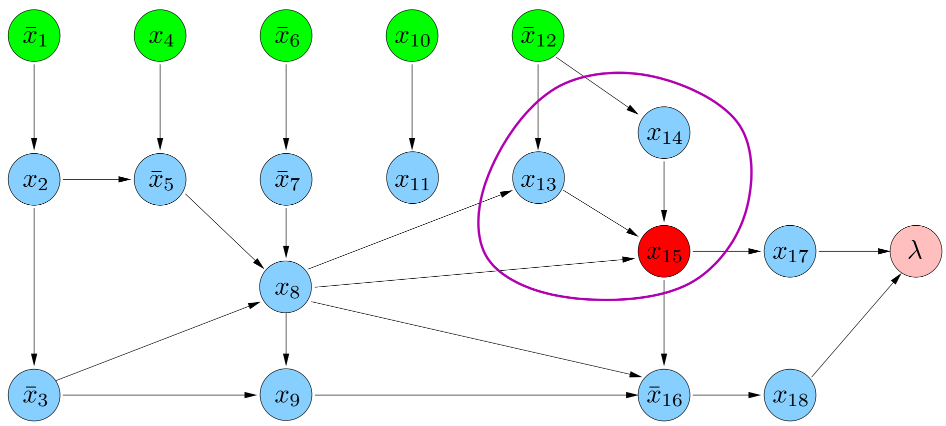

Figure 2. The cut separating the branching vertices (top row) and a deduced vertex (x_{15}) yields the reconvergence clause $\bar{x}_8 \vee x_{12} \vee x_{15}$.

Again, V_f consists of the vertices at the reason side's frontier of the cut. However, such a clause is only useful if $V_c \cup V_f$ contains an ℓ_u*-reconvergence*, i.e., two different paths from a vertex $\ell_i \in V_c \cup V_f$ to ℓ_u. Otherwise, it can be proved that all possible deductions of C_u can already be found by iterated BCP on the current clause database.

The cut shown in Figure 2 is a *UIP reconvergence cut* which connects the two successive *UIP*'s $\bar{x}_{12}$ and x_{15} of depth level 5: by applying all fixings of lower depth levels, $C_u = \bar{x}_8 \vee x_{12} \vee x_{15}$ reduces to the implication $\bar{x}_{12} \rightarrow x_{15}$. Note that BCP can now also deduce $\bar{x}_{15} \rightarrow x_{12}$, which is not possible without using C_u.

2.3 Non-chronological Backtracking

Suppose the conflict analysis procedure produced a clause with only one literal ℓ_u^d fixed at depth level d in which the conflict was detected. All other literals were fixed at depth levels smaller or equal to $d' < d$. If this clause would have been known earlier, the literal ℓ_u^d could already have been fixed to the opposite value in depth d'. Suppose the conflict analysis procedure also produced all *reconvergence clauses* necessary to connect ℓ_u^d to the branching vertex ℓ_b^d of depth d. Then, also the branching variable of depth d could have been fixed to the opposite value in depth d'.

Therefore, after having found such a conflict clause, the search tree's node in depth level d' can be reevaluated to apply the deductions leading to the opposite fixing of ℓ_b^d. Further deductions may lead to another conflict, thus rendering the whole subtree rooted in depth d' infeasible without evaluating its remaining leaves. In [15] it was empirically shown, that this so-called *non-chronological backtracking* can lead to large reductions in the number of evaluated nodes to solve SAT instances.

In our Example 2.2, the conflict analysis engine used in this paper produces the conflict clauses $C_B = x_3 \vee \bar{x}_8 \vee \bar{x}_{15}$ and $C_C = \bar{x}_8 \vee \bar{x}_9 \vee \bar{x}_{15}$. Additionally, the reconvergence clause $C_R = \bar{x}_8 \vee x_{12} \vee x_{15}$ is added to the clause database. Evaluating the node in depth 3 again, $x_{15} = 0$ (using C_C) and $x_{12} = 1$ (using C_R) can be deduced, leading together with $C_{15}, \ldots, C_{18}$ to another conflict (see Figure 3). Therefore, the subtree with $x_1 = 0$, $x_4 = 1$, and $x_6 = 0$ can be pruned

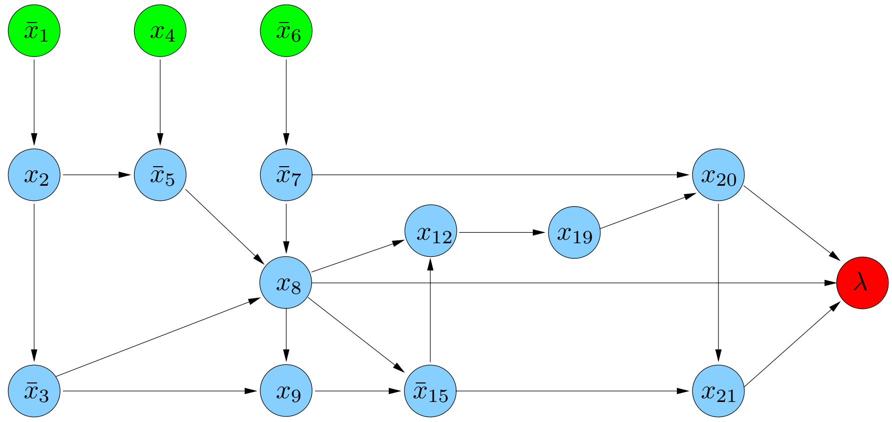

Figure 3. Reevaluation of the node in depth 3 after inserting conflict and reconvergence clauses again leads to a conflict.

without evaluating the intermediate branching decisions (in this case $x_{10} = 0$ and $x_{10} = 1$).

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 [20]).

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 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 a 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 binary 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 or bound exceedance of its LP relaxation. 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 bound exceedance is treated in Section 3.3.

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 non-binary 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. 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 $z_1, \ldots, z_5 \in \mathbb{Z}_{\geq 0}$.

$$2x_1 + 3z_1 + 2z_2 \leq 9 \tag{2}$$
$$+9x_2 - z_1 - 2z_2 \leq 0 \tag{3}$$
$$-3x_2 + 5x_3 - 3x_4 \leq 4 \tag{4}$$
$$-3x_2 + 9x_4 - 2z_3 \leq 6 \tag{5}$$
$$+9x_5 - z_2 + 2z_3 \leq 8 \tag{6}$$
$$-4x_6 - 7x_7 + 2z_3 \leq 3 \tag{7}$$
$$+5x_7 - 2z_2 \leq 2 \tag{8}$$
$$-x_5 + 5x_7 + 4z_2 - 5z_3 \leq 2 \tag{9}$$
$$x_1 - x_2 + x_3 - 2x_5 + x_6 - z_1 - 2z_2 + z_3 - 2z_4 + 4z_5 \leq 1 \tag{10}$$
$$+2x_2 - x_4 + 3x_5 - 2x_6 - z_1 + 5z_2 + z_3 + 2z_4 - 6z_5 \leq 2 \tag{11}$$
$$-2x_1 - 2x_3 + x_4 + x_5 + z_1 + 2z_2 - 2z_3 + 2z_4 - 2z_5 \leq 1 \tag{12}$$

By the basic bound strengthening techniques of Savelsbergh [20], 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

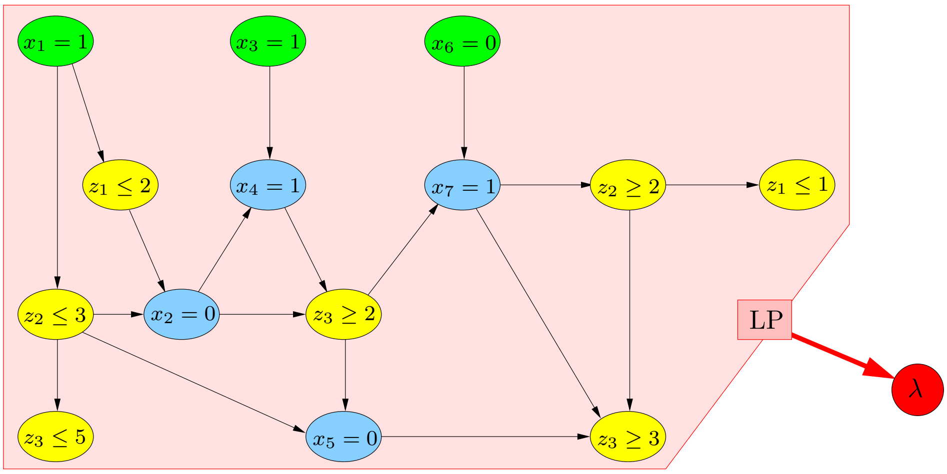

Figure 4. Conflict graph of Example 3.1. After applying the branching decisions $x_1 = 1$, $x_3 = 1$, $x_6 = 0$, and all inferred bound changes, the LP relaxation becomes infeasible. The implications on variables z_4 and z_5 are not included in the figure.

constraint (2) we can deduce $z_1 \leq 2$ and $z_2 \leq 3$ (see Figure 4). 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 Figure 4 with the LP relaxation being infeasible. Note that the non-binary 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 $\mathcal{B}_L = \{B_1, \dots, B_K\}$ with hyperplanes $B_k = L_{j_k}^{\mu_k} := \{x \mid x_{j_k} \geq \mu_k\}$ or $B_k = U_{j_k}^{\mu_k} := \{x \mid x_{j_k} \leq \mu_k\}$, $1 \leq j_k \leq n$, $l_{j_k} \leq \mu_k \leq u_{j_k}$, $k = 1, \dots, K$. The set $\mathcal{B}_L$ corresponds to the additional bounds imposed on the variables in the local subproblem. Thus, the subproblem is defined as

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

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,

[1]In a branch-and-cut framework, we have to either remove local cuts from the LP or mark the resulting conflict clause 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.

even if bound changes on non-binary 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 conflict clause. 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 $F = \{x \mid Ax \le b\}$. Let $\mathcal{B}_L = \{B_1, \dots, B_K\}$ be additional bounds with $B_k = \{x \mid x_{j_k} \ge \mu_k\}$ or $B_k = \{x \mid x_{j_k} \le \mu_k\}$, $1 \le j_k \le n$, for all $k = 1, \dots, K$, 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*

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

Lemma 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 $\{x \mid Ax \le b\} = \emptyset$, the task is to find a minimal cardinality subset of the rows of $Ax \le b$ 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 $\mathcal{B}_L = \{B_1, \dots, B_m\}$ with $B_i = \{(x, s) \mid s_i \ge 0\}$ for $i = 1, \dots, m$. 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'. $\square$

There are various heuristics for *minimal cardinality IIS* (see [19]). 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 $\tilde{u} \le u$. 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

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

is infeasible. Then its dual

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

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

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

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

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.

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 non-zero lower bounds the reduced costs r may also be negative. In this case, we can split up the reduced cost values into $r = r^u - r^l$. 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) 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

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

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 Figure 4 can be replaced by three arcs as shown in Figure 5.

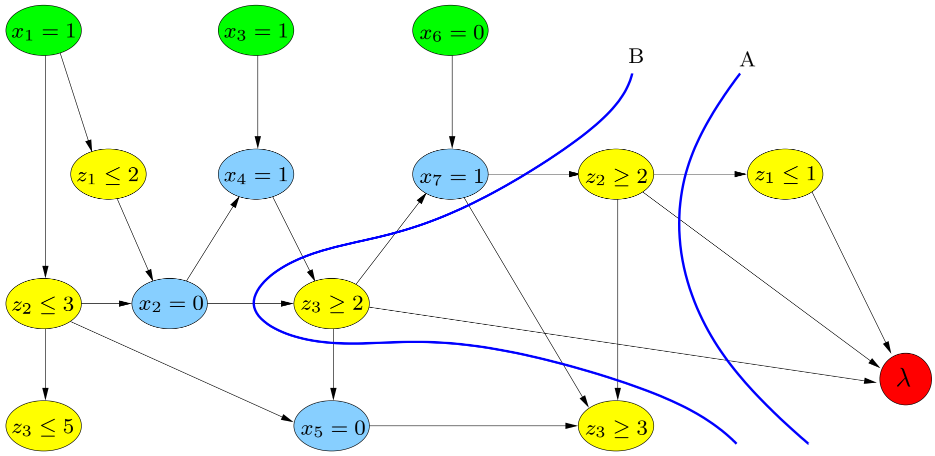

Figure 5. Conflict graph of Example 3.1 after the infeasible LP was analyzed. Cut 'A' is the *1-FUIP* cut. Cut 'B' was constructed by moving the non-binary variables of the conflict set of cut 'A' to the conflict side.

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 exceedance. 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

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

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

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

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 exceedance 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 non-binary 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

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

If at least one of the conflict variables x_j is continuous, the linearization of the conflict constraint would remain a strict inequality, which cannot be handled by an LP solver. If all conflict variables are integers, the conflict constraint has the form

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

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

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

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 hypotheses by computational studies.

We therefore refrain from creating conflict constraints with non-binary variables. Instead, we modified the cut selection rules in the conflict graph analysis in order to always choose the cut 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. However, this is not possible if the bound change was due to a branching decision, because branching vertices must be located on the reason 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.

Example 3.7 (continued) Figure 5 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. The *1-FUIP* cut selection scheme leads to the cut labeled '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 represented 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

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

4 Computational Results

In this section we examine the computational effectiveness of conflict analysis on several MIP instances. All calculations were performed on a 3.2 GHz Pentium-4EE workstation with 2 GB RAM. In all runs we used a time limit of 3600 seconds.

The conflict analysis techniques described in Sections 2 and 3 were implemented into the constraint and mixed integer programming framework SCIP version 0.79a (see [1]). Conflicts detected in constraint propagation and infeasible (or bound exceeding) LP relaxations of local nodes are analyzed to produce conflict clauses. Additionally, infeasibilities of LP relaxations that are detected in the strong branching calls of the reliability branching rule [3] are examined with the same algorithms. If the initial conflict set produced by the LP analysis already consists of only binary variables, the corresponding conflict clause is added to the constraint database. In every case, the conflict graph analysis produces additional conflict clauses with the *1-FUIP* and *All-FUIP* schemes [23] and reconvergence clauses to connect the last depth level's first UIP and branching vertex.

To solve the LP relaxations, the dual simplex algorithm of CPLEX is applied. It immediately stops after a dual feasible solution that exceeds the primal bound has been found. This solution, which is usually not optimal, can result in a very small initial bound exceedance measure (Algorithm 3.6 Step 2). In order to get useful initial conflict sets, we continue solving the LP to optimality without objective limit and use the optimal solution for LP infeasibility analysis. The optional Step 5 of Algorithm 3.6 is not applied, because it usually reduces the size of the initial conflict set by only a few additional bound changes and therefore does not justify its additional costs.

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 clauses are added on demand to the LP relaxations at local nodes and take part in the propagation process inside the local nodes' solving loop.

Because the recorded conflict clauses 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 clauses. Clauses are deleted, if they did not help for a couple of consecutive iterations to tighten the LP relaxation

[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.

during separation or propagation. Longer clauses are discarded earlier than clauses with fewer literals.

4.1 Test Set

Our first test set consists of instances from Miplib 2003 [2] and instances collected by Mittelmann [17]. We selected all problems where CPlex 9.03 needs at least 1000 branching nodes and 30 seconds and at most one hour CPU time for solving.[4]

As a second test set, we use IP models of chip design verification problems. The data are for a very simple arithmetic logical unit (ALU) of different register widths ranging from 4 to 10 bits and with different properties to be checked (see, e.g., [7]). The instances, the ALU model, and the property definitions can be found in the *contributed instances* section of Miplib 2003.

The so-called *property checking problem* addressed here is to prove the validity of a certain property for a given chip design. This problem can be modeled as 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. All of the ALU instances investigated here correspond to valid properties, i.e., the IP instances are infeasible.

Note that the 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 solvers' feasibility and integrality tolerances for this second test set to 10^{-9}.

4.2 Description of the Results

Table 1 gives some statistics on the problems in the first test set and shows the results with the default settings for MIP solving and those with activated conflict analysis. Columns 'Rows', 'Cols', 'Bins', and 'Ints' show the number of rows, columns, binary, and general integer variables of the problem instances, respectively. Columns 'Nodes' and 'Time' show the number of branching nodes and the total time in seconds needed to solve the instances. Values marked with a '>' denote that the instance could not be solved within the time limit. 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.

We can observe that conflict analysis yields a reduction of branching nodes of about 50% in terms of the geometric mean. Nevertheless, this does not reduce the total running time. In fact, the geometric mean of the running time increases by about 15%. Similar observations can be made for the reduced test set consisting of only those instances that could be solved by both settings within the time limit. A closer look at the single instances reveals that activating conflict analysis reduces the number of nodes on 21 instances and the solving time on 12 instances. The default MIP setting needs fewer nodes on 5 instances and is faster on 16 instances.

Table 2 gives a more detailed timing analysis of the two settings. The *basic time* (BTime) is the time spent for presolving, node processing (including LP solving), and primal heuristics. In particular, this also includes the additional overhead of processing the conflict constraints in domain propagation and cut separation. The

[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.

Name	Rows	Cols	Bins	Ints	CPLEX 9.03 Nodes	CPLEX 9.03 Time	SCIP 0.79a (default) Nodes	SCIP 0.79a (default) Time	SCIP 0.79a (conflict) Nodes	SCIP 0.79a (conflict) Time
10teams	230	2025	1800		2 771	63.5	596	73.8	313	77.0
aflow30a	479	842	421		37 116	86.9	14 269	77.6	6 322	66.3
mas74	13	151	150		5 311 878	1306.6	3 378 402	902.1	> 810 784	>3600.0
mas76	12	151	150		375 729	56.8	566 581	152.3	> 437 272	>3600.0
misc07	212	260	259		72 420	60.2	35 210	47.8	41 138	87.1
mzzv11	9499	10240	9989	251	3 820	443.4	4 668	1285.2	2 847	1269.5
pk1	45	86	55		365 710	86.8	230 529	86.3	352 533	644.0
qiu	1192	840	48		11 431	107.5	10 023	166.8	9 275	207.5
rout	291	556	300	15	843 504	1845.2	27 476	52.2	8 988	38.1
acc-6	3047	1335	1335		1 084	656.5	> 8 959	>3600.1	5 398	2582.4
bc1	1913	1751	252		9 939	270.4	62 624	1514.9	564	125.5
bienst1	576	505	28		12 292	113.7	9 064	61.3	5 842	52.2
bienst2	576	505	35		161 843	2381.4	89 989	868.6	77 768	980.8
mkc1	3411	5325	3087		14 236	48.9	> 449 959	>3600.0	> 294 204	>3600.0
neos2	1103	2101	1040		40 193	83.5	129 695	314.9	10 736	125.0
neos3	1442	2747	1360		835 129	2630.7	> 903 445	>3600.0	103 956	1310.7
neos6	1036	8786	8340		5 328	232.6	1 033	191.9	2 224	293.6
neos7	1994	1556	434	20	113 861	423.3	40 056	390.3	36 974	446.3
neos11	2706	1220	900		3 058	179.7	11 247	1018.4	9 832	1537.0
neos21	1085	614	613		7 531	85.1	1 611	36.3	1 579	62.0
neos22	5208	3240	454		13 520	45.2	30 598	282.0	24 615	318.3
prod1	208	250	149		97 507	68.2	67 311	45.2	43 486	64.9
ran10x26	296	520	260		20 176	42.0	50 525	116.0	21 451	85.8
ran12x21	285	504	252		81 525	159.2	129 096	282.2	47 764	218.8
ran13x13	195	338	169		62 936	63.6	103 302	118.2	21 459	60.5
seymour1	4944	1372	451		10 459	1056.2	4 378	880.9	4 206	1103.5
swath1	884	6805	2306		15 860	42.2	378	31.6	387	45.9
swath2	884	6805	2406		150 740	383.2	12 418	120.3	12 450	187.2
swath3	884	6805	2706		728 982	2180.0	54 858	401.0	44 382	545.9
Total (29)					9 410 578	15202.4	6 428 300	20318.4	2 438 749	23335.9
Geom. Mean					41 268	202.0	29 633	257.2	15 441	297.8
Reduced (24)					2 872 522	10502.7	1 120 954	8463.9	787 135	8642.9
Geom. Mean					32 994	178.8	17 509	179.4	9 720	187.4

Table 1. Comparison of runs with CPLEX 9.03, SCIP with default MIP settings, and SCIP with conflict analysis. Results marked with '>' were not solved to optimality within the time limit. The totals and geometric means are given for the full test set as well as the reduced test set excluding those instances (printed in bold face) not solved to optimality by one or both of the two SCIP settings.

switching time (STime) denotes the time used for setting up the subproblems to be processed. This includes updating the set of local constraints and the variables' local bounds. The *conflict analysis time* (CTime) summarizes the extraction of starting conflicts from infeasible and bound exceeding LPs as well as the conflict graph analysis itself. The largest fraction of this time is usually spent to continue solving bound exceeding LPs to optimality in order to get useful initial conflict sets. The additional columns 'Confs' and '∅Lits' show the number of conflict clauses generated during each run and the average number of literals per conflict clause, respectively.

One can see that the reduction of 50% in the number of branching nodes obtained with conflict analysis lead to a saving of 10% in the basic solution time in terms of the geometric mean. Unfortunately, this does not result in an overall performance improvement, because the additional costs for the conflict analysis itself (in particular to continue solving the LPs to optimality) outweigh the benefit of the node reduction. In addition, due to the additional constraints generated in the local nodes, the switching time increases. This increase in switching time has the most

	SCIP 0.79a (default MIP)				SCIP 0.79a (conflict analysis)						
Name	Nodes	Time	BTime	STime	Nodes	Time	BTime	STime	CTime	Confs	∅Lits
10teams	596	73.8	73.8	0.1	313	77.0	76.3	0.1	0.5	193	523.7
aflow30a	14269	77.6	76.1	1.5	6322	66.3	48.3	1.7	16.3	5757	41.3
mas74	3378402	902.1	773.7	128.4	>810784	>3600.0	2650.4	891.0	58.6	257919	19.0
mas76	566581	152.3	130.0	22.2	>437272	>3600.0	2481.4	1070.9	47.7	236310	20.2
misc07	35210	47.8	46.0	1.8	41138	87.1	66.1	4.9	16.1	38062	44.4
mzzv11	4668	1285.2	1278.1	7.1	2847	1269.5	1136.0	8.8	124.7	1858	26.4
pk1	230529	86.3	77.6	8.8	352533	644.0	454.9	158.8	30.2	160614	15.3
qiu	10023	166.8	165.3	1.6	9275	207.5	161.9	3.2	42.4	6245	16.3
rout	27476	52.2	49.8	2.4	8988	38.1	27.1	1.5	9.5	5931	12.6
acc-6	>8959	>3600.1	3598.5	1.6	5398	2582.4	2430.1	1.0	151.3	5197	138.3
bc1	62624	1514.9	1270.7	244.2	564	125.5	79.7	1.2	44.6	103	54.7
bienst1	9064	61.3	59.3	2.0	5842	52.2	46.5	1.0	4.7	3728	12.7
bienst2	89989	868.6	844.8	23.8	77768	980.8	787.7	86.6	106.4	58335	18.1
mkc1	>449959	>3600.0	3516.9	83.2	>294204	>3600.0	2242.1	88.9	1269.1	144076	76.8
neos2	129695	314.9	308.8	6.2	10736	125.0	96.6	2.4	26.0	11610	49.7
neos3	>903445	>3600.0	3546.8	53.2	103956	1310.7	935.4	45.2	330.1	91179	83.3
neos6	1033	191.9	191.0	0.9	2224	293.6	253.9	2.8	37.0	911	379.5
neos7	40056	390.3	375.9	14.4	36974	446.3	368.1	12.7	65.4	14335	4.6
neos11	11247	1018.4	1015.4	3.0	9832	1537.0	974.3	3.7	558.9	8358	17.9
neos21	1611	36.3	36.2	0.1	1579	62.0	44.2	0.2	17.6	3098	93.5
neos22	30598	282.0	273.7	8.3	24615	318.3	238.8	8.3	71.1	14380	14.4
prod1	67311	45.2	41.6	3.6	43486	64.9	53.5	8.5	3.0	24775	18.2
ran10x26	50525	116.0	112.8	3.1	21451	85.8	64.3	4.0	17.4	12403	19.3
ran12x21	129096	282.2	273.4	8.8	47764	218.8	163.3	15.0	40.5	25323	20.3
ran13x13	103302	118.2	115.1	3.1	21459	60.5	45.4	3.0	12.1	12440	16.2
seymour1	4378	880.9	879.9	0.9	4206	1103.5	906.7	1.2	195.7	3142	15.9
swath1	378	31.6	31.5	0.2	387	45.9	32.9	0.2	12.8	1021	149.4
swath2	12418	120.3	116.6	3.7	12450	187.2	120.9	3.8	62.5	7211	339.0
swath3	54858	401.0	380.6	20.4	44382	545.9	354.0	19.6	172.3	15272	211.4
Total (29)	6428300	20318.4	19659.6	658.8	2438749	23335.9	17341.0	2450.4	3544.5		
Geom. Mean	29633	257.2	247.0	5.7	15441	297.8	225.0	7.2	40.7		
Reduced (24)	1120954	8463.9	8093.7	370.2	787135	8642.9	6601.7	353.4	1687.8		
Geom. Mean	17509	179.4	173.4	4.1	9720	187.4	142.5	4.4	30.0		

Table 2. More detailed timing analysis of runs with default MIP settings and conflict analysis.

significant impact for the small instances `mas74`, `mas76`, 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 10 bits, denoted 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. The reduced test set excludes the instances for properties 2 and 6 (printed in bold face), which were solved by SCIP's preprocessing algorithm.

On this test set, conflict analysis clearly outperforms the default MIP settings in both, branching nodes and solving time. Most notably, property 1 can be solved with conflict analysis quite easily even for 10-bit input registers, while SCIP with default settings needs over a million branching nodes to prove the property, i.e., to show the infeasibility of the instance.

In contrast to the conflict clauses generated for the feasible MIP instances in Table 2, the clauses found for the ALU instances contain only very few literals. 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 clauses and the performance of conflict analysis for the instances of Table 2 do not seem to be correlated.

While the results on infeasible IPs are very promising, it remains to be seen in

	Size		CPLEX 9.03		SCIP 0.79a (default)		SCIP 0.79a (conflict)			
Name	Rows	Cols	Nodes	Time	Nodes	Time	Nodes	Time	Confs	∅Lits
alu4_1	609	283	15 427	9.3	931	7.2	81	5.6	207	5.5
alu4_2	609	285	105	0.2	0	0.0	0	0.0	0	0.0
alu4_6	623	290	5 877	1.9	0	0.1	0	0.1	0	0.0
alu4_7	620	287	753	0.6	8 033	7.8	2 652	5.2	656	2.2
alu4_8	641	299	3 938	1.8	9 409	9.1	9 552	11.7	1 603	2.2
alu5_1	624	293	246 385	111.2	239	4.4	34	3.5	106	3.8
alu5_2	624	295	1 180	0.5	0	0.0	0	0.0	0	0.0
alu5_6	638	300	30 528	8.0	0	0.1	0	0.1	0	0.0
alu5_7	641	299	4 907	2.2	62 221	39.0	34 212	32.3	6 081	1.9
alu5_8	664	313	16 008	5.9	24 677	20.3	10 994	14.8	1 293	2.0
alu6_1	627	295	356 337	170.4	37 153	48.4	138	6.0	288	6.0
alu6_2	627	297	4 938	1.7	0	0.0	0	0.0	0	0.0
alu6_6	641	302	118 734	32.8	0	0.1	0	0.1	0	0.0
alu6_7	644	301	14 651	7.5	280 971	160.4	143 089	128.8	22 479	1.7
alu6_8	667	315	347 037	112.7	76 557	60.4	40 018	47.8	5 331	1.8
alu7_1	636	301	322 706	150.1	67 555	89.0	48	3.4	128	3.8
alu7_2	636	303	18 601	7.7	0	0.0	0	0.0	0	0.0
alu7_6	650	308	218 851	66.7	0	0.1	0	0.1	0	0.0
alu7_7	656	308	197 121	99.4	646 307	375.4	223 494	199.1	50 951	2.6
alu7_8	680	323	727 242	281.0	646 777	493.4	183 340	231.1	18 144	1.8
alu8_1	645	307	3 153 983	1844.7	24 397	36.4	31	3.2	138	4.0
alu8_2	645	309	118 720	54.5	0	0.0	0	0.0	0	0.0
alu8_6	659	314	4 972 884	1403.9	0	0.1	0	0.1	0	0.0
alu8_7	668	315	1 459 674	485.7	2 086 829	1304.1	366 487	360.5	70 800	2.3
alu8_8	693	331	>8 505 766	>3600.0	2 316 613	1944.3	1 043 928	1348.5	89 898	2.1
alu9_1	660	317	745 348	453.1	507	6.2	65	4.9	176	4.2
alu9_2	660	319	2 436 102	868.6	0	0.0	0	0.0	0	0.0
alu9_6	674	324	10 886 390	3257.6	0	0.1	0	0.1	0	0.0
alu9_7	689	327	>6 847 385	>3600.0	>5 209 565	>3600.0	1 800 438	1857.8	315 433	2.2
alu9_8	716	345	>9 320 110	>3600.0	3 260 385	3558.3	1 847 070	3200.7	220 764	1.6
alu10_1	669	323	>5 679 319	>3600.0	1 141 285	1454.8	177	6.9	316	6.2
alu10_2	669	325	>7 951 888	>3600.0	0	0.0	0	0.1	0	0.0
alu10_6	683	330	>13 836 405	>3600.0	0	0.1	0	0.1	0	0.0
alu10_7	701	334	>8 340 048	>3600.0	>3 926 913	>3600.0	>3 085 949	>3600.0	569 980	2.7
alu10_8	729	353	>7 391 679	>3600.0	>2 807 538	>3600.0	>2 525 710	>3600.0	87 204	1.6
Total (35)			94 297 027	38239.7	22 634 862	20419.4	11 317 507	14672.6		
Geom. Mean			215 146	101.4	1 059	19.7	289	11.5		
Reduced (21)			53 695 824	25335.8	22 634 862	20418.7	11 317 507	14671.8		
Geom. Mean			322 700	157.3	110 033	143.1	12 671	58.6		

Table 3. Results for the ALU test set. Results marked with '>' were not solved to optimality within the time limit. For the calculations of the geometric mean, values smaller than 1.0 were treated as being equal to 1.0.

the future, whether more clever and faster heuristics to derive conflict clauses out of infeasible LPs as well as a more appropriate way of dealing with huge amounts of conflict clauses can also lead to an overall improvement in solving feasible mixed integer programs. The latter issue is also addressed in the SAT community, e.g., in Goldberg and Novikov [12]. Incorporating their ideas of clause database management should further improve the performance of conflict analysis for MIP solving.

References

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

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

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

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

[5] A. Atamtürk, G. Nemhauser, and M. Savelsbergh. Conflict graphs in integer programming. *European Journal of Operations Research*, 121:40–55, 2000.

[6] A. Biere and 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.

[7] R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *Proceedings of the IEEE VLSI Design Conference*, pages 741–746, 2002.

[8] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

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

[10] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[11] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[12] E. Goldberg and Y. Novikov. Berkmin: A fast and robust SAT solver. In *Design Automation and Test in Europe (DATE)*, pages 142–149, 2002.

[13] ILOG CPLEX. Reference Manual, 2004. `http://www.ilog.com/products/cplex`.

[14] LINDO. API Users Manual, 2003. `http://www.lindo.com`.

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

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

[17] H. Mittelmann. Decision tree for optimization software: Benchmarks for optimization software, 2003. `http://plato.asu.edu/bench.html`.

[18] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, July 2001.

[19] M.E. Pfetsch. *The Maximum Feasible Subsystem Problem and Vertex-Facet Incidences of Polyhedra*. PhD thesis, Technische Universität Berlin, 2002.

[20] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[21] K. Truemper. *Design of Logic-based Intelligent Systems*. John Wiley & Sons, Hoboken, New Jersey, 2004.

[22] R. Zabih and D.A. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–160, 1988.

[23] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.