



JAKOB WITZIG¹

AMBROS GLEIXNER²

Conflict-Driven Heuristics for Mixed Integer Programming

¹  0000-0003-2698-0767

²  0000-0003-0391-5903

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

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

Conflict-Driven Heuristics for Mixed Integer Programming

Jakob Witzig and Ambros Gleixner

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
{witzig,gleixner}@zib.de

February 6, 2019

Abstract

Two essential ingredients of modern mixed-integer programming (MIP) solvers are diving heuristics that simulate a partial depth-first search in a branch-and-bound search tree and conflict analysis of infeasible subproblems to learn valid constraints. So far, these techniques have mostly been studied independently: primal heuristics under the aspect of finding high-quality feasible solutions early during the solving process and conflict analysis for fathoming nodes of the search tree and improving the dual bound. Here, we combine both concepts in two different ways. First, we develop a diving heuristic that targets the generation of valid conflict constraints from the Farkas dual. We show that in the primal this is equivalent to the optimistic strategy of diving towards the best bound with respect to the objective function. Secondly, we use information derived from conflict analysis to enhance the search of a diving heuristic akin to classical coefficient diving. The computational performance of both methods is evaluated using an implementation in the source-open MIP solver SCIP. Experiments are carried out on publicly available test sets including MIPLIB 2010 and COR@L.

1 Introduction

The most commonly used method to solve *mixed-integer programs* (MIPs) is the *linear programming*-based (LP-based) branch-and-bound algorithm (Dakin 1965, Land and Doig 1960). In modern MIP solvers, this procedure is accelerated by various extensions (see e.g., Bixby et al. 2000, Laundry et al. 2009). Two examples of those extensions are primal heuristics (see e.g., Fischetti and Lodi 2010, Lodi 2013, Berthold 2014a) and conflict analysis (see e.g., Davey et al. 2002, Sandholm and Shields 2006, Achterberg 2007a, Witzig et al. 2017). A primal heuristic is an incomplete method without any guarantee of success, which is used to find feasible and improving solutions. Computational studies indicate that within a

MIP solver, disabling all primal heuristics would lead to a deterioration of solving time by approximately 11% – 32% (Berthold 2014a) and 5% – 15% (Achterberg and Wunderling 2013). Conflict analysis denotes a collection of techniques to learn from infeasible subproblems encountered during the MIP solve. The outcome of conflict analysis is a set of so-called conflict constraints that are used in the remainder of the solving process, e.g., for propagation. As a consequence, the proof of global optimality can be accelerated, mostly by reducing the number of subproblems that need to be explored, according to the study of Achterberg and Wunderling (2013) by as much as 28% on affected instances.

In this paper, we propose two complementary ways of combining both concepts. Firstly, we develop a primal diving heuristic that explicitly aims to generate conflict constraints. As we show by elementary calculations, this amounts to an optimistic fixing of variables to their best bound with respect to the objective function. Previous approaches that are targeted to gain additional conflict information starting from a feasible subproblem are based on, for example, an involved random sampling approach (Dickerson and Sandholm 2013) or use a black-box solver to perform a hybrid constraint programming and MIP search (Berthold et al. 2010, 2018). In contrast to that, our approach constitutes a more direct method.

Secondly, we use the information obtained by conflict analysis in order to guide the LP relaxation towards feasibility. To this end, we apply the concept of variable locks to conflict constraints and show that this type of locks is richer on information and yield a more dynamic criterion compared to variable locks as known from the literature (Achterberg 2007b). We use this to develop a new diving heuristic that harnesses variable locks implied by conflict constraints. Our experiments indicate that this heuristic outperforms the well-known coefficient diving heuristic (Berthold 2008).

To show how a MIP solver can benefit from the techniques presented in this paper as supplementary features, we carry out a detailed computational study for which both heuristics were implemented within the academic MIP solver SCIP 6.0 (Gleixner et al. 2018). The heuristics presented in this paper are – to the best of our knowledge – the first LP-based heuristics for MIP which explicitly produce and exploit conflict constraints.

This paper is organized as follows. In Section 2 we give a brief overview of all the background we need in the remainder of this paper: LP-based branch-and-bound, diving heuristics, and conflict analysis. In Section 3 we discuss how a diving heuristic can be used to generate additional conflict information explicitly. Afterward, we present a modification and extension of the well-known diving heuristic coefficient diving by using conflict information in Section 4. Finally, an intense computational study of the individual impact of both presented approaches is presented in Section 5. In Section 6 we conclude.

2 Background

We consider MIPs of the form

$$\min\{c^\top x \mid Ax \geq b, \ell \leq x \leq u, x_j \in \mathbb{Z} \forall j \in \mathcal{I}\}, \quad (1)$$

with objective coefficient vector $c \in \mathbb{R}^n$, constraint coefficient matrix $A \in \mathbb{R}^{m \times n}$, constraint right-hand side $b \in \mathbb{R}^m$, and variable bounds $\ell, u \in \overline{\mathbb{R}}^n$, where $\overline{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$. Moreover, let $\mathcal{I} \subseteq \mathcal{N} := \{1, \dots, n\}$ be the index set of integer variables.

LP-based Branch-and-bound. Branch-and-bound (Dakin 1965, Land and Doig 1960) is a divide-and-conquer method which splits the search space sequentially into smaller subproblems that are ideally easier to solve. For each subproblem a lower bound is computed. To this end, the integrality requirements are omitted and the LP relaxation

$$\min\{c^\top x \mid Ax \geq b, \ell \leq x \leq u, x \in \mathbb{R}^n\} \quad (2)$$

is solved. On the other hand, an upper bound on the global problem is given by the objective value of the incumbent solution, i.e., the best solution found so far, if available.

During the branch-and-bound procedure subproblems are regularly fathomed, either due to bounding or infeasibility. In the first case, subproblems whose lower bound exceeds the global upper bound are disregarded because they cannot contain an improving solution. Therefore, it is evident that branch-and-bound algorithms benefit directly from finding good solutions as early as possible. These solutions either originate directly from the LP relaxation when all variables fulfill the integrality conditions in the LP solution, or are constructed by so-called primal heuristics (Fischetti and Lodi 2010, Lodi 2013, Berthold 2014a). In the second case, the infeasibility of a subproblem is either proven by contradicting variable bound changes or by an infeasible LP relaxation. If a node is fathomed due to infeasibility modern MIP solvers use conflict analysis (Davey et al. 2002, Achterberg 2007b, Witzig et al. 2017) to “learn” from those subproblems. Note that every subproblem fathomed due to bounding can be interpreted as an infeasible subproblem after adding a cutoff constraint on the objective function that restricts the feasible region to improving solutions.

Diving Heuristics. A special type of primal heuristics are so-called *diving heuristics* such as fractional-diving and pseudo-cost diving (Berthold 2008). The principle idea of diving heuristics comes from the branch-and-bound procedure itself. Starting from a feasible but fractional LP solution, diving heuristics alternate between fixing some integer variables to a rounded value based on a fractional LP solution and reoptimizing the LP relaxation. This procedure can be viewed as a partial tree search along one path from the current subproblem to a leaf. Diving heuristics use a special branching rule that usually tends towards feasibility. By contrast, branching rules for complete tree search such as

Algorithm 1: GENERICDIVINGPROCEDURE

Input : LP solution x^{LP} , rounding function ϕ , score function ψ
Output : Solution candidate \hat{x} or NULL

- 1 $\hat{x} \leftarrow \text{NULL}, \tilde{x} \leftarrow x^{LP}$
- 2 $\mathcal{D} \leftarrow \{j \in \mathcal{I} \mid \tilde{x}_j \notin \mathbb{Z}\}$ // diving candidates
- 3 **while** $\hat{x} == \text{NULL}$ and $\mathcal{D} \neq \emptyset$ **do**
- 4 **forall** $i \in \mathcal{D}$ **do**
 1. Determine rounding direction: $d_j \leftarrow \phi(j)$
 2. Calculate variable score: $s_j \leftarrow \psi(j)$
- 5 Select candidate x_j with maximal score s_j and current local bounds ℓ_j and u_j
- 6 Update $\mathcal{D} \leftarrow \mathcal{D} \setminus j$
- 7 **if** $d_j == \text{up}$ **then** $\ell_j \leftarrow \lceil \tilde{x}_j \rceil$ **else** $u_j \leftarrow \lfloor \tilde{x}_j \rfloor$
- 8 (optional) Propagate this bound change
- 9 Update \mathcal{D} if propagation fixed some $j \in \mathcal{D}$
- 10 **if** *Infeasibility detected* **then**
- 11 Analyze infeasibility, add conflict constraints, perform 1-level backtrack
- 12 **If** $\mathcal{D} = \emptyset$ **goto** 20 or 5 otherwise
- 13 (optional) Solve local LP relaxation
- 14 **if** *Infeasibility detected* **then**
- 15 Analyze infeasibility, add conflict constraints, perform 1-level backtrack
- 16 **If** $\mathcal{D} = \emptyset$ **goto** 20 or 5 otherwise
- 17 Update \tilde{x} and \mathcal{D} if LP was solved
- 18 **if** $\tilde{x}_j \in \mathbb{Z}$ for all $j \in \mathcal{I}$ or $\mathcal{D} == \emptyset$ **then**
- 19 $\hat{x} \leftarrow \tilde{x}$
- 20 **return** \hat{x}

reliability branching (Achterberg et al. 2005) aim at a good subdivision of the problem. In modern MIP solvers, the basic and simple idea of diving heuristics (see Algorithm 1) is extended by constraint propagation (see Algorithm 1 Line 8) and conflict analysis (see Algorithm 1 Line 11 and 15).

Conflict Analysis for Infeasible LP Relaxations. If the LP relaxation of a subproblem of (1) with local bounds ℓ', u' is proven to be infeasible there exists a dual ray $(y, s) \in \mathbb{R}_+^m \times \mathbb{R}^n$ by the Lemma of Farkas (Farkas 1902, Dinh and Jeyakumar 2014) such that

$$y^\top A + s = 0 \tag{3}$$

$$y^\top b + s\{\ell', u'\} > 0, \tag{4}$$

where we use the notations $\{\ell', u'\} := \sum_{j \in \mathcal{N}: s_j > 0} s_j \ell'_j + \sum_{j \in \mathcal{N}: s_j < 0} s_j u'_j$. Therefore, the inequality

$$(y^\top A)x \geq y^\top b \quad (5)$$

is globally valid. We refer to (5) also as *Farkas proof*. Pólik (2015) and Witzig et al. (2017) describe how these constraints can be collected, managed, and used for constraint propagation to deduce tighter variable bounds in modern MIP solvers.

3 Farkas Diving

Our first aim is the design of a diving procedure such that the dual solution of the LP relaxation moves towards a valid Farkas proof, i.e., constraints (3) and (4) are satisfied. Suppose x^* is an optimal but fractional solution of a local LP relaxation with respect to bounds ℓ' and u' . Let (y^*, r^*) be an optimal solution of the dual LP

$$\max\{y^\top b + r\{\ell', u'\} \mid y^\top A + r = c, y \in \mathbb{R}_+^m, r \in \mathbb{R}^n\}, \quad (6)$$

where r_j is the *reduced cost* of x_j , for all $j \in \mathcal{N}$. The dual solution (y^*, r^*) is not feasible for (3) and (4) with $(y, s) = (y^*, r^*)$, but note that $(y, s) = (y^*, r^* - c)$ fulfills (3).

In order to reduce the violation of (4), we need to increase the lower bound ℓ'_j of x_j if $r_j^* - c_j > 0$ and decrease the upper bound u'_j if $r_j^* - c_j < 0$. By complementary slackness, all integer variables with fractional LP solution value have reduced costs zero, i.e., $r_j^* = 0$. Here, without loss of generality we assume $\ell'_j, u'_j \in \mathbb{Z}$ for all $j \in \mathcal{I}$. Hence, tightening the lower or upper bound of variable j reduces the violation of (4) by $|c_j| \cdot d_j$, where

$$d_j := \begin{cases} \lceil x_j^* \rceil - \ell'_j & \text{if } c_j < 0, \\ u'_j - \lfloor x_j^* \rfloor & \text{if } c_j > 0. \end{cases} \quad (7)$$

Therefore, the rounding direction of a variable j is implied by the sign of the objective coefficient c_j , i.e., upwards if $c_j < 0$, downwards if $c_j > 0$.

The previous part of this section took a strictly dual point of view. When switching the perspective to the primal side, the above rounding procedure can be interpreted as follows. On variables with negative objective coefficients we always tighten the lower bounds, i.e., the solution values of those variables are pushed towards the upper bound, and vice versa. Since (1) is a minimization problem, all variables are rounded into the best direction with respect to the objective function c . If $c_j = 0$ neither pushing to the lower nor upper bound has a direct impact on the objective function. In that case, a natural choice for breaking this tie is to consider the *fractionality* $f_j := x_j^* - \lfloor x_j^* \rfloor$ of the corresponding LP solution value. This leads to the following diving heuristic, defined by a rounding

function ϕ_F and scoring function ψ_F . For every variable $j \in \mathcal{I}$, let

$$\phi_F(j) := \begin{cases} \text{up} & \text{if } c_j < 0 \text{ or } c_j = 0 \wedge f_j \geq \frac{1}{2}, \\ \text{down} & \text{if } c_j > 0 \text{ or } c_j = 0 \wedge f_j < \frac{1}{2}. \end{cases} \quad (8)$$

While we use the fractionality only for tie-breaking, other diving heuristics, e.g., fractionality diving (Achterberg 2007b), use this criterion solely to determining the rounding directions.

In addition to the rounding direction ϕ_F , an order needs to be defined in which the diving candidates are explored. As discussed before, the violation of the dual infeasibility constraint (4) can be reduced by $|c_j| \cdot d_j$ when tightening the lower or upper bound of variable j . From the primal point of view, the potential change in the objective function depends on how much a variable can be pushed until it reaches one of its bounds. This change can be approximated by $|c_j| \cdot f_j^{rel}$, where

$$f_j^{rel} := \begin{cases} 1 - f_j & \text{if } \phi_{F(j)} = \text{up}, \\ f_j & \text{if } \phi_{F(j)} = \text{down}, \end{cases} \quad (9)$$

is the *relative fractionality* of x_j^* , for all $j \in \mathcal{I}$. This measure is used by, e.g., pseudo cost diving (Achterberg 2007b), to define an ordering in which the variables are explored during diving.

Combining both criteria, let the score of $j \in \mathcal{I}$ be given by

$$\psi_F(j) := |c_j| \cdot d_j \cdot f_j^{rel}. \quad (10)$$

A higher score is preferred.

In the following, we refer to this diving heuristic as *Farkas diving*. Its rounding procedure pushes all variable towards the so-called *pseudo solution* (Achterberg 2007b). In the pseudo solution, each variable takes the best bound with respect to its objective coefficient as solution value. This relaxation solution is overly optimistic and most of the time not feasible for the constraints $Ax \geq b$. However, if this strategy leads to a feasible solution, it may be expected to be very good.

4 Conflict Diving

We continue by exploring the complementary question of how conflict constraints can be used to guide the search of a diving heuristic. First, consider the following well-known concept.

Definition 1 (Variable Locks (Achterberg 2007b)). *For a mixed integer program of form (1), the number of down-locks and up-locks of variable $j \in \mathcal{N}$ is defined as the number of positive coefficients per column $\zeta_j^+ := |\{i \mid A_{ij} > 0\}|$ and the number of negative coefficients per column $\zeta_j^- := |\{i \mid A_{ij} < 0\}|$, respectively.*

Starting at an LP solution x^* satisfying $Ax \geq b$, a variable $j \in \mathcal{I}$ with zero down-locks ζ_j^+ (with zero up-locks ζ_j^-) can always be set to its lower bound (upper bound) without increasing the violation of any constraint. On the other hand, if a variable has down-locks (up-locks), rounding the variable downwards (upwards) might increase the violation of at least one constraint. If a constraint down-locks variable j and is tight with respect to the LP solution candidate x^* , rounding x_j^* downwards will violate the constraint. Hence, the variable locks measure the “risk” that rounding a variable leads to additional constraint violations.

Usually, only variable locks that are implied by model constraints (short: variable locks) are used during a MIP solve, e.g., during presolving, propagation, or primal heuristics. A well-known diving heuristic that solely relies on variable locks is *coefficient diving* (Berthold 2008). coefficient diving as implemented in SCIP follows the diving scheme of Algorithm 1. For every variable the rounding direction is determined based on the variable locks only. For a variable j that is locked in both directions, i.e., $\zeta_j^- > 0$ and $\zeta_j^+ > 0$, coefficient diving prefers the direction with less variable locks, i.e., the “safe” direction. The order to process the variables during diving is given by the number of variable locks, too. Here, variables with many variable locks on the chosen direction are preferred.

Processing variables first that tend to lead to infeasibilities is often called a *fail fast strategy*. This strategy was introduced by Haralick and Elliott (1980) in the context of artificial intelligence and constraint programming. Later, Berthold (2014a) verified that taking the most critical decisions first is a good strategy for MIP.

Variable locks are a very static criterion and include also model constraints that either do not propagate frequently or are not tight at the current LP relaxation. For this reason, we propose to consider also variable locks implied only by conflict constraints (short: *conflict locks*), which can be defined analogous to Definition 1. The following lemma shows that conflict locks can have the effect of measuring the “risk” more accurately.

Lemma 2. *Let $(y^\top A)x \geq y^\top b$ be a Farkas proof derived from an infeasible LP and (y, r) a dual ray proving the infeasibility of this local subproblem. If the conflict contributes to the conflict up-locks (conflict down-locks) of j , then, there exists at least one model constraint that contributes to the variable up-locks (variable down-locks) of j .*

Proof. The coefficient of x_j in the conflict constraint $(y^\top A)x \geq y^\top b$ is $\bar{a}_j := \sum_{k=1}^m a_{kj} \cdot y_i$. If $\bar{a}_j < 0$, i.e., the conflict up-locks x_j then there must exist at least one constraint k with $a_{kj} < 0$ because $y \geq 0$. Hence, constraint k contributes to the variable up-locks of x_j . The analogous argument holds for $\bar{a}_j > 0$. \square

This motivates the use of conflict locks: they measure the “risk” of violating both model and conflict constraints but focus on constraints that have been actively involved in pruning branch-and-bound nodes.

A similar concept which is used in SAT is VSIDS (variable state independent decaying sum) (Moskewicz et al. 2001). The VSIDS score takes the contribution of every variable (and its negated complement) into account. For every variable the number of clauses (in MIP speaking: conflict constraints) the variable is part of is counted. During continuing the search the VSIDS are periodically scaled by a predefined constant. With this periodically scaling the weight of older clauses is reduced over the time and more recent observation are weighted higher. In contrast to VSIDS, conflict locks are not periodically scaled and only respect conflict constraints that are part at the current subproblem. This is especially interesting within a MIP solver that uses a pool-based approach to maintain the conflict constraints (Witzig et al. 2017). Therefore, conflict locks give rise to the current set of variables that are involved in conflict constraints, whereas VSIDS also incorporate past conflict information.

While classical coefficient diving solely relies on variable locks, we exploit Lemma 2 and use a combination of both variable and conflict locks. Given a weight $\kappa \in [0, 1]$, the *up-weight* ρ_j^- and *down-weight* ρ_j^+ of a variable j is given as a convex combination of both lock types, i.e., $\rho_j^- := \kappa \cdot \xi_j^- + (1 - \kappa) \cdot \zeta_j^-$ and $\rho_j^+ := \kappa \cdot \xi_j^+ + (1 - \kappa) \cdot \zeta_j^+$, where ξ_j^- and ξ_j^+ denote the number of conflict up-locks and conflict down-locks, respectively.

Furthermore, we revert the rounding strategy of coefficient diving: we use a rounding function preferring the direction that is more likely to lead to infeasibilities whenever variable j has locks in at most one direction,

$$\phi_C(j) := \begin{cases} \mathbf{up} & \text{if } \rho_j^- > \rho_j^+ \text{ or } \rho_j^- = \rho_j^+ \wedge f_j \geq \frac{1}{2}, \\ \mathbf{down} & \text{if } \rho_j^+ > \rho_j^- \text{ or } \rho_j^- = \rho_j^+ \wedge f_j < \frac{1}{2}, \end{cases} \quad (11)$$

Here we may assume that all variables that have no locks at all, i.e., free variables, are already set to their best bound with respect to their objective coefficient. With this strategy we aim to guide the heuristic into parts of the search tree that are usually not explored by other heuristics. The order in which the diving candidates are explored is identical to the one used in coefficient diving,

$$\psi_C(j) := \begin{cases} \rho_j^- & \text{if } \phi_C(j) = \mathbf{up}, \\ \rho_j^+ & \text{if } \phi_C(j) = \mathbf{down}. \end{cases} \quad (12)$$

i.e., variables that have a large number of locks on the chosen rounding direction are preferred. With this scoring function we pursue a fail fast strategy in order to reduce the time spent by this heuristic. We use a fail fast strategy that is even more aggressive than coefficient diving since we already choose the critical direction. As a result, the heuristic tries to process variables that tend to infeasibility at the beginning of the diving path before the degree of freedom is further reduced during the dive. In the following, we refer to this diving heuristic as *conflict diving*.

5 Computational Results

In order to investigate whether and how MIP solvers can benefit from the methods presented in this paper, we carried out an extensive computational study regarding the individual impact of each heuristic. For each heuristic, we present computational results on its general performance impact, followed by additional experiments that provide more detailed insights regarding their particular properties.

In the first part of this section, we compare SCIP in its default configuration (`default`) to SCIP extended by Farkas diving. We will refer to the latter setting by `farkdiving`. In the second part of this section, we compare the individual impact of coefficient diving and conflict diving, to which we will refer by `coefdiving` and `confdiving`, respectively. As a baseline we use SCIP without both lock-exploiting diving heuristics (`nolockdiving`).

All experiments were performed with the academic MIP solver SCIP (Gleixner et al. 2018) (git hash bf6a486, based on SCIP 6.0), using SoPlex 4.0 as LP solver. To evaluate the generated data the *interactive performance evaluation tool* (IPET) (Hendel) was used. The experiments were run on a cluster of identical machines equipped with Intel Xeon E5-2690 CPUs with 2.6 GHz and 128 GB of RAM; a time limit of 7200 seconds was set. To account for the effect of performance variability (Danna 2008, Lodi and Tramontani 2013) all experiments were performed with four different global random seeds. As test set we used a union of MIPLIB 3 (Bixby et al. 1998), MIPLIB 2003 (Achterberg et al. 2006), MIPLIB 2010 (Koch et al. 2011), and the COR@L (Linderoth and Ralphs 2005) benchmark set. After removing all duplicates and problems that are known to be numerically unstable, the test set consists of 488 publicly available MIP problems, which we will refer to as MMMC. Every pair of MIP problem and seed is treated as an individual observation, effectively resulting in a test set of 1952 instances. We will use the term “instance” when referring to a problem-seed combination.

Aggregated results over all random seeds are shown in Table 1 and Table 3. Here, 5 and 11 instances, respectively, are excluded because at least one setting finished with numerical violations. Besides the results on MMMC, the tables state the impact on affected instances, i.e., instances for which the solving path differs among settings. Further, the subset of affected instances is grouped into a hierarchy of increasingly harder classes $[k, \text{tilim}]$. Class $[k, \text{tilim}]$ contains all instances for which all settings need at least k seconds and can be solved by at least one setting within the time limit. As explained by Achterberg and Wunderling (2013), this excludes instances that are “easy” for all settings in an unbiased manner. Detailed tables with instance-wise computational results can be found in the electronic supplement.

5.1 Farkas Diving

We first present computational experiments regarding the general impact of Farkas diving as proposed in Section 3. In this setup, Farkas diving solved the

Table 1: Aggregated computational results for Farkas diving on MMMC over four different random seeds. Relative changes by at least 5% are highlighted in bold.

	#	default					farkdiving				
		S	T	N	F _{virt}	I _{virt}	S	T _Q	N _Q	F	I
all	1947	1401	183	3366	2.653	0.806	1408	0.982	0.947	3.725	0.335
MPLIB 2010	348	305	349	6225	3.454	0.828	307	0.966	0.940	4.862	0.379
affected	706	686	66	1204	4.238	0.564	693	0.946	0.852	5.924	0.585
[10,tilim]	529	509	186	2420	5.168	0.733	516	0.918	0.808	6.112	0.573
[100,tilim]	297	277	697	4444	6.997	0.980	284	0.852	0.737	6.862	0.690
[1000,tilim]	127	107	2599	8072	11.764	1.291	114	0.837	0.715	10.803	1.150
afterroot	273	270	34	1095	9.747	0.689	272	0.879	0.783	15.319	1.513

LP relaxation at every diving node (cf. Line 13). Since this strategy is costly compared to other diving heuristics in SCIP, Farkas diving was executed at local nodes of the search tree, if the heuristic could already find a feasible solution at the root node. Moreover, since Farkas diving relies on the objective function, we executed the heuristic on instances with a nonzero objective function only. In the second part, we discuss additional computational experiments used to analyze the effect of the individual components of Farkas diving.

Overall Impact of Farkas Diving. Aggregated computational results comparing SCIP in its default configuration (**default**) and SCIP extended by Farkas diving (**farkdiving**) on MMMC are shown in Table 1. Due to the restricted execution strategy described above, **farkdiving** affected only 706 out of 1947 instances. However, on these 706 instances **farkdiving** led to a speed-up of 5.4% (**T_Q**) and reduced the tree size by 14.8% (**N_Q**).

On the subset of harder instances [100,tilim] the overall performance could even be improved by 14.8% and the size of the search tree could be reduced by more than 25%. The observed performance improvement is spread over the complete group of affected instances. This becomes apparent also in the performance profiles (Dolan and Moré 2002, Gould and Scott 2016) displayed in Figure 1. For a time factor of 1.0 the respective amount of instances that could be solved best with the respective setting is marked with a colored cross. In our experiments, we observed that on the set of affected instances **default** performs best on 369 instances, whereas **farkdiving** was marginal worse and performs best on 365 affected instances. On this group of instances, both profiles cross exactly once at time factor 1.021. Afterward, **farkdiving** is always superior to **default**. This fact indicates that **farkdiving** is especially superior to **default** on harder instances and is confirmed by the performance profiles over instances where both settings need at least 10, 100 or 1000 settings. On these groups of affected instances, **farkdiving** is clearly superior to **default**. For a time factor of 1.0 **farkdiving** performs best the harder the instances are. On the group of affected instances for which both settings need at least 10 seconds 248 can be solved best by **default**, whereas **farkdiving** performs best on 281 instances.

These results indicate that Farkas diving is expensive on easy instances compared to `default`, but superior on harder instances.

Success in Generating Conflicts. Farkas diving is motivated by the idea of explicitly diving towards a valid Farkas proof. To analyze how successful Farkas diving is in generating conflict constraints we considered all affected instances where Farkas diving was allowed to run after the root node, i.e., instances where Farkas diving was able to find a feasible solution at the root node. This subset contains 273 instances and is displayed in line “afterroot” of Table 1. On this subset of instances, Farkas diving was able to find 7.3% more conflict constraint than the “virtual best diving heuristic” in this regard. Here, the virtual best diving heuristic is determined by taking, for each instance run with `default` settings, the diving heuristic that generated the largest number of conflict constraints. Hence, Farkas diving indeed succeeded in generating an above-average number of conflict constraints. Figure 2 provides a more detailed comparison of the number of generated conflict constraints for increasingly hard subgroups of afterroot. The box plots (McGill et al. 1978) show for every setting and instance group the 1st and 3rd quartile (shaded box) as well as the median (dashed line). All observations below the 1st or above the 3rd quartile are marked with shaded diamonds. On all four groups of instances, Farkas diving led to more conflict constraints in the median and 3rd quartile than the virtual best diving heuristic in this regard. On instances where both settings needed at least 1000 seconds, Farkas diving produced on 50% of the instances more conflict constraints than the virtual best diving heuristic of SCIP with `default` settings on 75% did. These results indicate that the strategy of diving towards a valid Farkas proof as performed by Farkas diving leads to additional conflict information and succeeds over the whole set where this strategy is pursued. Note that our analysis is conservative. The virtual best diving heuristic strictly overestimates every single diving heuristic, hence the number of additionally generated conflict constraints would only increase when comparing to each diving heuristic individually.

Success in Generating Solution. Usually, the number of solutions found by diving heuristics is quite small (Khalil et al. 2017). From a primal viewpoint, Farkas diving follows a rounding strategy which leads to overly optimistic solutions that are expected to be infeasible most of the time. However, if this strategy succeeds, the solutions can be expected to be very good. A more detailed look into the success rate of Farkas diving with respect to finding primal solutions answers the question about the impact of these overly optimistic solutions on the overall MIP solver. In our experiments over several seeds, we could observe that on affected instances Farkas diving was able to find 39.7% more feasible solutions (**F**) and 3.7% more improving solutions (**I**) than the virtual best diving heuristic (**F_{virt}** and **I_{virt}**) of SCIP with `default` settings. Here, the virtual best diving heuristic is determined as above, for each instance selecting the heuristic with the largest number of feasible and improving solutions, respectively. Again, both **F**

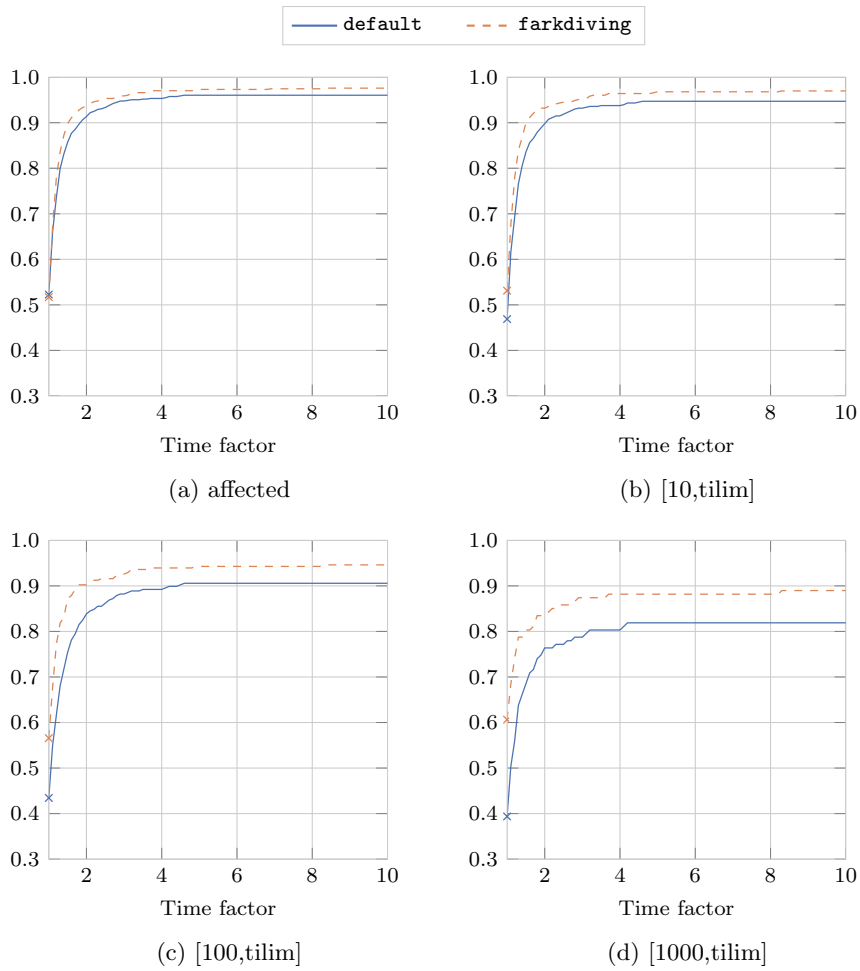


Figure 1: Performance profiles of **default** and **farkdiving** for four hierarchical groups of increasingly hard, affected instances.

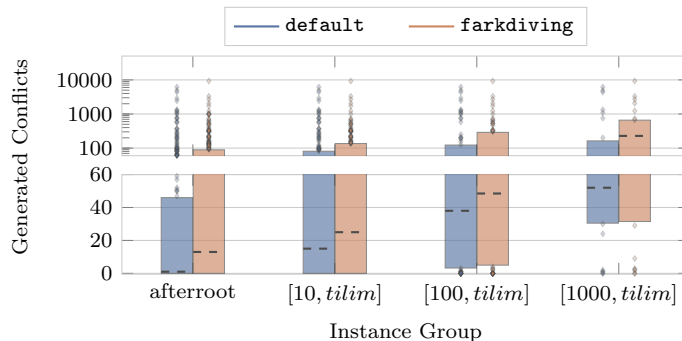


Figure 2: Box plot showing the number of generated conflicts by Farkas diving when running with `farkdiving` and the virtual best diving heuristic when running with `default` on increasingly hard subgroups of `afterroot`.

\mathbf{virt} and $\mathbf{I}_{\mathbf{virt}}$ overestimate every single diving heuristic in `default` setting. Note that this evaluation is conservative also in the sense that it includes instances for which Farkas diving was not allowed to run at local nodes of the tree, i.e., instances where Farkas diving did not find a solution at the root node, and, therefore, did not find a solution at all in these instances.

On the group of instances where Farkas diving was allowed to run within the tree (`afterroot`), the results are even more pronounced. Farkas diving was able to find 57.2% more feasible solutions and more than twice as many improving solutions than the virtual best diving heuristic in `default` setting. Figure 3 plots the relation of feasible and improving solutions found by Farkas diving and the virtual best diving heuristic of `default` on increasingly hard groups of `afterroot`. The box plots show for every setting and instance group the 1st and 3rd quartile of all numbers of feasible and improving solutions, respectively, (shaded box) as well the median (dashed line). All observations below the 1st or above the 3rd quartile are marked with shaded diamonds. On all four groups of instances, Farkas diving was able to find 10 or more feasible solution on at least 25% of the instances (observations above the 3rd quartile), whereas the 3rd quartile of the virtual best diving heuristic in this regard is always 1 for affected and `[10,tilim]`. On hard instances for which both setting need at least 1000 seconds the 1st and 3rd quartiles of both `default` and `farkdiving` are identical. Note, the set `afterroot` only contains instances for which Farkas diving was able to find at least one feasible solution at the root node. Thus, this set of instances is slightly biased; hence, it is not suitable to conclude that Farkas diving finds more feasible solutions than the virtual best diving heuristic in general. However, the results on this set of instances indicate that the rule we used to decide whether Farkas diving is allowed to run within the tree succeeds. On the complementary set consisting of 1562 instances that could be solved by at least one setting, i.e., those for which Farkas diving was not able to find a solution root node, `farkdiving` succeeded with respect to finding at least one feasible solution within the tree

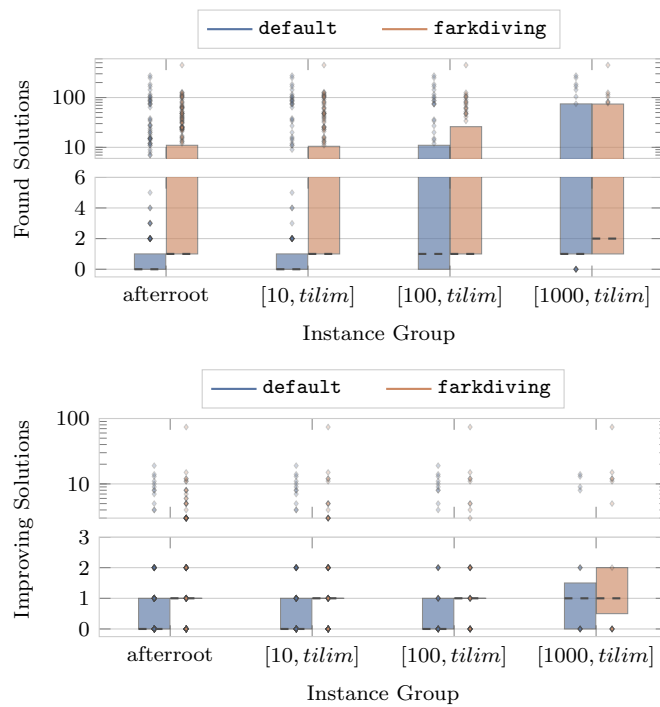


Figure 3: Box plots showing the number of feasible and improving solutions found by Farkas diving when running with `farkdiving` and the virtual best diving heuristic when running with `default`.

Table 2: `farkdiving` with and without adding feasible solutions and generated conflict constraints, respectively.

	#	S	T	N	T _Q	N _Q
<code>default</code>	273	270	33.82	1095	1.000	1.000
<code>farkdiving</code>	273	272	29.75	857	0.879	0.783
<code>farkdiving-noconfs</code>	273	266	31.79	976	0.940	0.891
<code>farkdiving-nosols</code>	273	269	32.73	1045	0.968	0.954

on only 0.019% of the instances. Considering the improving solutions found by the virtual best diving heuristic when running SCIP with `default` settings and Farkas diving indicates that whenever Farkas diving was allowed to run within the tree, i.e., it finds a feasible solution at the root node, Farkas diving yields at least one improving solution on 75% (all observations above the 1st quartile) of the instances of `afterroot`, `[10,tilim]`, and `[100,tilim]`. By contrast, the 1st and 3rd quartile of the virtual best diving heuristic are 0 and 1, respectively. Consequently, we can conclude that on instances that are cumbersome for the established diving heuristics of SCIP 6.0 with respect to finding solutions, Farkas diving can easily find both feasible and improving solutions if it is allowed to run within the tree.

However, the pure amount of feasible and improving solutions alone is not meaningful enough to conclude whether the solutions found by Farkas diving have a positive impact on the overall MIP solver. Therefore, we consider also the *primal integral* (Berthold 2013), which measures the progress of the primal bound towards the optimal solution. Compared to `default`, `farkdiving` improved the primal integral by 12.3%. This exhibits, maybe surprisingly so, that the optimistic strategy of Farkas diving is also very successful on the primal side and, thus, a valuable extension to the portfolio of primal heuristics.

Impact of Primal Solutions and Generated Conflicts. The previous results indicate that Farkas diving both produces an over-average number of solutions and conflict constraints if it is allowed to run within the tree, but do not finally answer which component is more relevant for performance. We tried to quantify this by testing two modified versions of Farkas diving: one that disables conflict analysis (`farkdiving-noconfs`) and one that discards feasible solutions found by Farkas diving (`farkdiving-nosols`). The aggregated results are reported in Table 2. For this experiment, we used the subset of 273 instances where Farkas diving was allowed to be executed after the root node when running Farkas diving, i.e., `afterroot`.

Both settings solve fewer instances than `default`. This result is not surprising since both `farkdiving-noconfs` and `farkdiving-nosols` spend computational time for running Farkas diving. As we have already discussed, Farkas diving is much more expensive than other diving heuristics in SCIP since it solves the LP at every node during diving. Consequently, by not performing conflict analysis

or not adding feasible solutions during Farkas diving one of the key ingredients is missing. Thus, on a few instances, Farkas diving needs the impact of both found solutions and generated conflict constraints to compensate the computational overhead compared to `default`.

When disabling conflict analysis (`farkdiving-noconfs`) the performance improvement compared to `default` decreased from 12.1% to 6%. The number of nodes explored during the tree search also increased (compared to `farkdiving`) but is still 10.9% smaller than SCIP with `default` settings. Disabling the addition of primal solutions found by Farkas diving to the main search (`farkdiving-nosols`) reduced the performance improvement compared to `default` from 12.1% to 3.2%, which corresponds to a slowdown by 9.2% compared to `farkdiving`. Interestingly, the tree increased by 17.9% compared to `farkdiving`. However, `farkdiving-nosols` still led to 4.6% smaller trees than SCIP with `default` settings.

In hindsight, it is not very surprising that disabling the addition of solutions has a larger impact than disabling conflict analysis during Farkas diving. With the very optimistic rounding strategy, Farkas diving was able to find 3.2 times as many best solutions¹ as all remaining diving heuristics together. Overall, on afterroot, 5.7% of all best solutions were found by Farkas diving. The only heuristic that was even more effective on this set of instances is the large neighborhood search heuristic RENS (Berthold 2014b), which contributed 7.9% of all best solutions. Thus, Farkas diving has a remarkable success rate, which leads to a not negligible impact on the primal side. As mentioned above, this is also reflected by the primal integral (Berthold 2013), which improved by 12.3% when running Farkas diving during the tree search (afterroot).

These results make clear that both the primal and dual aspects of Farkas diving contributes to the improved performance, but that the solutions found with the strategy of Farkas diving seem to be the main driver of the heuristic. This may come as a surprise because our initial motivation for the design of Farkas diving was the targeted generation of conflict constraints.

Impact of LP Frequency. By default, all diving heuristics in SCIP are configured to solve the LP dependent on the number of found bound deductions during constraint propagation over all variables. An LP solve is triggered whenever the number of bound changes since the last LP solve exceeds 0.15 times the number of variables. By contrast, Farkas diving solves the LP at every diving node. One motivation for this high LP frequency is to “pull” the variable assignments back towards the feasible region in order to counteract the very optimistic rounding strategy of Farkas diving, which tends to “push” the variable assignments out of the feasible region.

Within SCIP, every LP solution found during diving is automatically rounded to a solution satisfying all integrality requirements of the variables. If this solution also satisfies all constraints, i.e., is feasible for the entire MIP, the

¹A solution is called 'best' if it is an optimal solution or the best-known solution when reaching the time limit.

solution is added to the solution storage, whereby the diving heuristic that performs the current dive is credited for finding the feasible solution. Thus, Farkas diving may have a slight advantage over all other diving heuristics since it solves the LP relaxation more frequently. This fact might be one reason why Farkas diving was able to find more than 40% more solutions than the virtual best (\mathbf{F}_{virt}), see Table 1. Hence, in a final control experiment, we configured Farkas diving identically to all remaining diving heuristics in order to measure the impact of the increased LP frequency. We will refer to this configuration by `farkdiving-lp`.

On the set of affected instances for which our criteria for running Farkas diving within the tree is satisfied (afterroot), `farkdiving-lp` performed almost as “poorly” as `farkdiving-nosols` (see Table 2) with respect to solving time, nodes, and number of solved instances. Thus, when using `farkdiving-lp` the solving time could be improved by only 3.5% compared to `default`. Note, on the same set of instances `farkdiving` led to a speed-up of 12.1%. Compared to `farkdiving`, the amount of feasible solutions found by `farkdiving-lp` was reduced by 90%.

These results show that solving the LP relaxation frequently gives an important boost to the degree by *how much* Farkas diving improves performance. However, also Farkas diving with less LP solves, i.e., configured identically to all remaining diving heuristics, outperforms the `default` settings. Consequently, the fact *that* Farkas diving works indeed seems to be a result of its specific choice of rounding function ϕ_F and scoring function ψ_F .

5.2 Conflict Diving

The new conflict diving is closely related to the existing coefficient diving in the sense that they both exploit lock information and follow the same diving framework of Algorithm 1. Hence, we chose to include both in the computational analysis and compare their impact individually to SCIP without any of these lock-exploiting diving heuristics. In the following, we will refer to the latter setting by `nolockdiving`. We will refer to SCIP with conflict diving and with coefficient diving activated by `confdiving` and `coefdiving`, respectively. We first present computational experiments to quantify the general performance impact, followed by further computational results to analyze the impact of individual components in more detail.

Overall Impact of Conflict Diving. In our computational setup, conflict diving used the same parameter settings as coefficient diving, e.g., frequency of execution and LP solve frequency. The parameter to weight variable and conflict locks within conflict diving was set to $\kappa = 0.75$, i.e., conflict locks dominate by a factor of 3. Aggregated computational results of all three configurations on MMMC are shown in Table 3.

While `coefdiving` only showed minimally improved performance compared to `nolockdiving`, the setting `confdiving` was clearly superior. Conflict diving increased the number of solved instances by 16 (**S**), led to an overall speed-up

Table 3: Aggregated computational results for coefficient and conflict diving on MMMC over four different random seeds. Relative changes by at least 5% are highlighted in bold.

	#	nolockdiving			coefdiving				confdiving			
		S	T	N	S	T _Q	N _Q	C	S	T _Q	N _Q	C
all	1941	1372	216	3108	1381	0.998	1.003	1386.4	1388	0.978	0.972	1364.4
MiPLIB 2010	347	297	425	5788	299	1.008	1.015	374.9	300	0.986	0.975	343.6
affected	861	822	186	4259	831	0.998	1.014	686.2	838	0.952	0.945	892.2
[10,tilim]	775	736	287	5356	745	0.998	1.018	760.9	752	0.948	0.941	990.0
[100,tilim]	542	503	707	7982	512	0.988	1.000	1045.2	519	0.932	0.923	1377.9
[1000,tilim]	246	207	2365	19538	216	0.984	0.981	2068.5	223	0.874	0.882	2820.5

of 4.8% (**T_Q**), and reduced the tree size by 5.5% (**N_Q**) on affected instances. On the subset of affected instances, **confdiving** needed 4.6% less solving time and up to 6.8% fewer nodes compared to **coefdiving**. The node reduction on the affected instances may be explained by the increased number of generated conflict constraints (**C**).

Concerning the number of solutions, we observed that **confdiving** was only slightly more successful: **confdiving** found 11.1% more feasible solutions and 4.7% more improving solutions than **coefdiving**. The impact of these additional solutions is reflected by the primal integral, which decreased by 4.7% compared to **coefdiving** and by 4.9% compared to **nolockdiving**. However, both heuristics had a small success rate with respect to finding solutions: 0.037 (conflict diving) and 0.035 (coefficient diving) solutions per dive.

Finally, the performance profiles in Figure 4 show that the overall performance improvement was not caused by few instances but was spread over the complete group of affected instances: **confdiving** dominates **coefdiving** on all four groups of increasingly hard instances. For a time factor of 1.0 the respective percentage of instances that could be solved fastest with the respective setting is marked with a colored cross. On all four groups of instances, **confdiving** is clearly superior to **coefdiving** in the sense that the profiles do not cross. Between 52.0% and 54.8% of the instances in the respective group were solved fastest by **confdiving**, whereas only between 44.8% and 46.5% of the instances were solved fastest by **default**.

Length of Diving Paths. Both the rounding and scoring functions used by conflict diving aim for a fail fast strategy (Haralick and Elliott 1980, Berthold 2014a). In order to analyze whether conflict diving indeed achieves this design goal, we additionally measured the average length of diving paths for **coefdiving** and **confdiving**. On average, **confdiving** exhibits 30.2% shorter diving paths than **coefdiving** on affected instances.

The distribution of the length of diving paths over the subsets of affected instances can be seen in Figure 5. The box plots show for every setting and instance group the 1st and 3rd quartile of all diving path lengths (shaded box)

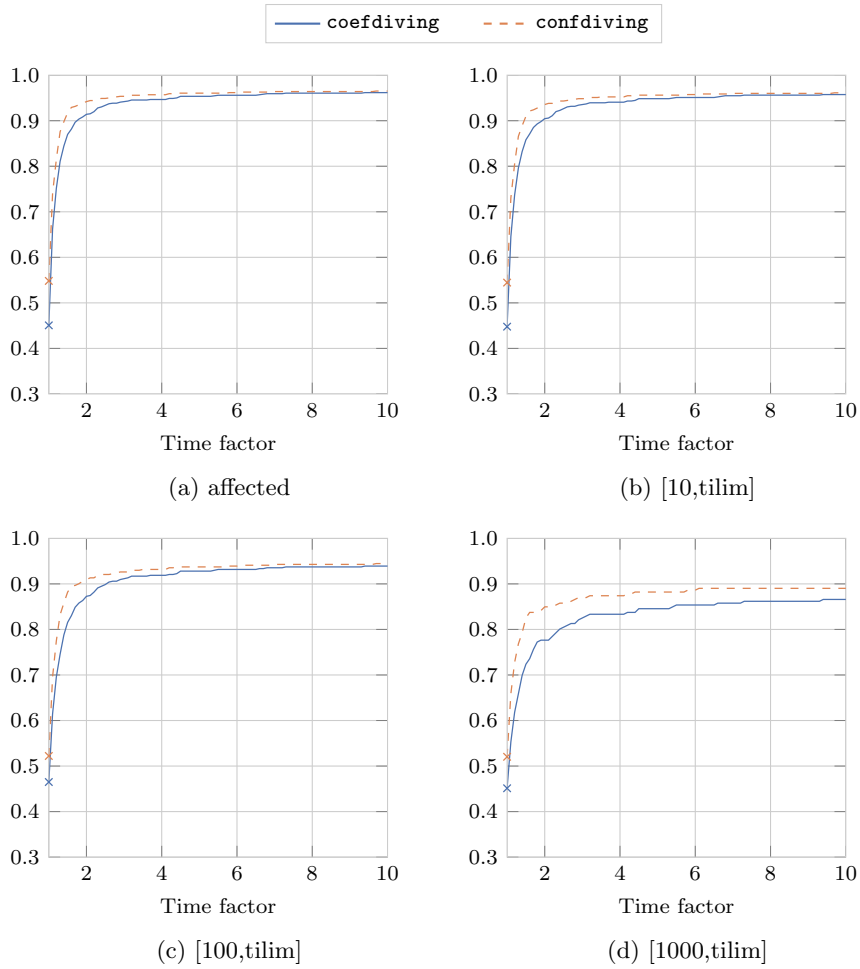


Figure 4: Performance profiles of `coefdiving` and `confdiving` for four hierarchical groups of increasingly hard affected instances.

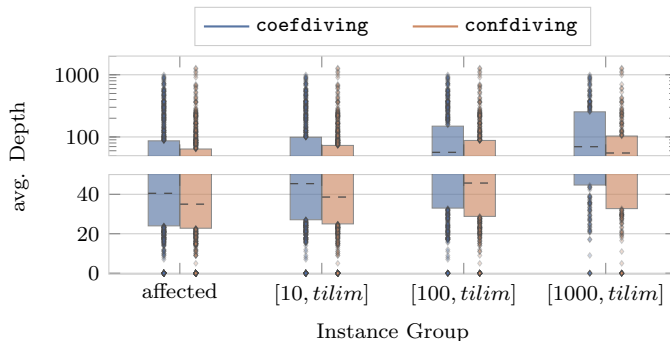


Figure 5: Box plots showing the average depth of diving paths generated coefficient diving with `coefdiving` and conflict diving with `confdiving` on the set of affected instances.

as well the median (dashed line). All observations below the 1st or above the 3rd quartile are marked with shaded diamonds. For all four instance groups, the 1st and 3rd quartile corresponding to conflict diving are smaller than for coefficient diving. The same observation holds for the median in all cases. For example, on `[100,tilim]` the 3rd quartile (149) of coefficient diving is 69.3% larger than the 3rd quartile (87) of conflict diving. On the same group of instances, the observed median path length of conflict diving is 23.6% shorter than the one of coefficient diving.

As we already discussed, both coefficient diving and conflict diving showed only a small number of found solutions per diving path, which is a good proxy for the number of successful paths, i.e., paths without backtracking due to infeasibility (cf. Line 11 and 15). Thus, these statistics confirm that conflict diving succeeds better in implementing a fail fast strategy and aborting the many “unsuccessful” dives not leading to a feasible solution early.

Moreover, this does not come at the expense of learning less conflict constraints. As can be seen in column C of Table 3, `confdiving` created 30.0% more conflict constraints.

Impact of Conflict Locks. In order to investigate whether conflict diving outperforms coefficient diving because of the inclusion of conflict locks or merely because of the difference in the rounding and scoring function, we conducted two further control experiments with

- a modified rounding function within conflict diving and
- different weights κ .

For the first control experiment we modified conflict diving to use the same rounding function as `coefdiving` and $\kappa = 0.5$ for the scoring function. The

resulting diving heuristic `conf-like-coefdiving` behaves the same as coefficient diving except for using a convex combination of variable and conflict locks in the variable selection score.

In our experiment `conf-like-coefdiving` was superior to `coefdiving` and led to a performance improvement of 2.3% on the set of affected instances. At the same time, the tree size could be reduced by 3.6% and four more instances could be solved. Similar to the actual version of conflict diving evaluated at the beginning of this section, conflict diving with `conf-like-coefdiving` settings generated more than twice as many conflict constraints and found 8% more improving solutions than coefficient diving. The performance profiles in Figure 6 show that conflict diving with the same rounding function like coefficient diving performs better the harder the instance, i.e., whenever many conflict constraints and therefore reliable conflict locks can be expected. It is not surprising that for a time factor of 1.0 `coefdiving` solves more instances best when easy instances are considered, too, since the number of conflict constraints can be expected to be small and, thus, the information gained by conflict locks might not be reliable enough. Even though the profiles are close to each other, they do not cross if both settings need at least 10, 100, or 1000 seconds. This result indicates that it is the inclusion of conflict locks in general that helps to guide diving better than pure variable locks.

For the second control experiment we varied the weights $\kappa \in \{0, 0.25, 0.5, 0.75, 1\}$ in `confdiving` in order to quantify the importance of conflict locks. The results indicate that the reduced length of diving paths is independent of the weighting of variable and conflict locks. For every choice of κ the average length of diving paths was reduced by 18% to 34%. Hence, the length of the diving paths seems to mainly depend on the rounding function ϕ_C , which always prefers rounding into the “risky” direction, i.e., the direction with more locks. We also confirmed that `confdiving` is superior to `coefdiving` on the affected instances for every evaluated κ .

Moreover, we observed that on instances with zero objective function a smaller conflict weight κ yields superior performance, while for instances with nonzero objective function a larger weight κ for conflict information seems to be the better choice. This observation may be related to how and when the decisions of conflict diving align with SCIP’s default branching rule and when they complement it. To make this clear, further background information is necessary. For branching variable selection, SCIP combines reliability pseudocost-branching (Achterberg et al. 2005), which estimates dual bound improvement, and hybrid branching (Achterberg and Berthold 2009), which includes conflict information via VSIDS (Moskewicz et al. 2001). Both VSIDS and conflict locks approximate the set of variables that frequently appear in conflict constraints. On problems with nonzero objective function typically the first, objective-based score dominates SCIP’s branching decisions, while on pure feasibility problems the latter, conflict-based score has more impact. This explains that on feasibility problems a larger κ might align the decisions of conflict diving unfavorably with the overall tree search and create redundancy; on the majority of problems with nonzero objective, where conflict information is underweighted in the branching

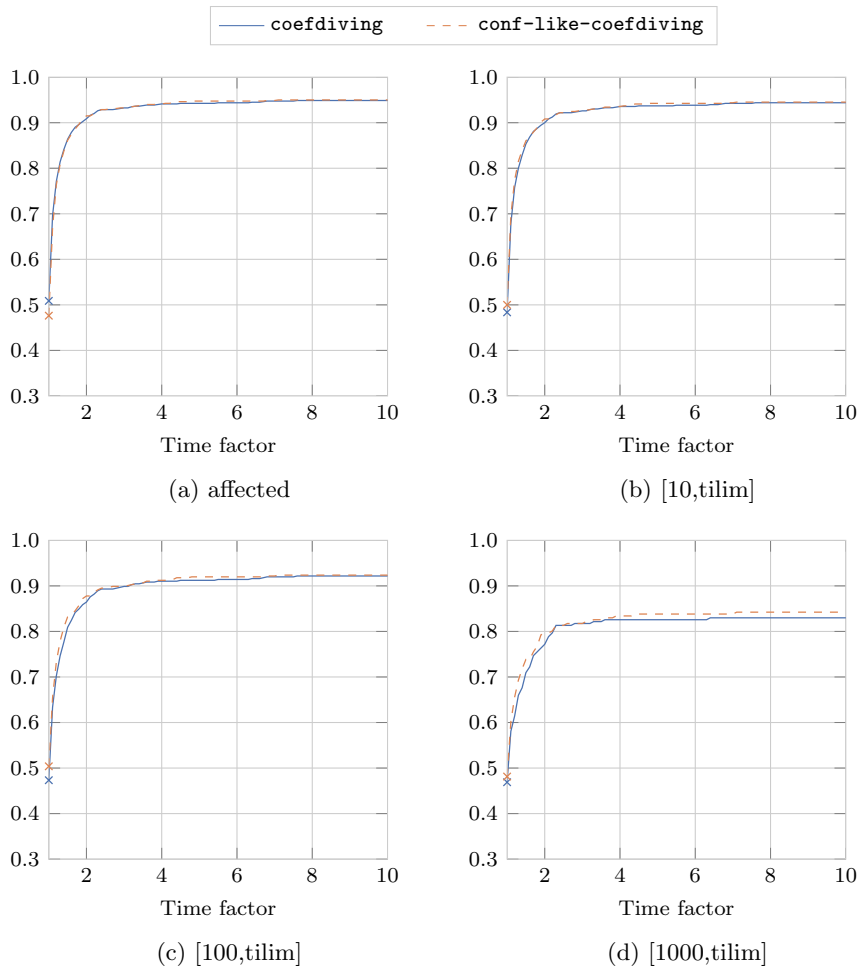


Figure 6: Performance profiles of `coefdiving` and the conflict diving modification `conf-like-coefdiving` for four hierarchical groups of increasingly hard affected instances.

Table 4: `confdiving` with and without adding feasible solutions and generated conflict constraints, respectively, on the set of affected instances.

	#	S	T	N	T _Q	N _Q
<code>nolockdiving</code>	861	822	185.62	4235	1.000	1.000
<code>coefdiving</code>	861	831	185.25	4290	0.998	1.013
<code>confdiving</code>	861	838	176.57	3999	0.951	0.944
<code>confdiving-noconfs</code>	861	825	181.99	4131	0.980	0.976
<code>confdiving-nosols</code>	861	838	177.64	4006	0.957	0.946

rule, a larger κ enables conflict diving to contribute more complementary information to the search. This suggests a further refinement of conflict diving by adapting the weight κ dynamically to the problem at hand.

Impact of Primal Solutions and Generated Conflicts. Finally, we analyzed the importance of found solutions and generated conflict constraints by comparing `confdiving` to two artificially modified versions of conflict diving: one that does not apply conflict analysis (`confdiving-noconfs`) and one that discards feasible solutions found by conflict diving (`confdiving-nosols`). Aggregated results on the set of affected instances are shown in Table 4.

Both variants `confdiving-noconfs` and `confdiving-nosols` are superior to `coefdiving` with respect to solving time and number of nodes. Whereas `confdiving-nosols` solved the same amount of instances as `confdiving`, disabling conflict analysis led to solving 13 instances less than `confdiving`. Hence, disabling the addition of feasible solutions found by conflict diving only has a marginal impact compared to standard conflict diving. By contrast, disabling conflict analysis during conflict diving leads to a slowdown of 3% compared to `confdiving`. Thus, the conflict constraints generated during conflict diving seem to be the main driver of the heuristic. This also aligns with our observations regarding the *dual integral* (Berthold 2013). The dual integral measures the progress of the dual bound towards the optimal objective value. This measure increased by 3.1% on average when disabling conflict analysis, which indicates that the increased number of conflict constraints helps to accelerate convergence of the proof of optimality.

6 Conclusion

In this paper, we presented two new ways how conflict analysis and primal heuristics can be combined to improve the performance of a MIP solver. We presented a primal heuristic, called Farkas diving, that simultaneously aims to construct valid Farkas proofs and feasible solutions. By design, Farkas diving is more expensive than previous diving heuristics in SCIP. Therefore, the heuristic is called conservatively, whereby the decision to keep the heuristic enabled for the remainder of the search is based on its success during the root node. On the

set of instances where Farkas diving was executed after the root node, it proved to be very successful in generating improving solutions and conflict constraints. Regarding both metrics, Farkas diving outperforms the virtual best of all other diving heuristics on this set of instances. Moreover, the overall solving time could be improved by 5.4% and the tree size could be reduced by 14.8% on the set of affected instances.

Furthermore, we applied the concept of variable locks to conflict constraints and used this additional information to guide the search of a second diving heuristic, called conflict diving. In addition, conflict diving pursues an aggressive fail fast strategy to prevent fruitless consumption of running time. This new diving heuristic is an extension and modification of the well-known coefficient diving heuristic. Our computational results indicate that this mix of variable and conflict locks in combination with an aggressive fail fast strategy outperforms coefficient diving on our complete test set. Conflict diving reduces the overall solving time and tree size on affected instances by 4.8% and 5.5%, respectively.

The two ways of combining primal heuristics and conflict analysis presented in this paper have highlighted that primal and dual solving techniques within a general-purpose MIP solver are not independent and that they can not only interact randomly and create performance variability, but be combined beneficially in a targeted manner.

Acknowledgments Many thanks to Timo Berthold for valuable discussions and Gregor Hendel for his continuous effort in developing IPET. The work for this article has been conducted within the Research Campus Modal funded by the German Federal Ministry of Education and Research (grant number 05M14ZAM) and has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 773897 (plan4res). The content of this paper only reflects the author’s views. The European Commission / Innovation and Networks Executive Agency is not responsible for any use that may be made of the information it contains.

References

- Achterberg T (2007a) Conflict analysis in mixed integer programming. *Discrete Optimization* 4(1):4–20.
- Achterberg T (2007b) Constraint integer programming.
- Achterberg T, Berthold T (2009) Hybrid Branching. van Hoesel WJ, Hooker JN, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 309–311 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-642-01929-6.
- Achterberg T, Koch T, Martin A (2005) Branching rules revisited. *Operations Research Letters* 33(1):42–54.
- Achterberg T, Koch T, Martin A (2006) MIPLIB 2003. *Operations Research Letters* 34(4):361–372, URL <http://dx.doi.org/10.1016/j.orl.2005.07.009>.

- Achterberg T, Wunderling R (2013) Mixed integer programming: Analyzing 12 years of progress. Jünger M, Reinelt G, eds., *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, 449–481 (Springer Berlin Heidelberg), URL http://dx.doi.org/10.1007/978-3-642-38189-8_18.
- Berthold T (2008) Heuristics of the Branch-Cut-and-Price-Framework SCIP. Kalcsics J, Nickel S, eds., *Operations Research Proceedings 2007*, 31–36.
- Berthold T (2013) Measuring the impact of primal heuristics. *Operations Research Letters* 41(6):611–614, URL <http://dx.doi.org/10.1016/j.orl.2013.08.007>.
- Berthold T (2014a) *Heuristic algorithms in global MINLP solvers*. Ph.D. thesis, URL <http://www.zib.de/berthold/Berthold2014.pdf>.
- Berthold T (2014b) RENS. *Mathematical Programming Computation* 6(1):33–54, ISSN 1867-2957, URL <http://dx.doi.org/10.1007/s12532-013-0060-9>.
- Berthold T, Feydy T, Stuckey PJ (2010) Rapid Learning for Binary Programs. Lodi A, Milano M, Toth P, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 51–55 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-642-13520-0.
- Berthold T, Stuckey PJ, Witzig J (2018) Local Rapid Learning for Integer Programs. Technical Report 18-56, ZIB, Takustr. 7, 14195 Berlin.
- Bixby ER, Fenelon M, Gu Z, Rothberg E, Wunderling R (2000) MIP: Theory and Practice — Closing the Gap. Powell MJD, Scholtes S, eds., *System Modelling and Optimization*, 19–49 (Boston, MA: Springer US), ISBN 978-0-387-35514-6.
- Bixby RE, Ceria S, McZeal CM, Savelsbergh MWP (1998) An Updated Mixed Integer Programming Library: MIPLIB 3.0. *Optima* 58:12–15.
- Dakin RJ (1965) A tree-search algorithm for mixed integer programming problems. *The Computer Journal* 8(3):250–255.
- Danna E (2008) Performance variability in mixed integer programming. *Workshop on Mixed Integer Programming, Columbia University, New York*, volume 20.
- Davey B, Boland N, Stuckey PJ (2002) Efficient Intelligent Backtracking Using Linear Programming. *INFORMS Journal of Computing* 14(4):373–386.
- Dickerson JP, Sandholm T (2013) Throwing darts: Random sampling helps tree search when the number of short certificates is moderate. *Sixth Annual Symposium on Combinatorial Search*.
- Dinh N, Jeyakumar V (2014) Farkas’ lemma: three decades of generalizations for mathematical optimization. *TOP* 22(1):1–22, ISSN 1863-8279, URL <http://dx.doi.org/10.1007/s11750-014-0319-y>.
- Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. *Mathematical programming* 91(2):201–213.
- Farkas J (1902) Theorie der einfachen ungleichungen. *Journal für die reine und angewandte Mathematik* 124:1–27, URL <http://eudml.org/doc/149129>.
- Fischetti M, Lodi A (2010) Heuristics in mixed integer programming. *Wiley Encyclopedia of Operations Research and Management Science* .
- Gleixner A, Bastubbe M, Eiffler L, Gally T, Gamrath G, Gottwald RL, Hendel G, Hojny C, Koch T, Lübbecke ME, Maher SJ, Miltenberger M, Müller B, Pfetsch ME, Puchert C, Rehfeldt D, Schlösser F, Schubert C, Serrano F, Shinano Y, Viernickel JM, Walter M, Wegscheider F, Witt JT, Witzig J (2018) The SCIP Optimization Suite 6.0. Technical Report 18-26, ZIB, Takustr. 7, 14195 Berlin.

- Gould N, Scott J (2016) A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software (TOMS)* 43(2):15.
- Haralick RM, Elliott GL (1980) Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence* 14(3):263–313.
- Hendel G (????) IPET interactive performance evaluation tools. <https://github.com/GregorCH/ipet>.
- Khalil EB, Dilkina B, Nemhauser GL, Ahmed S, Shao Y (2017) Learning to Run Heuristics in Tree Search. *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 659–666, IJCAI’17 (AAAI Press).
- Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2011) MIPLIB 2010. *Mathematical Programming Computation* 3(2):103–163, URL <http://dx.doi.org/10.1007/s12532-011-0025-9>.
- Land AH, Doig AG (1960) An Automatic Method of Solving Discrete Programming Problems. *Econometrica* 28(3):497–520.
- Laundy R, Perregaard M, Tavares G, Tipi H, Vazacopoulos A (2009) Solving Hard Mixed-Integer Programming Problems with Xpress-MP: A MIPLIB 2003 Case Study. *INFORMS Journal on Computing* 21(2):304–313, URL <http://dx.doi.org/10.1287/ijoc.1080.0293>.
- Linderoth JT, Ralphs TK (2005) Noncommercial software for mixed-integer linear programming. *Integer programming: theory and practice* 3:253–303.
- Lodi A (2013) The heuristic (dark) side of MIP solvers. *Hybrid metaheuristics*, 273–284 (Springer).
- Lodi A, Tramontani A (2013) Performance variability in mixed-integer programming. *Theory Driven by Influential Applications*, 1–12 (INFORMS).
- McGill R, Tukey JW, Larsen WA (1978) Variations of box plots. *The American Statistician* 32(1):12–16.
- Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an efficient SAT solver. *Proceedings of the 38th annual Design Automation Conference*, 530–535 (ACM).
- Pólik I (2015) Some more ways to use dual information in MILP. *International Symposium on Mathematical Programming* (Pittsburgh, PA).
- Sandholm T, Shields R (2006) Nogood learning for mixed integer programming. *Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization, Montréal*.
- Witzig J, Berthold T, Heinz S (2017) Experiments with Conflict Analysis in Mixed Integer Programming. Salvagnin D, Lombardi M, eds., *Integration of AI and OR Techniques in Constraint Programming*, 211–220 (Cham: Springer International Publishing), ISBN 978-3-319-59776-8.