Zuse Institute Berlin

Takustrasse 7
D-14195 Berlin-Dahlem
Germany

GERALD GAMRATH, TIMO BERTHOLD, STEFAN HEINZ,
MICHAEL WINKLER

Structure-driven fix-and-propagate heuristics for mixed integer programming

ZIB Report 17-56 (October 2017)

Zuse Institute Berlin
Takustrasse 7
D-14195 Berlin-Dahlem

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

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

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

Structure-driven fix-and-propagate heuristics for mixed integer programming

Gerald Gamrath[*], Timo Berthold[†], Stefan Heinz[‡], Michael Winkler[§]

Abstract

Primal heuristics play an important role in the solving of mixed integer programs (MIPs). They often provide good feasible solutions early in the solving process and help to solve instances to optimality faster. In this paper, we present a scheme for primal start heuristics that can be executed without previous knowledge of an LP solution or a previously found integer feasible solution. It uses global structures available within MIP solvers to iteratively fix integer variables and propagate these fixings. Thereby, fixings are determined based on the predicted impact they have on the subsequent domain propagation. If sufficiently many variables can be fixed that way, the resulting problem is solved as an LP and the solution is rounded. If the rounded solution did not provide a feasible solution already, a sub-MIP is solved for the neighborhood defined by the variable fixings performed in the first phase. The global structures help to define a neighborhood that is with high probability significantly easier to process while (hopefully) still containing good feasible solutions. We present three primal heuristics that use this scheme based on different global structures. Our computational experiments on standard MIP test sets show that the proposed heuristics find solutions for about three out of five instances and therewith help to improve several performance measures for MIP solvers, including the primal integral and the average solving time.

Keywords: mixed-integer programming, primal heuristics, fix-and-propagate, large neighborhood search, domain propagation

Mathematics Subject Classification: 90C10, 90C11, 90C59

1 Introduction

Mixed integer linear programming problems (MIPs) minimize (or maximize) a linear objective function subject to linear constraints and integrality restrictions on a part of the variables. More formally, a MIP is stated as follows:

$$z_{MIP} = \min\{c^T x : Ax \leq b, \ell \leq x \leq u, x \in \mathbb{R}^n, x_i \in \mathbb{Z} \text{ for all } i \in \mathcal{I}\} \tag{1}$$

with objective function $c \in \mathbb{R}^n$, constraint matrix $A \in \mathbb{R}^{m \times n}$, and constraint right-hand sides $b \in \mathbb{R}^m$. We allow lower and upper bounds $\ell, u \in \bar{\mathbb{R}}^n$ on variables, where $\bar{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$, and the restriction of a subset of variables $\mathcal{I} \subseteq \mathcal{N} = \{1, \ldots, n\}$ to integral values. In the remainder

[*]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, gamrath@zib.de
[†]Fair Isaac Germany GmbH, c/o ZIB, Takustr. 7, 14195 Berlin, Germany, timoberthold@fico.com
[‡]Fair Isaac Germany GmbH, c/o ZIB, Takustr. 7, 14195 Berlin, Germany, stefanheinz@fico.com
[§]Gurobi GmbH, c/o ZIB, Takustr. 7, 14195 Berlin, Germany, winkler@gurobi.com

of this paper, we denote by $\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I})$ a MIP of form (1) in dependence of the provided data.

This modeling scheme allows to express many real-world optimization problems from various fields like production planning (Pochet and Wolsey, 2006), scheduling (Heinz et al., 2013), transportation (Borndörfer et al., 2010), or telecommunication networks (Lee and Lewis, 2006). On the other hand, the strict specifications for the problem statement make it possible to solve arising optimization problems for all these applications using the same algorithm. Therefore, very powerful generic solvers for MIPs have been developed over the last decades, which are used widely in research and practice (Lodi, 2010; Bixby, 2012; Achterberg and Wunderling, 2013).

These solvers are based on a branch-and-bound algorithm (Land and Doig, 1960; Dakin, 1965), which is accelerated by various extensions. The basic concept is to split the problem into sub-problems until they are easy enough to be solved. During this process, a lower bound is computed for each sub-problem by solving its linear programming (LP) relaxation $\mathcal{P}(c, A, b, \ell, u, \emptyset)$, that is the problem obtained from (1) by omitting the integrality restrictions. At the same time, the objective value of the incumbent—the best feasible solution found so far—provides an upper bound on the global optimum. In combination, these bounds allow to speed up the solving process by disregarding sub-problems whose lower bound exceeds the upper bound since those cannot lead to an improving solution.

It is evident that this algorithm profits directly from finding good solutions as early as possible. On the one hand, these solutions originate from integral LP relaxation solutions, on the other hand, so-called *primal heuristics* try to construct new feasible solutions or improve existing ones. Primal heuristics are incomplete methods without any success or quality guarantee which nevertheless are beneficial on average. There are different common approaches applied by many heuristics, e.g., rounding of the LP solution or diving, which iteratively changes the current sub-problem temporarily and solves the corresponding LP relaxation until an integral solution is obtained. For more details on primal heuristics, we refer to Berthold (2006); Fischetti and Lodi (2010); Berthold (2014a). In this paper, we introduce three novel heuristics which combine a fix-and-propagate scheme (Berthold and Gleixner, 2010; Achterberg and Wunderling, 2013) with the large neighborhood search (LNS) paradigm, cf. Danna et al. (2004). The former is typically used for before-LP heuristics and iteratively fixes a variable and propagates this change to apply all implied changes to the domains of other variables. The latter defines a sub-problem, the neighborhood, by adding restrictions to the problem, and then solves this sub-problem as a MIP. A more detailed discussion of these heuristic concepts is given in Section 2.

By modeling a specific problem as a MIP and solving it with a MIP solver, one profits from the decades of developments within this area. However, knowledge about the structure of the problem which could be exploited by a problem specific approach can hardly be fed into a MIP solver due to the generality of the algorithm. MIP solvers try to partially compensate this by detecting some common structures within the problem and exploiting them in the solving process. Examples for this are multi-commodity flow subproblems (Achterberg and Raack, 2010) and permutation structures (Salvagnin, 2014). This detection is often done in the presolving phase, which is a preprocessing step trying to remove redundancies from the model and tighten the formulation. An overview of different global structures in MIP solvers and details about three of them, the clique table, the variable bound graph, and the variable locks, are given in Section 3.

The heuristics presented in this paper use these global structures to apply a fix-and-propagate scheme. This means, they repeatedly fix variables and perform domain propagation to consider the direct consequences of these fixings on the domains of other variables. While this is a known approach in MIP heuristics (Berthold and Gleixner, 2010; Achterberg and Wunderling, 2013; Berthold and Hendel, 2015), our new heuristics take a step further and make domain propagation their driving force.

They use the global structures to predict the effects of domain propagation in the fixing phase and by this determine the fixing order and fixing values for the variables. After the fix-and-propagate phase and if the problem was reduced sufficiently, the remaining problem is then solved as an LP and the LP solution is rounded and tested for feasibility. If this did not provide a feasible solution already, the problem obtained after the fix-and-propagate phase is solved as a LNS sub-MIP.

A detailed description of the general scheme of the structure-driven fix-and-propagate heuristics is discussed in Section 4. The three instantiations of the heuristic scheme for the three discussed global structures are presented in Sections 5 to 7. The impact of the heuristics on the overall solving process is evaluated by the computational experiments presented in Section 8. Finally, Section 9 gives our conclusions and an outlook.

Previous work by the authors on this topic has been published in the conference paper Gamrath et al. (2015). It introduces previous versions of primal heuristics that are based on clique table and variable bound graph and gives a short computational evaluation of the heuristics. The present paper extends this work significantly. First, a third heuristic is introduced which is based on the variable locks. Second, we perform thorough computational studies in Sections 5.1, 6.1, and 7.1 to analyze the effort spent by the heuristics as well as their success in fixing variables based on the structure and constructing a feasible solution in the following. Additionally, also the two heuristics based on clique table and variable bound graph are significantly improved compared to the versions described in the previous paper. This includes a different fixing scheme employed by the clique-driven fix-and-propagate heuristic which directly works on the cliques rather than computing a clique partition. The variable-bound-driven fix-and-propagate heuristic now takes into account cliques when sorting the variable bound graph topologically (see Section 6) and was extended by two more fixing scheme variants. Finally, infeasible fixings in the fixing phase are now undone by a backtracking step and the fixing phase is continued. This allows to reach higher fixing rates, allowing to find more solutions by LP solving rather than the more expensive sub-MIP solve.

2 Primal heuristics and large neighborhood search for MIP

Primal heuristics are algorithms that try to find feasible solutions of good quality for a given optimization problem within a reasonably short amount of time. There is typically no guarantee that they will find any solution, let alone an optimal one.

For mixed integer linear programs (MIPs) it is well known that general-purpose primal heuristics like the Feasibility Pump (Achterberg and Berthold, 2007; Fischetti et al., 2005; Fischetti and Salvagnin, 2009) are able to find high-quality solutions for a wide range of problems. Over time, primal heuristics have become a substantial ingredient of state-of-the-art MIP solvers (Berthold, 2006; Bixby et al., 2000). Discovering good feasible solutions at an early stage of the MIP solving process has several advantages:

- The bounding step of the branch-and-bound (Land and Doig, 1960) algorithm depends on the quality of the incumbent solution; a better primal bound leads to more nodes being pruned and hence to smaller search trees.
- The same holds for certain presolving and domain propagation strategies such as reduced cost fixing. Better solutions can lead to tighter domain reductions, in particular more variable fixings. This, as a consequence, might lead to better dual bounds and the generation of stronger cutting planes.

- In practice, it is often sufficient to compute a heuristic solution whose objective value is within a certain quality threshold. For hard MIPs that cannot be solved to optimality within a reasonable amount of time, it might still be possible to generate good primal solutions quickly.
- Improvement heuristics such as RINS (Danna et al., 2004) or Local Branching (Fischetti and Lodi, 2003) need a feasible solution as starting point.

The last ten years have seen various publications on general-purpose heuristics for MIPs, including Achterberg and Berthold (2007); Achterberg et al. (2012); Andrade et al. (2017); Bergman et al. (2014); Bertacco et al. (2007); Berthold (2014b, 2017); Berthold et al. (2010, 2017); Dey et al. (2016); Fischetti and Monaci (2014); Fischetti and Salvagnin (2009); Gardi (2013); Ghosh (2007); Guastaroba et al. (2017); Guzelsoy et al. (2013); Hansen et al. (2006); Koc and Mehrotra (2017); Lazić (2016); Munguía et al. (2017); Rothberg (2007); Salvagnin (2014); Wallace (2010). For an overview, see Berthold (2006, 2014a); Lodi (2013).

Large neighborhood search lies at the heart of many MIP heuristics, such as Local Branching (Fischetti and Lodi, 2003), RINS (Danna et al., 2004), Crossover (Berthold, 2006), DINS (Ghosh, 2007), RENS (Berthold, 2014b), Proximity Search (Fischetti and Monaci, 2014), and Analytic Center Search Berthold et al. (2017). The main idea of LNS is to restrict the search for "good" solutions to a neighborhood centered at a particular reference point. This is typically the incumbent or another feasible solution, but it may as well be an infeasible integer point or a partial solution, see Fischetti and Lodi (2008). The hope is that the restricted search space makes the sub-problem much easier to solve, while still providing solutions of high quality. Of course, these restricted sub-problems do not have to be solved to optimality; they are mainly searched for an improving solution.

DINS, RINS, RENS, Crossover, and Analytic Center Search define their neighborhoods by variable fixings. LNS heuristics that are based on variable fixings suffer from an inherent conflict: the original search space should be significantly reduced; thus, it seems desirable to fix a large number of variables. At the same time, the more variables get fixed, the higher is the chance that the sub-problem does not contain any improving solution or even becomes infeasible.

The present paper addresses this issue by applying a fix-and-propagate scheme that is guided by global structures. The hope is that this scheme maintains feasibility of the restricted search space while still reducing it significantly through the means of domain propagation. The fix-and-propagate procedure can potentially find a complete assignment of the variables or end up with an empty search space; in either case the suggested heuristics will terminate. If neither is the case, a large neighborhood search will be conducted on the restricted search space.

3 Global structures in MIP solvers

Mixed integer programs are restricted to linear constraints, a linear objective, and integrality conditions. This makes MIP solvers easily accessible and exchangeable if a MIP model is at hand. From the modeling point of view, however, there is hardly any possibility to pass additional structural information to a solver, e.g., that and how certain model variables are connected via the combinatorics of a network structure. Nevertheless, there are certain global structures available within MIP solvers, starting from the data directly given in the model like the variable domains and the constraint matrix. Additionally, modern MIP solvers aim at detecting some more structures within a model and making use of them for heuristics, cutting plane separation or presolving, see e.g., Achterberg and Raack (2010); Salvagnin (2014).

Examples of global structures that are detected in presolving or during root node processing include the clique table, the implication graph, the variable bound graph, multi-commodity flow

structures, permutation structures, and symmetries. Multi-commodity flows and permutations are examples of rather specific constructs that occur in only a handful of models—but are crucial for solving them. Cliques and variable bound constraints, in contrast, can be found in many MIPs of different types. So far, they have been mainly used for cutting plane generation and domain propagation, see, e.g., Achterberg (2007b). The remainder of the section explains three global structures in more detail: the clique table and the variable bound graph, both detected during presolving, and the variable locks, which captures the presence of variables in the constraint matrix.

3.1 The clique table

A *clique* is a set $\mathcal{C}$ of binary variables of which at most one variable can be set to one. A clique can be given directly as a linear inequality $\sum_{i \in \mathcal{C}} x_i \leq 1$ or derived from more general constraints such as knapsacks: given a constraint $\sum_{i \in J} w_i x_i \leq C$ with binary variables $x_i, i \in J$, each subset $\mathcal{C} \subseteq J$ for which $w_j + w_k > C$ for all $(j, k) \in \mathcal{C} \times \mathcal{C}$ defines a clique. In addition, presolving techniques such as probing (Savelsbergh, 1994) can be used to detect cliques which are given implicitly and cannot be extracted directly from a single model constraint.

Similarly, *negated cliques* (Winkler, 2014) can be extracted from the problem. A negated clique is a set of binary variables of which at most one variable can be set to zero. However, for the ease of presentation, we transfer this back to the first case by introducing *negated variables* of the form $x'_i := 1 - x_i$. Negated variables are auxiliary variables and directly linked to the respective original variable, such that fixing a negated variable fixes the original one to the reverse value and vice versa. Thus, a negated clique is a clique on negated variables. Note that we also allow a mix of original and negated variables to be present in a clique. For the remainder of this article, we will not further discriminate between original and negated variables and assume cliques to be of the form given above.

In modern MIP solvers, the set of all detected cliques is stored in the so-called *clique table*. This global structure forms a relaxation of the MIP and is used by solver components, e.g., to create clique cuts (Johnson and Padberg, 1982) or to deduce stronger reductions in presolving and propagation (Savelsbergh, 1994). In Section 5, we will show how the clique table can be used to guide a fix-and-propagate heuristic.

3.2 The variable bound graph

Variable bound constraints are linear inequalities which contain exactly two variables. Typical examples for such constraints are precedence constraints on start time variables in scheduling or big-M constraints modeling fixed-costs in production planning. Depending on the sign of the coefficient, the variables bound each other. For example, a constraint $ax + by \geq c$ with $a > 0$ implies that x is bounded from below by $\frac{c}{a} - \frac{b}{a}y$. If $a < 0$, the latter provides an upper bound on x. These dependencies are called variable bound relations. They expresses the dependency of one bound of a variable on a bound of another variable. In order to avoid confusion with fixed bounds on variables, we will use the term *vbound* when referencing variable bound relations in the following.

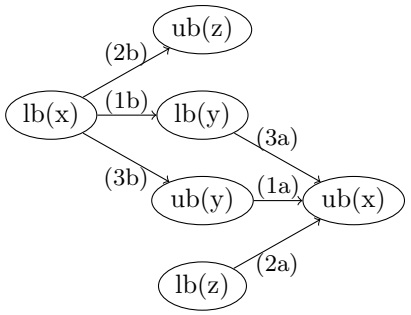
Similar to the clique information, vbounds cannot only be deduced from variable bound constraints, but can also be identified within more general constraints or during presolving, e.g., by probing. They are exploited by different solver components, e.g., for c-MIR cut separation, where they can be used to replace non-binary variables with binary ones (Marchand and Wolsey, 2001). In order to make vbounds available for those components, they can be stored in a global structure, the *variable bound graph*. In this directed graph, each node corresponds to the lower or

$$
\begin{aligned}
x - 2y &\leq 3 \quad (1)\\
x + 2z &\leq 2 \quad (2)\\
x + 3y &\leq 6 \quad (3)
\end{aligned}
$$

(a) constraint set.

$$
\begin{aligned}
x &\leq 2y + 3 && (1a)\\
y &\geq \tfrac{1}{2}x - \tfrac{3}{2} && (1b)\\
x &\leq 2 - 2z && (2a)\\
z &\leq 1 - 0.5x && (2b)\\
x &\leq 6 - 3y && (3a)\\
y &\leq 2 - \tfrac{1}{3}x && (3b)
\end{aligned}
$$

(b) vbounds.



(c) variable bound graph.

Figure 1: Example of a variable bound graph.

upper bound of a variable and each vbound is represented by an arc pointing from the influencing bound to the dependent bound. For an example of a variable bound graph, see Figure 1. We regard three constraints on variable x, y, and z, as shown in part (a). Each of these constraints provides two bounds on the involved variables as stated in part (b). Thereby, vbounds (1a) and (1b) are derived from constraint (1), (2a) and (2b) from (2), and (3a) and (3b) from (3). The resulting variable bound graph is illustrated in part (c). Each arc is labeled with the vbound it represents.

If a bound of a variable is tightened, implications can be read from this graph by following all paths starting at the corresponding node. Therefore, the graph can be used to compute an estimate of the impact that a bound change will have. This observation is the basis for the variable-bound-driven fix-and-propagate heuristic presented in Section 6.

3.3 Variable locks

In contrast to the two previous structures, variable locks are directly defined by the constraint matrix. They are a measure for how many constraints may forbid to increase or decrease the value of a variable. In case of a MIP of form (1) with only $\leq$-constraints, the number ζ_i^+ of *up-locks* of a variable x_i is the number of constraints in which this variable has a positive coefficient a_{ri}, while the number ζ_i^- of *down-locks* counts the number of constraints with negative coefficient of the variable. A more general definition for variable locks in constraint integer programming is given in Achterberg (2007b). In this paper, however, we focus on MIP and can thus stay with the simple definition above.

In the special case that a variable has no locks in one direction, its value in a solution can be increased without rendering a constraint infeasible. A simple rounding heuristic was introduced in Achterberg (2007b) which is based on this argument. On the other hand, duality fixing (Fügenschuh and Martin, 2005) fixes variables to their bound if they have no locks in that direction and the objective coefficient has the right sign. The variable-locks-driven fix-and-propagate heuristic presented in Section 7 uses the variable locks to decide which variable is most influential and to which value it should be fixed to retain feasibility.

4 A framework for structure-driven fix-and-propagate heuristics

In this section, we present a new primal heuristic scheme for mixed integer programming which is based on global structures collected by MIP solvers. We discuss the general idea, but do not go into detail how particular structures are used. The latter will be the topic of the following sections where we discuss instantiations of this scheme for the global structures introduced in Sec. 3 and evaluate their performance.

The general scheme is illustrated in Algorithm 1. In a first step, a subset of the integer variables is fixed based on the respective structure (lines 3–16). This is done in a fix-and-propagate manner (Achterberg and Wunderling, 2013; Berthold and Hendel, 2015), i.e., in each iteration, the global structure is used to decide on one variable to be fixed. This is done by the `structure_fixing` method called in line 5. This method is given the current problem as well as some global structure and returns the index k of a variable that should be fixed to one of its two bounds. The second return value specifies if the variable is supposed to be fixed to the lower or upper bound. Additionally, the method gives back a return value `result`, which is either `continue`, `solve LP`, or `stop`. The default return value is `continue`, which just continues the fix-and-propagate process. On the other hand, `stop` and `solve LP` will both stop the fixing phase, see line 6, and go directly to the LP solving starting in line 17. After the fixing is applied (line 7), domain propagation is performed (line 8). This uses a method `domain_propagation`, which performs domain propagation on the given MIP and returns the updated MIP as well as the information whether an infeasibility was detected during propagation. By default, we limit the domain propagation call to perform only two rounds of propagation to avoid performance issues. Nevertheless, this is usually enough to avoid trivial infeasibilities and apply implied bound changes on other variables—those contained in the global structure, but also other variables in the problem.

If domain propagation detects an infeasibility for the current assignment of variables, we backtrack one level, i.e., we undo the last fixing as well as the domain reductions deduced from it. Then, we remove the fixing value that led to the infeasibility from the domain of the respective variable and propagate this reduction (see lines 9–13). If the propagation detects infeasibility for this problem as well, one of the assignments we did before must cause the infeasibility (or the global problem is already infeasible). We do not backtrack several levels in this case in order to avoid too much effort being spent before finding the invalid assignment, but rather stop the heuristic immediately (line 14). If the updated problem is feasible, however, we continue with the fix-and-propagate procedure.

Note that the need to backtrack repeatedly indicates that the global structure used by the heuristics is missing essential components of the problem and directs the search to a wrong region. Therefore, Algorithm 1 is passed a limit κ on the number of backtracks performed. By default, we set $\kappa = 10$. If this number of backtracks is reached, the fix-and-propagate phase is stopped even if there are unfixed discrete variables left in the structure (line 16). Otherwise, the fix-and-propagate phase is iterated until all variables in the global structure were fixed.

After the fixing phase, we check its success. For this, we compare the number of fixed integer variables to the total number of integer variables (line 17). Ideally, the heuristic fixed all integer variables, but it may happen that some of the variables are not contained in the global structure employed for the fixing process, or that the fixing phase stopped prematurely due to the backtrack limit or the `result` value being `stop`. The heuristic does not require all integer variables to be fixed at that point yet. It solves an LP on the remaining problem and tries to round the LP solution with a simple rounding heuristic (Achterberg, 2007b), see lines 18–21. If enough integer variables were fixed before, this LP is significantly smaller (and hopefully easier

Algorithm 1: Generic structure-driven fix-and-propagate heuristic

input : - MIP $\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I})$
- fixing thresholds α, β
- backtrack limit κ
- global structure $\mathcal{S}$

output: - feasible solution or `NULL`, if no solution was found

begin
  `back` $\leftarrow 0$, `result` $\leftarrow$ `stop`
  `// 1. try to fix all integer variables in the structure`
  while $\{i \in \mathcal{I} \cap \mathcal{S} \mid \ell_i < u_i\} \neq \emptyset$ **do**
    $(\tilde{\ell}, \tilde{u}) \leftarrow (\ell, u)$
    `// get variable fixing based on global structure`
    (k, `lower`, `result`)$\leftarrow$ `structure_fixing`($\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I})$, $\mathcal{S}$)
    `// fix variable`
    if *result* $\neq$ *continue* **then break**
    if *lower* **then** $\tilde{u}_k \leftarrow \ell_k$ **else** $\tilde{\ell}_k \leftarrow u_k$
    `// perform domain propagation`
    $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I}),$ `inf`$) \leftarrow$ `domain_propagation`$(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I}))$
    `// infeasibility detected: backtrack and exclude infeasible value`
    if *inf* **then**
      `back` $\leftarrow$ `back` $+ 1$
      $(\tilde{\ell}, \tilde{u}) \leftarrow (\ell, u)$
      if *lower* **then** $\tilde{\ell}_k \leftarrow \ell_k + 1$ **else** $\tilde{u}_k \leftarrow u_k - 1$
      `// perform domain propagation`
      $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I}),$ `inf`$) \leftarrow$ `domain_propagation`$(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I}))$
      if *inf* **then return** *NULL*
    $\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I}) \leftarrow \mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I})$
    if *back* $\geq \kappa$ **then break**
  `// 2. LP solving`
  if $|\{i \in \mathcal{I} \mid \ell_i = u_i\}| \geq \alpha|\mathcal{I}| \vee$ *result* $=$ *solve LP* **then**
    $x^\star \leftarrow$ solve $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \emptyset)$
    `// try to round LP solution`
    $x^\star \leftarrow$ `simple_round`$(x^\star)$
    if $x^\star_i \in \mathbb{Z}$ *for all* $i \in \mathcal{I}$ **then**
      return $x^\star$
    else
      `// 3. LNS approach`
      $\mathcal{P}(\tilde{c}, \tilde{A}, \tilde{b}, \tilde{\ell}, \tilde{u}, \tilde{\mathcal{N}}, \tilde{\mathcal{I}}) \leftarrow$ presolve $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I})$
      if $|\tilde{\mathcal{N}}| \leq \beta|\mathcal{N}|$ **then**
        $x^\star \leftarrow$ solve $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{N}, \mathcal{I})$ (with working limits)
        return $x^\star$
  else
    return *NULL*

to solve) than the original LP relaxation. But not only the LP solving effort, also the success probability of the rounding heuristic depends on the success of the fixing phase. Therefore, the heuristic scheme demands that a fixing rate of at least α after the fixing phase is reached (line 17) and stops otherwise. The only exception is the case that the `result` value was `solve LP`. Then, the heuristic continues without checking the fixing rate.

If rounding the LP solution was not successful, the heuristic employs an LNS approach to construct a feasible solution to the neighborhood defined by the remaining unfixed variables. However, since this is typically the most expensive step of the algorithm, we first apply fast presolving methods to the LNS sub-MIP defined by the fixings obtained in the previous phase, see line 23. These methods remove fixed variables and redundant constraints, apply bound tightening and duality fixing, and perform aggregations of variables, amongst others. While we checked before that we fixed a sufficient number of discrete variables, we now require a reduction of β in the overall problem size in terms of variables (line 24). Since the processing time of branch-and-bound nodes in the sub-MIP depends mainly on the LP solving time, a sufficiently decreased problem size is a good indicator for reasonable LNS times. Finally, the sub-MIP is solved, see line 25, and the best feasible solution found during sub-MIP solving is returned (line 26).

In order to limit the effort spent within the heuristics, we use working limits for the sub-MIP solving. First, the fixing thresholds α and β ensure that the problem is significantly easier after the fixing phase. Second, we aim at performing a quick partial solve of the sub-MIP. Therefore, we disable separation in the LNS sub-MIP solving, use only fast presolving algorithms, and disable all LNS heuristics to avoid recursion. Additionally, we disable strong branching and use the inference branching rule of SCIP (Achterberg, 2007b). If a primal feasible solution was found already, we set an objective limit such that the solution is improved by at least 1 %. Finally, a node limit of 5000 is used together with a limit of 500 for the number of stalling nodes, i.e., consecutively processed nodes without finding a new best solution.

This general scheme is the same for all three heuristics proposed in this paper. The difference between them is how and in which order the variables are fixed in the first phase. This is defined by the global structures which represent interconnections between variables which can and will be propagated. The novel concept of the heuristics is that the order in which variables are fixed and the fixing values take into account the predicted impact a fixing will have on the domain propagation step. By this, domain propagation is not used as a supplementary subroutine to support the search, but as a driving mechanism to take decisions within the search: we choose fixings of which we know that they propagate well. How this is done for the three global structures is explained in the following sections.

5 The clique-driven fix-and-propagate heuristic

The idea behind the clique-driven fix-and-propagate heuristic is the following: Given a clique $\mathcal{C}$, at most one variable $x_i, i \in \mathcal{C}$ may be set to 1 in a feasible solution, all other variables need to be set to 0. Thus, by fixing one of the variables to 1, we force domain propagation to fix at least all other variables in the clique to 0. Consequently, it seems beneficial to choose large cliques in order to trigger many propagation changes. On the other hand, by choosing the cheapest variable, i.e., the variable with smallest objective coefficient as the one to fix to 1, we can aim at constructing solutions with a small objective value.

The fixing algorithm of the clique-driven fix-and-propagate heuristic is illustrated in Algorithm 2. In a first step, the next clique to process is selected (line 2). This is done with a greedy strategy: We select the clique with the largest number of unfixed variables. Note that

Algorithm 2: `clique_fixing`

input : - MIP $\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I})$
- clique table $\mathcal{T}$

output: - index of binary variable x_k which should be fixed next
- should x_k be fixed to 0?
- result of the call: `continue`

```
1  begin
       // 1.  select clique
2      C* ← arg max{|{x_i ∈ C | ℓ_i < u_i}| | C ∈ T}
       // best index and corresponding smallest objective
3      k* ← −1
4      c* ← ∞
       // 2.  find cheapest variable to fix
5      for j ∈ C* do
6          if u_{c_j} = 1 and c_j < c* then
7              k* ← j
8              c* ← c_j
9      return (k*, FALSE, continue)
```

the selection criterion implies that no variable contained in the clique is already fixed to 1 since propagation would have fixed all other variables to 0 otherwise. Then, we search for the unfixed variable with the smallest objective coefficient and return that this variable should be fixed to 1, see lines 3–9 How successful this fixing strategy is and which fixing thresholds should be used, cf. Algorithm 1, is analyzed in the computational experiments presented in the remainder of this section.

5.1 Computational analysis

Our computational experiments are based on an implementation of the clique-driven fix-and-propagate heuristic within the academic MIP solver SCIP 4.0.0 (Achterberg, 2007b; Maher et al., 2017) with SoPlex 3.0.0 (Wunderling, 1996; Maher et al., 2017) as underlying LP solver. We use a modified version of the heuristic as described in this paper, which will replace the old version in the next release of SCIP. It is implemented in a primal heuristic plugin of SCIP and called once at the beginning of the root node processing. Note that finding new incumbent solutions is often most effective at the root node, when a new primal bound might directly lead to global fixings, tighter cutting planes and better initial branching decisions. This is the typical application of fix-and-propagate heuristics, since the diversification of the heuristic search for calls during the subsequent branch-and-bound phase is smaller than for other heuristics that rely on local LP optima.

In our first experiments, we ran SCIP with the clique-driven fix-and-propagate heuristic for just the root node. This allows us to compare the heuristic runtime to the overall root processing time. Additionally, we set a time limit of 3600 seconds and a memory limit of 16 GB. All results were obtained on a cluster of 3.2 GHz Intel Xeon X5672 CPUs with 12 MB cache and 128 GB main memory, running only one job per cluster node at a time. The experiments were performed on the MMMC test set which contains all instances from the last three MIPLIB benchmark

subset	size	sols	root time	heur time	F&P time	LP time	LNS time
all	496	189	83.93	2.28	0.11	0.08	2.09
stopped early	226	–	91.49	0.12	0.12	–	–
LP solved	270	189	77.59	4.09	0.10	0.15	3.84
LNS applied	55	45	49.76	19.39	0.04	0.50	18.85

Table 1: Statistics for the clique-driven fix-and-propagate heuristic without fixing thresholds on the MMMC test set.

sets (Bixby et al., 1998; Achterberg et al., 2006; Koch et al., 2011) as well as the Cor@l test set (COR@L, 2014). We removed duplicates and the instances neos-1058477, neos-847051, and npmv07 because they caused numerical troubles. The numerical troubles do not depend on the usage of the heuristic; they can also be observed if it is disabled. This left us with a total of 496 instances.

For our first experiment, we set the fixing thresholds α and β to 0 in order to always continue with LP and LNS solving. The backtracking limit κ of 10 stayed unchanged. This experiment is meant to show the potential of the clique-driven fix-and-propagate heuristic and at the same time derive good default values for α and β.

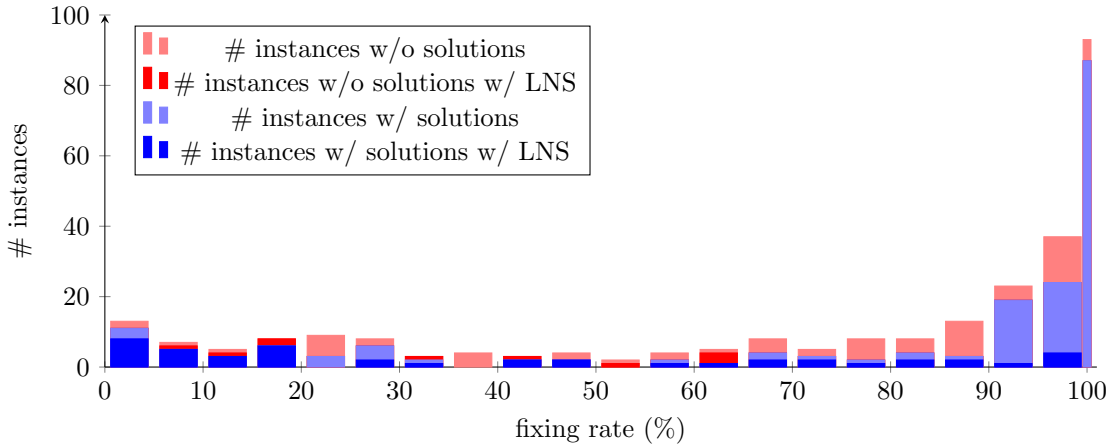
Table 1 gives first aggregated results for this experiment. The rows list information for different subsets of the instances. All instances (row 1), those stopped before the LP was solved (row 2), and those where the fix-and-propagate phase was successful and the heuristic solved at least the subsequent LP (row 3). The last row shows the instances where the LNS sub-MIP was solved. For each subset, we list its size, i.e., the number of instances in this category, the number of instances for which the heuristic found a solution, and the average root node processing time (including presolving). Additionally, we show the average running time of the heuristic, as well as the average times for the fix-and-propagate phase, the LP solving, and the LNS sub-MIP solving.

Out of the 496 instances, 102 instances contain no cliques or were solved during presolve. Another 124 instances ran into a dead-end, where fixing the best variable in the current clique to 0 or 1 both triggered domain propagation to detect infeasibility. These two cases together account for row "stopped early". On these instances, the clique-driven fix-and-propagate heuristic is fast and consumes only 0.13 % of the root node running time.

For the remaining 270 instances, the clique-driven fix-and-propagate heuristic solved the LP. In many cases, the LP solution could be rounded already, in some cases it also proved infeasibility. If neither of the two happened, the LNS sub-MIP was solved, which was the case for 55 instances. Overall, the heuristic was able to find a solution for 189 of the instances, which corresponds to a success rate of 70 % for the instances where the heuristic did not stop early. We can observe that the effort for the fix-and-propagate phase in relation to the total root processing time is slightly smaller for the instances where the fix-and-propagate phase was successful, as compared to those where it failed. The total time of the heuristic, however, increases significantly, since LP solving and in particular sub-MIP solving make out the majority of the effort spent within it. If it comes to solving a sub-MIP, this dominates the other steps and accounts for more than 97 % of the heuristic runing time. This is the main reason why the clique-driven fix-and-propagate heuristic without any fixing limits makes up for 2.7 % of the total root time.

In order to reduce the running time of the heuristic, the fixing thresholds α and β should be used to avoid too much time being spent in LP and sub-MIP solving. To this end, we investigate the impact of the fixing rates on both effort and success of the heuristic in the LP and sub-MIP

Figure 2: Number of instances and solutions found by the clique-driven fix-and-propagate heuristic per fixing rate. The blue part of each bar represents the instances for which the heuristic found a solution, the red part the instances where it was unsuccessful. The darker parts represent instances where a sub-MIP was solved, the light parts show instances where the LP solution was rounded to a feasible solution (light blue) or was infeasible (light red).



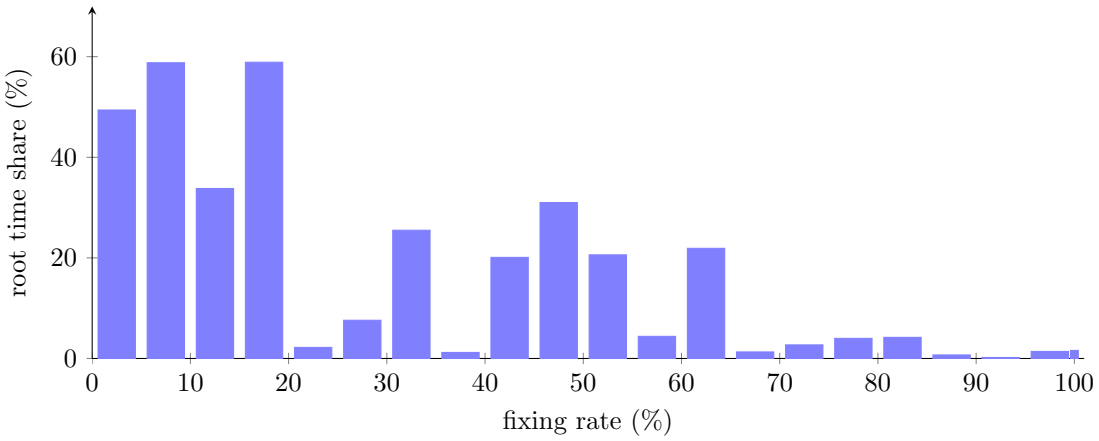
calls.

Figure 2 illustrates the fixing rates for the 270 instances for which at least an LP was solved. Each bar of the histogram shows the number of instances for which the integer fixing rate was within a certain range. More specifically, bar k represents all instances with fixing rate in $[5(k-1)\%, 5k\%)$, with a additional bar at 100 % for the case that all integer variables were fixed. Each bar is further divided into two parts by color: the blue part illustrates the instances where the clique-driven fix-and-propagate heuristic was successful in constructing a feasible solution while the red part shows instances where no solution was found by the heuristic. Each of these parts is further divided into two segments. The dark one represents instances where the heuristic solved the LNS sub-MIP. For the instances illustrated by the light part, the LP solution could be rounded to a feasible solution (light blue) or the LP was infeasible (light red).

What can be observed is that the success rate is high in particular at the two ends of the histogram. A total of 153 instances have a fixing rate of 90 % or higher, the success rate for these instances is 85 %. On many of these instances, the clique-driven fix-and-propagate heuristic fixes all of the variables, in that case the success rate goes up to 93.5 %. On the other hand, the success rate is also very high for the instances with a fixing rate of 20 % or lower: a solution is found for 75.8 % of the 33 instances in this range. This success rate is a consequence of the low fixing rate; by fixing only few variables, the chance to render the problem infeasible is reduced. On the other hand, the chance to find a solution by rounding the LP after the fixing phase is also small. Consequently, a sub-MIP solve is needed for 78.8 % of the instances while the LP solution can be rounded to a feasible solution for only 3 of the instances. However, solving a sub-MIP on a problem whose size is only slightly reduced compared to the original problem is not what LNS heuristics aim for, because the high effort for the sub-MIP could normally better be spent just continuing to solve the original problem.

The effort spent within the clique-driven fix-and-propagate heuristic for different fixing rates is illustrated in Figure 3. Again, each bar represents all instances with integer fixing rate within a certain 5 % range, with one additional bar for instances where all integer variable were fixed.

Figure 3: Average effort of the clique-driven fix-and-propagate heuristic per fixing rate, relative to total root processing time.



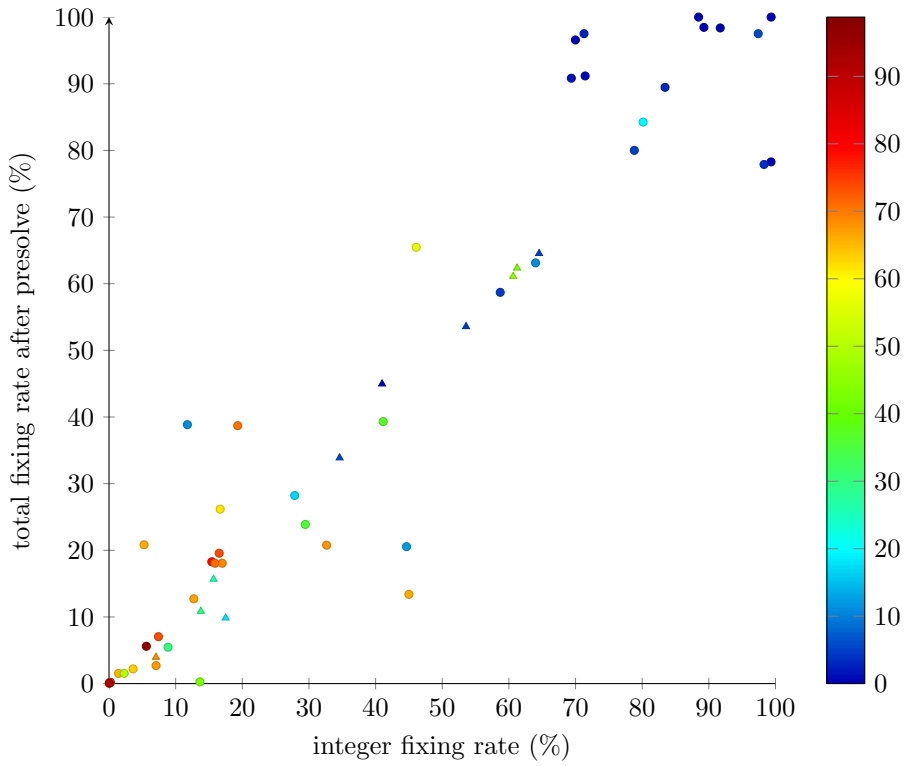
The height of the bars represents the average time spent in the heuristic compared to the total root (and preprocessing) time. The heuristic time counts into the root time, so that this share is between 0 and 100 for each instance. A value close to 100 % would mean that the preprocessing and root time without the clique-driven fix-and-propagate heuristic is negligible compared to the running time of the heuristic. On the other hand, a value of 50 % means that the heuristic needs about the same time as preprocessing and the remaining root procedures. The result for low fixing rates is as expected: For instances with fixing rate of 20 % or lower, the heuristic consumes 51.3 % of the aggregated root time. On the other hand, only 1.3 % of the aggregated root processing time is spent for the heuristic for instances with a fixing rate of 90 % or higher.

A reasonable choice seems to be to only continue with LP and (potentially) sub-MIP solving if a fixing rate of at least 65 % was obtained in the fixing phase. This slightly increases the number of solutions constructed by the heuristic to 146 as compared to 130 for a limit of 90 %. Many of the additional solutions are constructed via a sub-MIP, however, no unsuccessful sub-MIP calls can be observed. On the one hand, this can be seen as a success of the fixing scheme, on the other hand, the LP solving works very well here as a filter. It identifies all infeasible instances so that the effort stays relatively low and accounts for only 1.6 % of the root processing time.

Figure 4 takes into account the problem size reduction of the presolved sub-MIP as well, which can be limited by parameter β in Algorithm 1. Since this is only relevant for instances which solve the LNS sub-MIP, the scatter plot illustrates those 55 instances. Each instance is represented either by a circle if it found a solution, or by a triangle if it was not successful. The x-coordinate indicates the integer fixing rate after the fix-and-propagate phase, the y-coordinate represents the problem size reductions after presolve of the sub-MIP. Finally, the color shows the effort spent for solving the LP and sub-MIP, again relative to the total root node processing time.

As a first observation, we see that except for some outliers, the points are roughly located on the diagonal, meaning that a reduction in the number of integer variables often causes a similar reduction in the total problem size after presolve, as was to be expected. At the top right where both fixing rates are high, all sub-MIPs find a solution, and the joint time for LP and sub-MIP solving is below 5 % of the root processing time for each instance except for one where it amounts to about 20 %. Overall choosing the limit β on the fixing rate after presolve similar to the limit α of integer variables fixed in the fixing phase seems reasonable.

Figure 4: Clique-driven fix-and-propagate heuristic time for LP and sub-MIP solving (color) per fixing rate of integer variables (x-axis) and total fixing rate after sub-MIP presolving (y-axis). Each symbol represents one instance where the heuristic solved a sub-MIP. Circles illustrate instances for which the heuristic was successful, triangles represent unsuccessful instances.



This leads us to setting $\alpha = \beta = 65\,\%$ as default in the clique-driven fix-and-propagate heuristic for our following experiments as well as the version to be released. Table 2 lists the same information as Table 1, but for the heuristic with the updated limits. We see that the number of instances that were stopped before the LP solving increased from 226 to 301, this is caused by 75 instances that did not reach the desired fixing rate. The heuristic still finds 146 solutions. This are 43 solutions less than in the previous experiment; however, 31 of those had been constructed with the aid of an expensive sub-MIP. Overall, the number of sub-MIP calls is reduced from 55 to 14 with a 100 % success rate now whenever the heuristic solves a sub-MIP. Since also the neighborhoods to investigate are smaller due to the fixing limit, the average LNS time is significantly reduced from almost 19 seconds to less than half a second. On the 195 instances where an LP is solved, the success rate of the heuristic is now 74.9 % and the LP solving time is reduced by more than 70 %. All of this together leads to a significantly smaller runtime of the heuristic than in the previous experiment. On average over all instances, the heuristic needs about 0.2 % of the root processing time, which is a good value given a success rate of 29.4 %.

subset	size	sols	root time	heur time	F&P time	LP time	LNS time
all	496	146	81.80	0.13	0.11	0.01	0.01
stopped early	301	–	76.93	0.09	0.09	–	–
LP solved	195	146	89.31	0.19	0.12	0.04	0.03
LNS applied	14	14	82.21	0.55	0.06	0.03	0.46

Table 2: Statistics for the clique-driven fix-and-propagate heuristic with final fixing thresholds $\alpha = \beta = 65\,\%$ on the MMMC test set.

6 The variable-bound-driven fix-and-propagate heuristic

In the variable-bound-driven fix-and-propagate heuristic, we implemented different rules for determining the variable fixings. All of them make use of an (almost) topological sorting of the variable bound graph. Recall that a topological sorting of an acyclic directed graph is an order of the nodes, such that for every arc (i, j) node i precedes node j in the order. Since the variable bound graph can contain cycles, we may need to break them by randomly removing one of the arcs in the cycle. We call a topological sorting of this reduced graph *almost topological* and use this sorting to define the order in which variables are fixed.

Note that each clique $\mathcal{C}$ represents a set of vbounds as well: for each pair of variables $x_i, x_j \in \mathcal{C}$, the two vbounds $x_i \leq 1 - x_j$ and $x_j \leq 1 - x_i$ are implied, which correspond to the arcs (lb(x_j), ub(x_i)) and (lb(x_i), ub(x_j)) in the variable bound graph, respectively. We take those implied vbounds into account when computing the topological sorting. Special care has to be taken to avoid unnecessarily high runtimes for the sorting process. This is done with a simple depth-first search, which has linear effort in the number of nodes and arcs. However, the number of edges represented by a single clique is quadratic in the number of variables contained in the clique, so adding all of them explicitly may significantly increase the sorting time if large cliques are present in the problem. We use an improvement suggested in Achterberg and Wunderling (2013) for a similar application: we observe that each clique needs to be considered only twice. The first time when we are regarding a node $lb(x_i)$, $x_i \in \mathcal{C}$, we examine all arcs (lb(x_i), ub(x_j)) with $x_j \in \mathcal{C}, i \neq j$. Now, when regarding the next node $lb(x_j)$, $x_j \in \mathcal{C}$, we only need to consider the arc (lb(x_j), ub(x_i)), all other arcs point to nodes which have already been processed. In further examinations of nodes $lb(x_k)$, $x_k \in \mathcal{C}$, the clique can be disregarded by the same argument. Thus, the variable-bound-driven fix-and-propagate heuristic does also use information stored in the clique table as the clique-driven heuristic does, but it only uses it to refine the topological sorting of the variable bound graph. The final fixing scheme as summarized in Algorithm 3 is very different to the one of the clique-driven fix-and-propagate heuristic. As before, Algorithm 3 is a sub-algorithm of Algorithm 1 and called in each iteration of the fix-and-propagate phase to determine a variable to fix and the corresponding fixing value.

The fixing method starts with an initialization step (lines 2–4). The method *topological_sort* called for this purpose returns an array of nodes in almost topological order. The method already sorts out disconneced nodes, nodes corresponding to continuous variables and nodes from which only nodes corresponding to continuous variables can be reached. Additionally, the set $\mathcal{U}$ of unprocessed nodes is initialized to all nodes in the order.

Then, the first unprocessed node in the topological order is selected. If it corresponds to an already fixed variable, e.g. by a previous iteration of the fixing phase, it is ignored and the next variable is selected (lines 9–11). Recall that each node v of the variable bound graph represents a bound of a variable. Tightening this bound causes some bound changes on other variables, as

Algorithm 3: `variable_bound_fixing`

input : - MIP $\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I})$
- variable bound graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$
- clique table $\mathcal{T}$
- Bool: `tighten` – should as much change be caused as possible?
- int: `obj` – should the objective function be taken into account?

output: - index of binary variable x_k which should be fixed next
- should x_k be fixed to its lower bound (otherwise: upper bound)?
- result of the call: `continue` or `stop` fixing

begin
  `// initialization`
  if *1st call* **then**
    `// topological sorting; the order does not contain independent nodes, nodes which correspond to or only influence continuous variables`
    $\pi \leftarrow$ topological_sort$(\mathcal{G}, \mathcal{T})$;
    $\mathcal{U} \leftarrow \mathcal{V} \cap \pi$;
  $k \leftarrow 0$;
  while $k = 0$ **do**
    `// select first unprocessed node`
    $k \leftarrow \min\{1 \leq i \leq |\pi| \mid \pi_i \in \mathcal{U}\}$;
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\pi_k\}$;
    `// discard fixed variables`
    if $\ell_{idx(\pi_k)} = u_{idx(\pi_k)}$ **then**
      $k \leftarrow 0$;
      continue ;
    `// fixing value`
    if *tighten* **then**
      fixtolower $\leftarrow$ not lower(π_k);
    else
      fixtolower $\leftarrow$ lower(π_k);
    `// consider objective function`
    if *obj = 1* **then**
      `// do not fix to worse bound w.r.t. objective`
      if *fixtolower* $\neq$ *(*$c_{idx(\pi_k)} \geq 0$*)* **then**
        $k \leftarrow 0$;
    else if *obj = 2* **then**
      `// do not fix to better bound w.r.t. objective`
      if *fixtolower* $\neq$ *(*$c_{idx(\pi_k)} \leq 0$*)* **then**
        $k \leftarrow 0$;
  if $k > 0$ **then**
    return *(*$idx(\pi_k)$*, fixtolower,* `continue`*)*;
  else
    return *(0,FALSE,*`stop`*)*;

defined by all paths in the variable bound graph starting at node v. Consequently, the earlier a node is considered within the almost topological order, the more impact on other bounds we expect when tightening the corresponding bound.

The first variant by which the variable-bound-driven fix-and-propagate heuristic determines fixings aims at obtaining a large neighborhood by fixing variables such that only few additional restrictions are caused. This results in a neighborhood with a higher probability both for containing feasible solutions as well as high-quality solutions. To this end, this variant fixes the variable to the bound represented by the current node. This means that not the bound corresponding to the current node is tightened, but the opposite bound, which comes later in the topological order (if even) and thus causes fewer reductions on other bounds. The second variant uses an opposing argument: A large neighborhood is more expensive to process and finding any solutions in there might need more effort than in a smaller neighborhood with more fixed variables. Therefore, we fix the variable to the reverse bound, i.e., tighten the bound corresponding to the node in the variable bound graph. This forces change on many other bounds of variables, a concept known to be rather effective in order to drive the solution to feasibility faster, cf. Pryor and Chinneck (2011). The parameter `tighten` is used to switch between the two variants in Algorithm 3, lines 12–15. Thereby, a value of `FALSE` corresponds to the first variant, `TRUE` to the second.

We obtain two variations for each of the previously mentioned variants by taking into account the objective function (triggered by setting `obj` to 1 or 2 in Algorithm 3, lines 16–21, instead of the default value 0). For this, we need the notion of the best bound of a variable, which is the bound that leads to the best objective contribution of the variable, i.e., its lower bound if its objective value is non-negative, and its upper bound otherwise. If `obj` is set to 1, fixings are only applied if the variable is fixed to its best bound. Conversely, for `obj` = 2, fixings are only applied if the variable is not fixed to its best bound. The motivation for the first variation is clearly to aim at obtaining high-quality solutions. Variation 2 is based on the observation that naturally, the constraints of the problem typically push variables away from their best bound, while fixing them to their worst bound might give a higher chance for a feasible solution in the end. These variants may keep variables in the variable bound graph unfixed. If this is the case, the fixing algorithm returns `stop` as a `result` to signal that no further fixings will be performed by the current scheme (line 25).

Overall, this gives us six fixing schemes for the variable-bound-driven fix-and-propagate heuristic. We will investigate in the following how well they complement each other or if any of them is dominated by the others.

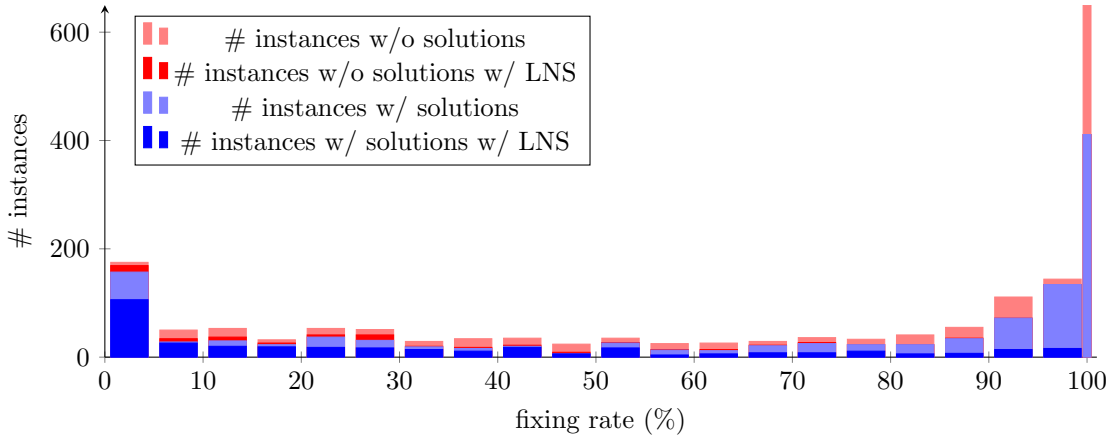
6.1 Computational analysis

Our computational experiments are performed in the same environment, with the same software, and on the same instance set as described in Section 5.1. Again, the updated version of the variable-bound-driven fix-and-propagate heuristic is called once at the start of root node processing as motivated in Section 5.1.

In a first experiment, we ran all six variants without any interaction between them in order to assess their individual performance. More specifically, all variants were run one after the other, without applying primal solutions found by one of the variants during the process. This avoids that the behavior of the subsequent variants is changed, e.g., because only better solutions would be accepted, and allows to compare the number of solutions found by each variant as well as their quality in a fair way.

The results of a first experiment with fixing thresholds α and β set to 0 are summarized in Table 3 as well as Figures 5 and 6. They display the same information as Table 1 and Figures 2 and 3, respectively. For a fair comparison, we are not aggregating the results of the six variants

Figure 5: Number of instances and solutions found by the variable-bound-driven fix-and-propagate heuristic per fixing rate. The blue part of each bar represents the instances for which the heuristic found a solution, the red part the instances where it was unsuccessful. The darker parts represent instances where a sub-MIP was solved, the light parts show instances where the LP solution was rounded to a feasible solution (light blue) or was infeasible (light red).



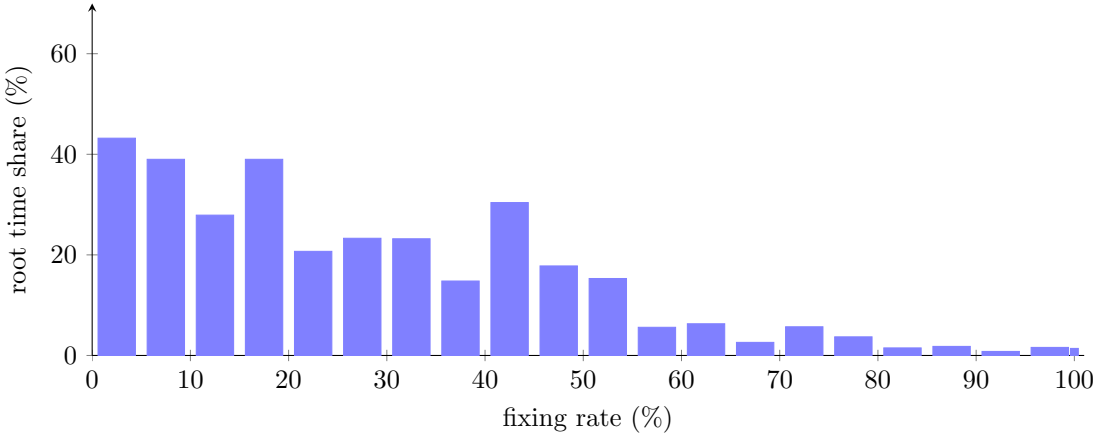
of the variable-bound-driven fix-and-propagate heuristic, because this would naturally lead to higher success rates. Instead, we are counting each (potential) call of each of the six variants, which increases the number of observations from 496 to 2976. Note that we are not counting the time of all six variants into the root node processing time when looking at a single variant, but reduce the root time by the time spent in the other five variants.

We see that the variable-bound-driven fix-and-propagate heuristic is stopped early due to missing structure or infeasible assignments in 42.2 % of the calls, while having a success rate of 67.5 % on the other instances. These numbers are similar to those of the clique-driven heuristic, as is the running time of an average heuristic call in the former case. A main difference that can be observed is that the average time for a heuristic call is much higher. It is increased by almost an order of magnitude, which can be attributed to similar increases in the average LP and LNS time. A closer look at the results for the individual instances revealed that this is mainly caused by a few instances for which the LNS process consumed almost an hour of runtime, while normally, the root node processing was finished within seconds. The arithmetic mean is particularly prone to such changes which result in very large numbers an few instances.

subset	calls	sols	root time	heur time	F&P time	LP time	LNS time
all	2976	1161	94.44	19.38	0.23	0.99	18.17
stopped early	1255	–	69.09	0.12	0.12	–	–
LP solved	1721	1161	112.93	33.43	0.31	1.70	31.42
LNS applied	410	352	203.22	136.09	0.18	4.02	131.89

Table 3: Statistics for the six variants of the variable-bound-driven fix-and-propagate heuristic without fixing thresholds on the MMMC test set (496 instances). Each call of a variant is counted individually, resulting in 2976 observations.

Figure 6: Average effort of the variable-bound-driven fix-and-propagate heuristic per fixing rate, relative to total root processing time.



Figures 5 and 6 second that this change is mainly due to some outliers. The fixing rates, as shown in Figure 5 have a similar distribution as for the clique-driven heuristic, however, there are 10.2 % of the heuristic calls with a fixing rate α of less than 5 %, as opposed to 4.8 % for the clique-driven heuristic. The figures suggest that a fixing threshold $\alpha = 65\,\%$ is again reasonable. This way, we filter out 36.1 % of the heuristic calls with an average running time of 29.0 % of the root node processing time, see Figure 6. The remaining 1099 heuristic calls have a success rate of 67.6 % with an average running time of 1.6 % of the root node processing time. Only 74 sub-MIPs are solved which construct 70 feasible solutions. Thus, we will use fixing thresholds $\alpha = \beta = 65\,\%$ for the variable-bound-driven fix-and-propagate heuristic in the following.

Some of the solutions in the previous experiment have been constructed by different variants on the same instance. This is investigated in Table 4. For each of the six variants (defined by the first two columns which present the values of the two parameters `tighten` and `obj`) the table shows its success in finding primal solutions. Thereby, we are only counting heuristic calls that reached the fixing rate limits $\alpha = \beta = 65\,\%$. We present four different statistics. Column "sols" lists the total number of solutions found by the variant, while column "best sols" displays the number of instances where the variant found a solution with best objective value among the variants. This includes cases where multiple variants found solutions with the same

`tighten`	`obj`	sols	best sols	single best sols	single sols
FALSE	0	113	46	2	1
FALSE	1	112	54	25	3
FALSE	2	139	78	35	7
TRUE	0	147	84	17	2
TRUE	1	90	59	26	6
TRUE	2	142	72	24	12

Table 4: Comparison of the six variants of the variable-bound-driven fix-and-propagate heuristic with fixing thresholds $\alpha = \beta = 65\,\%$ on the MMMC test set (496 instances).

subset	size	sols	root time	heur time	F&P time	LP time	LNS time
all	496	172	79.26	1.74	1.06	0.11	0.57
stopped early	278	–	75.99	0.04	0.04	–	–
LP solved	218	172	86.74	3.91	2.36	0.26	1.30
LNS applied	53	50	102.42	6.17	0.45	0.39	5.33

Table 5: Statistics for the variable-bound-driven fix-and-propagate heuristic with final fixing thresholds $\alpha = \beta = 65\,\%$ on the MMMC test set. For each instance, the calls of the five variants enabled by default are merged, giving one result. Thereby, an instance is counted for one of the last two rows if at least one variant solved an LP or performed an LNS search.

best objective value, while column "single best sols" only includes instances where the solution was strictly better than the solutions found by all other variants. Finally, column "single sols" summarizes the number of instances for which the respective variant was the only one able to construct a feasible primal solution.

The three variants (`tighten`=`FALSE`, `obj`=2), (`tighten`=`TRUE`, `obj`=0), and (`tighten`=`TRUE`, `obj`=2), which find the highest number of solutions also find many best solutions. Variant (`tighten`=`TRUE`, `obj`=1) is ranked worst with respect to the number of solutions found, but taking into account the objective during fixing proves benificial for finding better solutions than the other variants: it is ranked second for the number of strictly best solutions. Overall, (`tighten`=`FALSE`, `obj`=0) performs significantly worst. Although finding the fourth-highest number of solutions, it is ranked last for all other criterions. For all but two instances, there is at least one other variant which finds a solution of equal or better quality. For all other variants, there are at least 17 instances where the variant finds the single best solution and often several instances where no other variant finds a solution.

These results, together with the relatively small effort for the call of a single variant led to the following default settings for the heuristic. Only variant (`tighten`=`FALSE`, `obj`=0) is disabled by default, all five other variants are called sequentially in each call of the variable-bound-driven fix-and-propagate heuristic.

The performance of the final version of the variable-bound-driven fix-and-propagate heuristic is presented in Table 5. As opposed to Table 3, we now list the results of each call of the heuristic, which includes the sub-calls of the five variants enabled by default. Thereby, improving primal solutions found by one variant are directly applied, which may influence the variants executed later. Additionally, the heuristic is not run on instances for which the variable bound graph spans only such a small fraction of the variables that reaching the fixing rate is very unlikely. As a consequence, the heuristic reaches the LP solving and rounding step for only 44 % of the instances. Note that an instance is counted for having solved an LP or LNS sub-MIP if at least one of the variants did so on this instance. The latter case happens for 53 instances, for 50 of those, a feasible primal solution can be constructed. Overall, the variable-bound-driven fix-and-propagate heuristic consumes about 2.2 % of the root node processing time. The effort is negligible for instances where it is stopped before the LP solving, while for the remaining instances, it increases to 4.51 %. This seems still reasonable given the success rate of 78.9 % on this set of instances.

7 The variable-locks-driven fix-and-propagate heuristic

The variable-locks-driven fix-and-propagate heuristic works on a structure which is present in each MIP: the constraint matrix, and in particular the variable locks, see Sect. 3.3. Other than cliques and variable bound graph which represent only a part of the original constraints, but also cover relations implicitly given in the problem and detected during presolving, the locks take into account all given constraints.

The heuristic is motivated by greedy heuristics for set covering problems: Starting with an all-zero solution, one selects one variable which is contained in the highest number of constraints and fixes it to 1. By this, all these set covering constraints are fulfilled independent of the other variables' values. In subsequent steps, a variable is selected which is contained in the highest number of not-yet fulfilled constraints.

How the variable-locks-driven fix-and-propagate heuristic translates this approach to general MIP is shown in Algorith 4. In each iteration of the fixing process, a "high-impact" binary variable is selected, where the impact of a variable is decided based on the sum of its up- and down-locks, which corresponds to the number of constraint it is part of, cf. line 5. Then, the given variable is fixed to the bound where it has the smaller number of locks, see lines 8 to 12. This aims at reaching feasibility fast and possibly ensuring that some constraints are already fulfilled after a few fixings, no matter how the values of the remaining variables in the constraint will be chosen—as long as they are within their updated bounds. If a variable has the same number of up- and down locks, we use a randomized approach to determine its fixing value. The

Algorithm 4: `locks_fixing`

input : - MIP $\mathcal{P}(c, A, b, \ell, u, \mathcal{N}, \mathcal{I})$

output: - index of binary variable x_k which should be fixed next; or -1 if no further fixings should be performed
- should x_k be fixed to it 0 (otherwise: 1)?
- result of the call: `continue` or directly `solve LP`

1 **begin**

`// get variable locks`

2 **for** $i \in \mathcal{B}$ **do**

3 $\quad \zeta_i^+ \leftarrow |\{r \in [1, \ldots, m] : a_{ri} > 0 \wedge \max\{A_{r\cdot}x | \ell \leq x \leq u\} > b_r\}|$

4 $\quad \zeta_i^- \leftarrow |\{r \in [1, \ldots, m] : a_{ri} < 0 \wedge \max\{A_{r\cdot}x | \ell \leq x \leq u\} > b_r\}|$

`// select variable with highest sum of up- and down-locks`

5 $k \leftarrow \arg\max_{i \in \mathcal{B}}\{\zeta_i^+ + \zeta_i^-\}$

`// no variable with locks left, stop fixing phase`

6 **if** $\zeta_k^+ = \zeta_k^- = 0$ **then**

7 $\quad$ **return** *(0,FALSE,solve LP)*

`// fix variable to bound where it has fewer locks`

8 **if** $\zeta_k^+ > \zeta_k^-$ **then**

9 $\quad$ **return** *(k,TRUE,continue)*

10 **else if** $\zeta_k^+ = \zeta_k^-$ **then**

11 $\quad$ **return** *(k,TRUE,continue)* with probability 33 %

12 **return** *(k,FALSE,continue)*

subset	size	sols	root time	heur time	F&P time	LP time	LNS time
all	496	253	78.46	0.47	0.22	0.22	0.03
stopped early	210	–	68.82	0.24	0.24	–	–
LP solved	286	253	85.54	0.64	0.20	0.39	0.05
MIP solved	9	9	417.14	3.48	0.90	1.06	1.51

Table 6: Statistics for the variable-locks-driven fix-and-propagate heuristic without fixing thresholds on the MMMC test set.

variable is then fixed to 1 with a probability of 67 %, see line 11. We chose this probability based on a preliminary experiment where is showed a good performance.

If a constraint is already fulfilled, the locks it contributes to the contained variables are disregarded (see lines 2–4), so that the impact as well as the fixing direction are always determined with respect to the not-yet fulfilled constraints only. Therefore, it may happen that all constraints are fulfilled already and none of the remaining variables has any locks left. In this case, we stop the fixing procedure and return that the LP should be solved directly in order to determine optimal values for the remaining binary variables, cf. lines 6–7.

7.1 Computational analysis

For the computational analysis of the variable-locks-driven fix-and-propagate heuristic, we used the same environment, the same software, and the same instance set as described in Section 5.1. As the previously presented heuristics, the heuristic is called once at the start of root node processing.

We performed the same experiments as described in Section 5.1 to derive good fixing thresholds α and β. The results are summarized in Table 6 and Figures 7 and 8.

The variable-locks-driven fix-and-propagate heuristic is different to the other two heuristics presented in this paper in two aspects. First, it does not necessarily aim at fixing as many binary variables as possible. It constantly monitors how many of the constraints became redundant with respect to the tightened domains and stops the fixing phase as soon as all constraints became redundant. The values for all remaining variables can easily be determined by the subsequent LP solve. This is also taken into account in the general framework for structure based heuristics, where the fixing rate does not need to be checked in that case (see Algorithm 1, line 17). For better comparability, such instances will be counted as having a fixing rate of 100 % in the following, since the remaining problem can be optimized as an LP.

On the other hand, the variable-locks-driven fix-and-propagate heuristic is based on a structure which covers the whole problem, in particular all binary variables. This is different to clique table and variable bound graph which often cover only a part of the binary variables (at least when disregarding trivial cliques containing only one variable as the clique-driven heuristic does). This means the the heuristic can always specify a fixing value for all binary variables unlike clique- and variable-bound-driven fix-and-propagate heuristic which will just stop the fixing phase as soon as all variables in the respective structure were fixed. As a consequence, the heuristic typically fixes a high number of variables or fails repeatedly while trying to do so. This leads to 210 instances where the heuristic is stopped before the LP solving. For the remaining 286 instances, the heuristic successfully constructs a feasible primal solution in 88.5 % of the cases. To do so, it performs the LNS step for only 9 instances. This small number is a direct consequence of the effectiveness of domain propagation and obtaining high fixing rates, which

Figure 7: Number of instances and solutions found by the variable-locks-driven fix-and-propagate heuristic per fixing rate. The blue part of each bar represents the instances for which the heuristic found a solution, the red part the instances where it was unsuccessful. The darker parts represent instances where a sub-MIP was solved, the light parts show instances where the LP solution was rounded to a feasible solution (light blue) or was infeasible (light red).

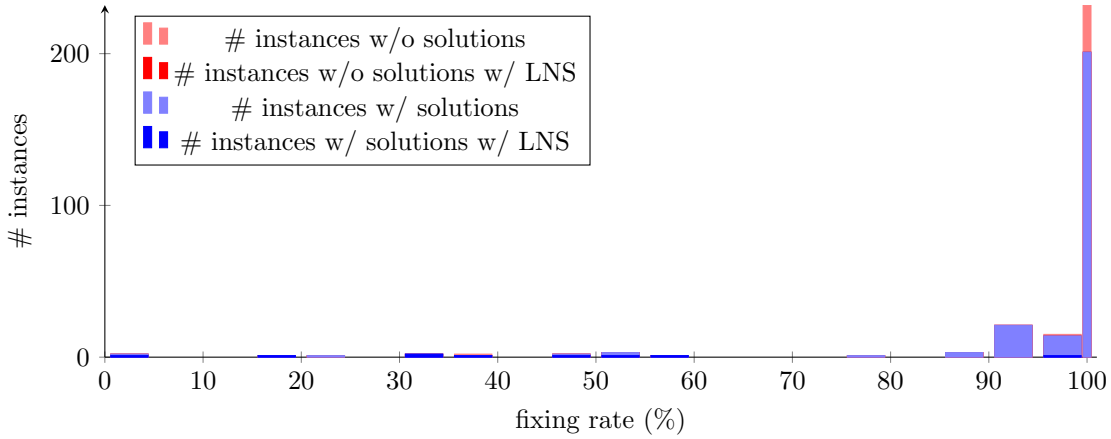
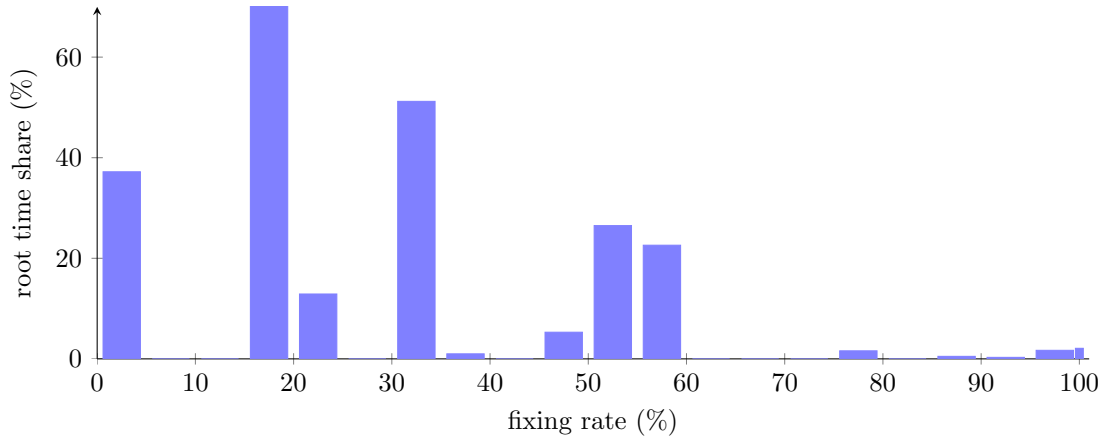


Figure 8: Average effort of the variable-locks-driven fix-and-propagate heuristic per fixing rate, relative to total root processing time.



make it more probable to be successful in rounding the LP solution.

The fixing rates are illustrated in Figure 7. It shows that 232 of the 286 instances for which at least an LP was solved have a fixing rate of 100 %. On the other hand, only 18 instances which solve an LP have a fixing rate of less than 90 %, 8 of them perform an LNS search.

Figure 8 shows the effort of the variable-locks-driven fix-and-propagate heuristic is very small for fixing rates larger than 60 %. Note that we cannot really take a conclusion for fixing rates between 60 % and 75 %, since this case does not happen in our experiment. What we can say, though, is that for instances with a fixing rate of less than 60 %, the average running time of the heuristic accounts for 28.3 % of the root processing time, while for higher fixing rate this is reduced to 1.9 %.

As a consequence, and for consistency reasons, we set the fixing thresholds α and β to 65 % by default. The final statistics for variable-locks-driven fix-and-propagate heuristic are summarized in Table 7. The heuristic finds 240 solutions, which results in a success rate of 48.4 % over the complete test set and 88.2 % for the subset of instances where the heuristic is not stopped early. It only solves a single LNS sub-MIP and is reasonably fast, accounting for about 0.5 % of the average root processing time

8 Computational results

In the previous sections, we did a local analysis of the newly proposed heuristics. This means that we evaluated each heuristic individually, focussing on its success in terms of solutions being found. Additionally, we investigated their runtimes and which part of the heuristic algorithm consumed how much time. In this section, we investigate the impact of the structure-driven fix-and-propagate heuristics on the overall solving behavior of SCIP and evaluate how well they can be combined. For this, we use SCIP version 4.0 with default settings except for the settings enabling the three investigated heuristics.

Before we come to the overall performance, let us shortly look at how well the heuristics complement each other. Table 8 presents the same information as Table 4, now for the three final heuristics. That is, it lists for each heuristic the number of solutions being found and the number of instances where it found the best solution among the three heuristics (possibly tied). The last two columns show the number of instances where a heuristic found the strictly best solution and the only solution among the three heuristics, respectively. We see that the variable-locks-driven fix-and-propagate heuristic (row "locks") finds a solution for 48.4 % of the instances, while clique-

subset	calls	sols	root time	heur time	F&P time	LP time	LNS time
all	496	240	78.15	0.42	0.22	0.20	0.00
stopped early	224	–	81.04	0.25	0.25	–	–
LP solved	272	240	75.76	0.56	0.19	0.36	0.00
LNS applied	1	1	24.70	1.88	1.15	0.05	0.69

Table 7: Statistics for the variable-locks-driven fix-and-propagate heuristic with final fixing thresholds $\alpha = \beta = 65\,\%$ on the MMMC test set.

heuristic	sols	best sols	strictly best sols	single sols
clique	146	66	53	9
locks	240	140	115	94
vbound	171	118	92	27

Table 8: Number of solutions found by the three structure-driven heuristics on the MMMC test set (496 instances). We list the number of solutions found by the heuristic, the number of instances where it found the best solution among the three heuristics (but other possibly found a solution with same value), the number of instances where it found the strictly best solution among the heuristics, and the number of instances where it was the only of the three heuristics able to construct a feasible solution.

		worse solution		
		clique	locks	vbound
better	clique	–	94	68
	locks	155	–	130
	vbound	117	127	–

(a) instances where at least one heuristic found a solution

		worse solution		
		clique	locks	vbound
better	clique	–	71	43
	locks	38	–	20
	vbound	67	86	–

(b) instances where both heuristics found a solution

Table 9: Pairwise comparison of the structure-driven fix-and-propagate heuristics, showing the number of instances where one heuristic found a better solution than the other.

and variable-bound-driven fix-and-propagate heuristic (rows "clique" and "vbound"), which both depend on a more specific structure in the problem, are only able to generate solutions for 29.4 % and 34.5 % of the instances, respectively. About half of the solutions being found are not dominated by a solution found by one of the other two heuristics. This share is highest for the variable-bound-driven heuristic for which 69.0 % of its solutions are not dominated, clearly a result of running five different fixing variants within the heuristic. The variable-locks-driven fix-and-propagate heuristic finds solutions for 94 instances where both other heuristics lack the needed structure or are unsuccessfull for another reason. This makes it have the highest number of strictly best solutions among the heuristics, as well. However, if also one of the other heuristics finds a solution on an instance, the solution found by the variable-locks-driven heuristic is in most cases dominated with respect to its objective value.

This gets more clear in Table 9, where we do a pairwise comparison of the heuristics with respect to the objective value of solutions being constructed. On the left side, all instances are regarded where at least one of the two heuristics found a solution, not counting instances where both found a solution of equal objective value. Here, the variable-locks-driven fix-and-propagate heuristic performs best, followed by the variable-bound-driven one. On the right side the evaluation is restricted to instances where both heuristics found a solution; now the variable-bound-driven heuristic performs best with the variable-locks-driven heuristic being ranked last. Summing up, the variable-locks-driven fix-and-propagate heuristic performs very well with respect to the number of solutions found, but does not take into account the objective function. The clique- and variable-bound-driven heuristics both consider the objective function and construct solutions with better objective values but are successful for fewer instances. Among the two, the variable-bound-driven heuristic is more successful, but also considerably more expensive due to running up to five fixing scheme variants. Based on this analysis, we decided to configure the heuristics in a way such that the clique-driven heuristic is called first, followed by the variable-locks-driven one and finally the variable-bound-driven heuristic. We are running the faster heuristics first, because a solution generated by them provides a cutoff bound that may speed up the LP and sub-MIP solving of the variable-bound-driven heuristic.

For the overall performance evaluation, we ran SCIP once without any of the three structure-driven primal heuristics, once for each of the three heuristics with only this heuristic enabled and the other two disabled, and once with all three heuristics enabled. We used a cluster of 2.50 GHz Intel Xeon E5-2670 v2 CPUs with 128 GB main memory. Each job was run exclusively on one node with a time limit of 7200 seconds and a memory limit of 100 GB. In order to reduce the impact of performance variability (see Koch et al., 2011; Lodi and Tramontani, 2013), we ran each instance six times with different random seeds (one of them being the default seed). We

setting	opt	sols	first sol (s)	primal int.	time (s)
all instances (2967)					
nostructheur	1863	0	4.2	2072.5	431.7
onlyclique	1875	869	4.2	2017.5	429.0
onlyvbound	1877	1023	3.8	1971.7	428.9
onlylocks	1874	1443	4.0	2024.9	430.0
allstructheur	1880	1730	3.5	1899.9	421.2
hard instances (874)					
nostructheur	791	0	7.3	3003.3	686.9
onlyclique	803	187	7.1	2986.7	667.6
onlyvbound	805	231	6.4	2900.2	661.0
onlylocks	802	405	6.6	2750.5	671.0
allstructheur	808	479	5.7	2603.1	630.2

Table 10: Solution process statistics for SCIP with default settings and with additional structure-driven heuristics.

removed 27 instance/seed combinations that caused numerical troubles in one of the experiments, leaving us with 2967 instance/seed combinations. In a slight abuse of notation, we will refer to each instance/seed combination as an individual instance in the following. Table 10 summarizes the results of this experiment.

The table is divided into two parts. Each part lists solving process statistics for the five settings, each represented by one row, on a different (sub-)set of instances. The first part of the table considers all 2967 instances. The second part regards "hard", but solvable, instances, i.e., instances that at least one setting solved while at least one of the settings needed more than 100 seconds on this instance. The first column denotes the "setting", followed by columns listing the number of instances solved to "opt"imality within the time limit of two hours and the number of solutions ("sols") found by the structure-driven fix-and-propagate heuristics. Column four lists the time in seconds until the "first sol"ution was found for the instances, averaged by the shifted geometric mean (Achterberg, 2007b) with a shift of 1 second. Column "primal int." shows the shifted geometric mean with a shift of 100 of the primal integrals[1] for the instances. Finally, we present the shifted geometric mean of the running "time" in seconds with a shift of 10 seconds.

On the complete test set, each of the structure-driven fix-and-propagate heuristics helps to increase the number of instances solved to optimality within the time limit, while enabling all of them leads to to best results with 17 more instances being solved than without any of the heuristics. When running all three heuristics, an initial solution is constructed for 58.3 % of the instances. Each of the heuristic individually reduces the time needed to find a first feasible solution, by 1.7 % (clique-driven heuristic) to 10.3 % (variable-bound-driven heuristic). Used together, they even accomplish a reduction of 16.4 %. With respect to the primal integral, each heuristic alone accounts for a reduction by 2.3 % to 4.9 %, by enabling all three heuristics, a reduction of 8.3 % is obtained. But not only these measures tailored to primal heuristics are improved: the most important measure, the solving time to optimality, is also slightly improved by running the heuristics, up to a speed up of 2.4 % with all three heuristics enabled. Let us

[1] We compute the primal integral (Berthold, 2013) of instance i as $P(i) = \int_{t=0}^{t_{max}} \gamma_i(t)\, dt$ with t_{max}= 7200 seconds and $\gamma_i(t)$ the primal gap at time t.

note that the actual speedup is potentially slightly higher as about one third of the instances in the test set do not change their running time because they time out with all settings.

In order to analyze the effects of the heuristics in more detail, we regard the subset of "hard" instances, as introduced above. This excludes trivial instances and instances that are very easy to solve. For those instances, the potential for improvement is smaller while the overhead for running the heuristic has a larger impact. The hard instances, on the other hand, are the ones where algorithmic improvements are particularly needed and where we expected a larger impact of the heuristics. This is proven true by the second part of Table 10. The "hard" subset only excludes instances solved with all settings and those solved with none of the settings. Consequently, the number of additionally solved instances stays unchanged. The success rates of clique- and variable-bound-driven heuristic are both reduced by about 8 % as compared to the complete test set. The variable-locks-driven fix-and-propagate heuristic is almost as successful as before, while running all heuristic still finds a solution for 54.8 % of the hard instances. When running all heuristics, the time to the first solution is reduced by 21.2 %, the average primal integral by 13.3 %, and the solving time by 8.3 %.

These are impressive numbers for primal heuristics in mixed-integer programming. The improvement is not caused by the solutions found by the heuristics alone, however. There is also a side-effect which impacts performance: the generation of conflict constraints (Achterberg, 2007a; Witzig et al., 2017). They capture the essence of infeasible assignments detected during the fix-and-propagate phase and help to guide the subsequent search. To assess this effect, we further split the hard instances into the 479 instances where at least one of the heuristics found a solution on the one side and the remaining 395 on the other side. For the first set, the positive effects of running all three structure-driven heuristics are strengthened. The time to the first solution is reduced by 45.9 % and the average primal integral by 25.8 %. The average solving time goes down by 12.3 % while 11 more instances are solved. On the 395 instances where no solution is constructed, the time to the first solution is increased by 0.5 % and the average primal integral by 2.3 %. This is due to the overhead caused by the heuristics which slows down the inital root processing of the main MIP solve. In the long run, however, conflict constraints generated by the seemingly unsuccessful heuristic calls improve the solving time by 3.1 % and even help to solve 6 more instances. One could interpret the heuristic call in this case as some form of rapid learning (Berthold et al., 2010). An additional computational experiment in which we disabled the creation of conflict constraints showed a reduction in the solving time improvement by about one third, while the impact on the average time to a first solution and average primal integral is considerably smaller (about 5 % and 20 %, respectively). This shows that the main task of our newly proposed primal heuristics, namely generating primal feasible solutions, is still the main reason for the improvements we observed.

9 Conclusions and outlook

In this paper, we presented three primal heuristics which are based on global structures available within MIP solvers. Those structures are the clique table, the variable bound graph, and the variable locks based on the constraint matrix. The heuristics use these structures to define a sequence of variable fixings applied in a fix-and-propagate approach. The LP relaxation of the resulting sub-problem is then solved and rounded. If the rounded LP solution is not feasible, the sub-problem is solved in an LNS fashion. In our approach, domain propagation is not only used as a tool to avoid infeasible fixings, but rather are the fixing order and the fixing values decided based upon their effect on the domain propagation step. The global structures provide the tools to predict this effect by representing a part of the domain reductions that can be deduced from

a variable fixing.

We performed a detailed analysis of the three heuristics to derive appropriate default settings. Our final computational experiments indicate that all three heuristics complement each other in the academic MIP solver SCIP. When applying all of them at the beginning of the branch-and-bound search, they are able to generate solution for almost 60 % of the instances in standard MIP benchmark sets. This reduces ths shifted geometric means of both the time to the first solution as well as the primal integral significantly. The structure-driven fix-and-propagate heuristics prove to perform particularly well on hard instances, where they decrease the solving time by more than 8 %. Therefore, the updated versions of all three heuristics will be part of the next SCIP release and enabled by default.

Acknowledgements

The work for this article has been conducted within the *Research Campus Modal* funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM).

References

Tobias Achterberg. Conflict analysis in mixed integer programming. *Discret. Optim.*, 4(1):4–20, March 2007a. doi: 10.1016/j.disopt.2006.10.006.

Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007b.

Tobias Achterberg and Timo Berthold. Improving the Feasibility Pump. *Discrete Optimization*, Special Issue 4(1):77–86, 2007.

Tobias Achterberg and Christian Raack. The MCF-separator: detecting and exploiting multi-commodity flow structures in MIPs. *Mathematical Programming Computation*, 2(2):125–165, 2010.

Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of Combinatorial Optimization*, pages 449–481. Springer, 2013.

Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. doi: 10.1016/j.orl.2005.07.009.

Tobias Achterberg, Timo Berthold, and Gregor Hendel. Rounding and propagation heuristics for mixed integer programming. In Diethard Klatte, Hans-Jakob Lüthi, and Karl Schmedders, editors, *Operations Research Proceedings 2011*, Operations Research Proceedings, pages 71–76. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-29210-1_12.

Carlos E. Andrade, Shabbir Ahmed, George L. Nemhauser, and Yufen Shao. A hybrid primal heuristic for finding feasible solutions to mixed integer programs. *European Journal of Operational Research*, 263(1):62–71, 2017. doi: https://doi.org/10.1016/j.ejor.2017.05.003.

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and Tallys Yunes. BDD-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014. doi: 10.1007/s10732-014-9238-1.

Livio Bertacco, Matteo Fischetti, and Andrea Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, Special Issue 4(1):63–76, 2007.

Timo Berthold. Primal heuristics for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006.

Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614, 2013.

Timo Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, Technische Universität Berlin, 2014a.

Timo Berthold. RENS – the optimal rounding. *Mathematical Programming Computation*, 6(1): 33–54, 2014b.

Timo Berthold. Improving the performance of MIP and MINLP solvers by integrated heuristics. In Karl Franz Dörner, Ivana Ljubic, Georg Pflug, and Gernot Tragler, editors, *Operations Research Proceedings 2015*, pages 19–24. Springer International Publishing, Cham, 2017. doi: 10.1007/978-3-319-42902-1_3.

Timo Berthold and Ambros M. Gleixner. Undercover – a primal heuristic for MINLP based on sub-MIPs generated by set covering. In Pierre Bonami, Leo Liberti, Andrew J. Miller, and Annick Sartenaer, editors, *Proceedings of the EWMINLP*, pages 103–112, April 2010.

Timo Berthold and Gregor Hendel. Shift-and-propagate. *Journal of Heuristics*, 21(1):73–106, February 2015. doi: 10.1007/s10732-014-9271-0.

Timo Berthold, Thibaut Feydy, and Peter J. Stuckey. Rapid learning for binary programs. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 51–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi: 10.1007/978-3-642-13520-0_8.

Timo Berthold, Michael Perregaard, and Csaba Mészáros. Four good reasons to use an interior point solver within a MIP solver. Technical Report 17-42, ZIB, Takustr.7, 14195 Berlin, 2017.

Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

Robert E. Bixby, Sebastiàn Ceria, Cassandra M. McZeal, and Martin W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, (58):12–15, June 1998.

Robert E. Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. MIP: Theory and practice – closing the gap. In Michael J. D. Powell and Stefan Scholtes, editors, *Systems Modelling and Optimization: Methods, Theory, and Applications*, pages 19–49. Kluwer Academic Publisher, 2000.

Ralf Borndörfer, Martin Grötschel, and Ulrich Jäger. Planning problems in public transit. In Martin Grötschel, Klaus Lucas, and Volker Mehrmann, editors, *Production Factor Mathematics*, pages 95–121. Springer Berlin Heidelberg, 2010.

COR@L. MIP Instances, 2014. `http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/`.

R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2004.

Santanu Dey, Andres Iroume, Marco Molinaro, and Domenico Salvagnin. Improving the randomization step in feasibility pump. *arXiv preprint arXiv:1609.08121*, 2016.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

Matteo Fischetti and Andrea Lodi. Repairing MIP infeasibility through local branching. *Computers & Operations Research*, 35(5):1436–1445, 2008. doi: 10.1016/j.cor.2006.08.004. Special Issue: Algorithms and Computational Methods in Feasibility and Infeasibility.

Matteo Fischetti and Andrea Lodi. Heuristics in mixed integer programming. In James J. Cochran, Louis A. Cox, Pinar Keskinocak, Jeffrey P. Kharoufeh, and J. Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010. Online publication.

Matteo Fischetti and Michele Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, 2014.

Matteo Fischetti and Domenico Salvagnin. Feasibility pump 2.0. *Mathematical Programming Computation*, 1:201–222, 2009.

Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

Armin Fügenschuh and Alexander Martin. Computational integer programming and cutting planes. In Karen Aardal, George L. Nemhauser, and Robert Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 69–122. Elsevier, 2005.

Gerald Gamrath, Timo Berthold, Stefan Heinz, and Michael Winkler. Structure-based primal heuristics for mixed integer programming. In *Optimization in the Real World*, volume 13, pages 37 – 53. Springer Tokyo, 2015. doi: 10.1007/978-4-431-55420-2_3.

Frédéric Gardi. Toward a mathematical programming solver based on local search. Habilitation thesis, Université Pierre et Marie Curie, 2013.

Shubhashis Ghosh. DINS, a MIP improvement heuristic. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Proceedings*, volume 4513 of *LNCS*, pages 310–323. Springer Berlin Heidelberg, 2007.

G. Guastaroba, M. Savelsbergh, and M.G. Speranza. Adaptive kernel search: A heuristic for solving mixed integer linear programs. *European Journal of Operational Research*, 263(3): 789–804, 2017. doi: https://doi.org/10.1016/j.ejor.2017.06.005.

Menal Guzelsoy, George Nemhauser, and Martin Savelsbergh. Restrict-and-relax search for 0-1 mixed-integer programs. *EURO Journal on Computational Optimization*, pages 1–18, 2013. online first publication.

Pierre Hansen, Nenad Mladenović, and Dragan Urošević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33(10):3034–3045, 2006.

Stefan Heinz, Wen-Yang Ku, and J.Christopher Beck. Recent improvements using constraint integer programming for resource allocation and scheduling. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 12–27. Springer Berlin Heidelberg, 2013.

Ellis L Johnson and Manfred W Padberg. Degree-two inequalities, clique facets, and biperfect graphs. *North-Holland Mathematics Studies*, 66:169–187, 1982.

Utku Koc and Sanjay Mehrotra. Generation of feasible integer solutions on a massively parallel computer using the feasibility pump. *Operations Research Letters*, 2017. doi: https://doi.org/10.1016/j.orl.2017.10.003.

Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

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

Jasmina Lazić. Variable and single neighbourhood diving for MIP feasibility. *Yugoslav Journal of Operations Research*, 26(2), 2016.

EvaK. Lee and DavidP. Lewis. Integer programming for telecommunications. In MauricioG.C. Resende and PanosM. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 67–102. Springer US, 2006.

Andrea Lodi. Mixed integer programming computation. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer Berlin Heidelberg, 2010.

Andrea Lodi. The heuristic (dark) side of MIP solvers. In El-Ghazali Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 273–284. Springer Berlin Heidelberg, 2013.

Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory Driven by Influential Applications*, chapter 1, pages 1–12. INFORMS, 2013. doi: 10.1287/educ.2013.0112.

Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin, 2017.

Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001. doi: 10.1287/opre.49.3.363.11211.

Lluís-Miquel Munguía, Shabbir Ahmed, David A. Bader, George L. Nemhauser, and Yufen Shao. Alternating criteria search: a parallel large neighborhood search algorithm for mixed integer programs. *Computational Optimization and Applications*, 2017. doi: 10.1007/s10589-017-9934-5.

Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming.* Springer Science & Business Media, 2006.

Jennifer Pryor and John W. Chinneck. Faster integer-feasibility in mixed-integer linear programs by branching to force change. *Computers & Operations Research*, 38(8):1143–1152, 2011.

Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

Domenico Salvagnin. Detecting and exploiting permutation structures in MIPs. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, volume 8451 of *Lecture Notes in Computer Science*, pages 29–44. Springer Berlin Heidelberg, 2014.

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

Chris Wallace. ZI round, a MIP rounding heuristic. *Journal of Heuristics*, 16(5):715–722, 2010.

Michael Winkler. Presolving for pseudo-Boolean optimization problems. Diploma thesis, Technische Universität Berlin, 2014.

Jakob Witzig, Timo Berthold, and Stefan Heinz. Experiments with conflict analysis in mixed integer programming. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, pages 211–220. Springer International Publishing, Cham, 2017. doi: 10.1007/978-3-319-59776-8_17.

Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus.* PhD thesis, Technische Universität Berlin, 1996.