Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

GERALD GAMRATH, TIMO BERTHOLD, STEFAN HEINZ, MICHAEL WINKLER

Structure-based primal heuristics for mixed integer programming

This paper is to appear in the book “Optimization in the Real World – Towards Solving Real-World Optimization Problems”. The final publication is available at http://www.springerlink.com/.

ZIB Report 15-26 (April 2015)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

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

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

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

Structure-based primal 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 help to reach optimality faster and provide good feasible solutions early in the solving process. In this paper, we present two new primal heuristics which take into account global structures available within MIP solvers to construct feasible solutions at the beginning of the solving process. These heuristics follow a large neighborhood search (LNS) approach and use global structures to define a neighborhood that is with high probability significantly easier to process while (hopefully) still containing good feasible solutions. The definition of the neighborhood is done by iteratively fixing variables and propagating these fixings. Thereby, fixings are determined based on the predicted impact they have on the subsequent domain propagation. The neighborhood is solved as a sub-MIP and solutions are transferred back to the original problem. Our computational experiments on standard MIP test sets show that the proposed heuristics find solutions for about every third instance and therewith help to improve the average solving time.

Keywords: mixed-integer programming, large neighborhood search, primal heuristics, 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:

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

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

This allows to model many real-world optimization problems from various fields like production planning [28], scheduling [20], transportation [13], or telecommunication networks [24]. 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

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

algorithm. Therefore, very powerful generic solvers for MIPs have been developed over the last decades, which are used widely in research and practice [25, 11, 5].

These solvers are based on a branch-and-bound algorithm [23, 15], which is accalerated by various extensions. The basic concept is to split the problem into subproblems, until they are easy enough to be solved. During this process, a lower bound is computed for each subproblem 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 subproblems 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 subproblem temporarily and solves the corresponding LP relaxation until an integral solution is obtained. For more details on primal heuristics, we refer to [6, 18, 8]. In this paper, we introduce two novel heuristics based on the large neighborhood search (LNS) paradigm. This concept defines a subproblem, the neighborhood, by adding restrictions to the problem, and then solves this subproblem as a MIP. A more detailed discussion of LNS 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 (see [4, 31]). This detection is often done in the presolving phase, which is a preprocessing step trying to remove redundancies from the model and tighten the fomulation. Additionally, several global structures are detected in this phase and stored for later use. An overview of different global structures in MIP solvers and details about two of them, the clique table and the variable bound graph, are given in Section 3.

The heuristics presented in this paper define a neighborhood based on the clique table and the variable bound graph. 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 (see, e.g., the shift-and-propagate heuristic [10]), 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 problem was reduced sufficiently, the remaining problem is then solved as a LNS sub-MIP.

A detailed description of the general scheme of the heuristics and how the global structures are used exactly is discussed in Section 4. The impact of the heuristics is evaluated by the computational experiments presented in Section 5. Finally, Section 6 gives our conclusions and an outlook.

2 Large neighborhood search for MIP

Large neighborhood search (LNS) heuristics are an important component of modern MIP solvers, see, e.g., [6, 18, 26, 8]. The main idea of LNS is to restrict the search for "good" solutions to a neighborhood centered at a particular reference point—typically the incumbent or another feasible solution. The hope is that such a restricted search space makes the subproblem much easier to solve, while still providing solutions of high quality. Of course,

these restricted subproblems do not have to be solved to optimality; they are mainly searched for an improving solution.

To define the neighborhood, the feasible region of the MIP is restricted by additional constraints: most often variable fixings or some very restrictive cardinality constraint. A good definition of the neighborhood is the crucial point for LNS heuristics. There are different characteristics of a "good" neighborhood:

1. it should contain high quality solutions,
2. these solutions should be easy to find, and
3. the neighborhood should be easy to process.

Naturally, these three goals are conflicting in practice.

The relaxation induced neighborhood search (RINS) [16] uses two reference points: The incumbent MIP solution which fulfills the first two requirements and the optimum of the LP relaxation which fulfills the latter two. RINS defines the neighborhood by fixing all integer variables which take the same value in both solutions. In contrast to RINS, the *relaxation enforced neighborhood search* (RENS) [9] does not require an incumbent solution. RENS fixes all integer variables that take an integral value in the optimal solution of the LP relaxation. *Crossover* [6] is an improvement heuristic that is inspired by genetic algorithms [30] and requires more than one feasible solution. For a set of feasible solutions, it fixes variables that take identical values in all of them.

Local Branching [17] measures the distance to the reference point in Manhattan norm on the integer variables and only considers solutions which are inside a k-neighborhood of the reference point, where k is typically between 10 and 20. *DINS* [19] combines the ideas of RINS and Local Branching. It defines the neighborhood by introducing a distance function between the incumbent solution and the optimum of the LP relaxation. When applied during a branch-and-bound search, it further takes into account how variables change their values at different nodes of the tree.

DINS, RINS, RENS, and Crossover 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 subproblem does not contain any improving solution or even becomes infeasible.

The present paper addresses this issue by the use of global structures and propagation for defining a set of variables to be fixed and an order of fixing them.

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, modern MIP solvers aim at detecting structures within a model and making use of them for heuristics, cutting plane separation or presolving, see e.g., [4, 31].

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., [1]. The remainder of the section explains the clique table and the variable bound graph in more detail.

3.1 The clique table

A *clique* is a set of binary variables of which at most one variable can be set to one. A clique can be given directly as a linear inequality $\sum x_i \leq 1$ or derived from more general constraints such as knapsacks: given a constraint $\sum_{i \in J} w_i x_i \leq c$, each subset $J' \subseteq J$ for which $w_j + w_k > c$ for all $(j,k) \in J' \times J'$ defines a clique. In addition, presolving techniques such as probing [32] can be used to detect cliques which are only given implicitly and cannot be extracted directly from a model constraint.

Similarly, *negated cliques* [33] 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 [21] or to deduce stronger reductions in presolving and propagation [32]. In Section 4, we will show how the clique table can be used to define a neighborhood for a LNS based primal heuristic.

3.2 The variable bound graph

Variable bound constraints are linear inequalities which contain exactly two variables. 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. Consequently, a variable bound relation expresses the dependency of one bound of a variable on a bound of another variable. Typical examples for the use of variable bound constraints are precedence constraints on start time variables in scheduling or big-M constraints modeling fixed-costs in production planning.

Similar to the clique information, variable bound relations cannot only be deduced from variable bound constraints, but can also be identified within more general constraints or during presolving, e.g., by probing. Variable bound relations 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 variables [27]. In order to make variable bound relations available for those components, they are stored in a global structure, the *variable bound graph*. In this directed graph, each node corresponds to the lower or upper bound of a variable and each variable bound relation is represented by an arc pointing from the influencing bound to the dependent bound. If a bound of a variable is tightened, implications can be read from this graph by following all paths starting at the corresponding node.

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, bounds (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 labelled with the bound it represents.

4 Structure-based primal heuristics

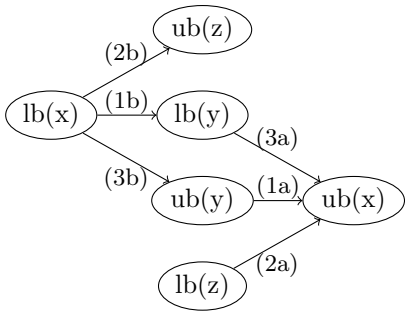
In this section, we present two new primal heuristics for mixed integer programming which are based on global structures collected by MIP solvers, see Sec. 3. Both heuristics are explicitly designed for use inside a MIP solver and not as standalone procedures. They

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

(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) Variable bounds.



(c) Variable bound graph.

Figure 1: Example of a variable bound graph.

employ the LNS paradigm (see Sec. 2) and are start heuristics, i.e., they do not need a feasible solution as reference point but aim at constructing an initial feasible solution.

In this aspect, they are similar to RENS; however, they differ in the definition of the neighborhood. While RENS uses a solution to the LP relaxation as reference point, our novel heuristics can even run before an LP is solved: They analyze the clique table and the variable bound graph built during the presolving phase and derive a set of variable fixings with a high probability of success for the subsequent LNS call.

The general scheme is the same for both heuristics. It is illustrated in Algorithm 1. In a first step, a subset of the integer variables is fixed based on the respective structure (lines 1–1). This is done in a diving-like fashion, i.e., in each iteration, the global structure is used to decide on one variable to be fixed. After that, two rounds of domain propagation are performed 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. This process is iterated until all discrete variables in the used global structure were fixed. It can be interpreted as a dive in the tree with a domain propagation call after each fixing instead of solving a linear program, which is similar to the shift-and-propagate heuristic [10]. 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. This is done in lines 1–1, and uses a method `domain_propagation`, which—given a MIP and the number of propagation rounds—performs domain propagation and returns the updated MIP as well as the information whether an infeasibility was detected during propagation. After the first backtrack, we stop the fixing process in order to avoid too much effort being spent by repeated backtracking.

If sufficiently many variables were fixed in this first phase, we solve an LP on the remaining problem and try to round the LP solution with a simple rounding heuristic [1] (see lines 1–1). Note that this LP is significantly smaller (and hopefully easier to solve) than the original LP relaxation because of the fixing threshold.

If the LP solution and the simple rounding heuristic did not lead to a primal feasible solution, the LNS search is started. For this, the neighborhood is defined by the fixings obtained in the previous phase. A sub-MIP for this neighborhood is created and solved (with a number of working limits to restrict the computational effort), see line 1. If a feasible solution was found during sub-MIP solving, it is returned.

The difference between the two heuristics is how and in which order the variables are fixed in the first step. The *clique heuristic* uses the clique table, while the *variable bound*

Algorithm 1: Structure-based heuristics – general scheme

input : - MIP $\mathcal{P}(c, A, b, \ell, u, \mathcal{I})$
- fixing threshold α
- clique table $\mathcal{T}$ (`NULL` for variable bound heuristic), or
- variable bound graph $\mathcal{G}$ (`NULL` for clique heuristic)

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

begin
  `// 1. fixing phase`
  if $\mathcal{T} \neq$ *NULL* **then**
    `// apply Alg. 2 to fix variables`
    $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}), \texttt{inf}) = \texttt{clique\_fixing}(\mathcal{P}(c, A, b, \ell, u, \mathcal{I}), \mathcal{T})$;
  else
    `// apply Alg. 3 to fix variables`
    $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}), \texttt{inf}) = \texttt{variable\_bound\_fixing}(\mathcal{P}(c, A, b, \ell, u, \mathcal{I}), \mathcal{G})$;
  `// 2. backtracking`
  if *inf* **then**
    $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}) \leftarrow$ undo last fixing $x_k = f_k$ and induced propagations;
    `// remove` f_k `from domain of` x_k
    if $\tilde{\ell}_k = f_k$ **then**
      $\tilde{\ell}_k = \tilde{\ell}_k + 1$;
    else
      $\tilde{u}_k = \tilde{u}_k - 1$;
    `// perform 2 rounds of domain propagation`
    $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}), \texttt{inf}) = \texttt{domain\_propagation}(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}), 2)$;
    if *inf* **then**
      return *NULL*;
  `// 3. LP solving`
  if $|\{i \in \mathcal{I} \mid \ell_i < u_i\}| \leq \alpha|\mathcal{I}|$ **then**
    $x^\star \leftarrow$ solve $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \emptyset)$;
    `// try to round LP solution`
    $x^\star \leftarrow \texttt{simple\_round}(x^\star)$;
    if $x^\star_i \in \mathbb{Z}$ *for all* $i \in \mathcal{I}$ **then**
      return $x^\star$;
    else
      `// 4. sub-MIP solving`
      $x^\star \leftarrow$ solve $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I})$ (with working limits, see Sec. 5);
      return $x^\star$;
  else
    return *NULL*;
end

heuristic takes into account the variable bound graph. Both these structures 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 two named global structures is explained in the following.

4.1 Using the clique structure to define variable fixings

In order to define a set of variable fixings with high probability of both being feasible and leading to a good solution, the clique structure can be used as illustrated in Algorithm 2.

First, a set of cliques is computed which partitions the set of binary variables. Note that a single binary variable always forms a trivial clique, so this partition is guaranteed to exist. Additionally, each clique cover can be transformed into a clique partition, since every subset of a clique forms a clique itself. Using a clique partition is a heuristic approach to finding a promising fixing order.

Given a clique of the partition, we first check if a variable within the clique was fixed to one already, e.g., by domain propagation in a previous iteration (lines 2–2). If this is not the case, we choose a variable with smallest objective coefficient among the unfixed variables in the clique and fix it to one, see lines 2–2. After that, two rounds of domain propagation are performed, which fixes all other unfixed variables in the clique to zero, but might also identify valid bound changes for variables in other cliques or even for variables not contained in the clique table. This is repeated until the propagation detects infeasibility or all cliques were handled. Since fixing a variable in a clique to one causes multiple other fixings, this scheme helps to reduce the neighborhood size. On the other hand, we set the cheapest variable in the clique to one in order to not increase the objective value too much and thereby aim at finding high quality solutions.

After all binary variables have been fixed by this algorithm, the remaining problem is solved by an LNS approach as illustrated in Algorithm 1.

4.2 Using the variable bound graph to define variable fixings

In the variable bound 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. 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. The fixing algorithms of the variables bound heuristics are summarized in Algorithm 3. As before, this is a sub-algorithm of Algorithm 1 and defines the subproblem to be solved by an LNS approach in a subseuqent step.

The nodes of the variable bound graph are processed in the almost topological order, skipping continuous variables since we only want to fix integer variables (line 3). Additionally, variables fixed by domain propagation in a previous iteration are ignored, see line 3. 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 defined by all paths in the variable bound graph starting at node v. Consequently, the earlier a variable bound is considered within the almost topological order, the more impact on other bounds we expect when tightening it.

All four variants process the nodes of the variable bound graph in almost topological order, but use different rules to decide whether and to which value the variable corresponding to a node is fixed (see lines 3–3).

Algorithm 2: `clique_fixing`

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

output: - subproblem $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I})$ to solve as sub-MIP
- Bool `inf` – `TRUE` if selected assignement was detected to be infeasible

```
1  begin
       // 1.  clique partition
2      compute clique partition C̄ = {C_1, ..., C_k};
3      for i ∈ {1, ..., |C̄|} do
4          fixed ← FALSE;
           // look for variable fixed to 1, e.g., by propagation of previous
              cliques
5          for j ∈ C_i do
6              if ℓ̃_j = 1 then
7                  if fixed then
                       // two variables fixed to 1 in a clique → infeasible
8                      return (P(c, A, b, ℓ̃, ũ, I), TRUE);
9                  fixed ← TRUE;
10         if fixed = FALSE then
               // fix cheapest variable to 1
11             x_{c_1}, ..., x_{c_{n_i}} ← sort variables of clique C_i by increasing objective coefficient;
12             for j ∈ {1, ..., n_i} do
13                 if ũ_{c_j} = 0 then
14                     continue;
15                 else
16                     ℓ̃_{c_j} ← 1;
17                     break ;
               // perform 2 rounds of domain propagation
18             (P(c, A, b, ℓ̃, ũ, I), inf) = domain_propagation(P(c, A, b, ℓ̃, ũ, I), 2);
19             if inf then
20                 return (P(c, A, b, ℓ̃, ũ, I), TRUE);
21     return (P(c, A, b, ℓ̃, ũ, I), FALSE);
22 end
```

The first variant by which the variable bound 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. In Algorithm 3, this variant is obtained when parameters `tighten` and `obj` are both set to `FALSE`.

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 variable bounds, a concept known to be rather effective in order to drive the solution to feasibility faster, cf. [29]. In Algorithm 3, this corresponds to `tighten` = `TRUE` and `obj` = `FALSE`.

We obtained two more variants by extending the previous ones to take into account the objective function (triggered by setting `obj` to `TRUE` in Algorithm 3). 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 nonnegative, and its upper bound otherwise.

Variant three is based on variant one, in the sense that it fixes variables to the bound provided by the node in the topological sorting in order to obtain a large neighborhood. It even goes one step further: The variables are never fixed to their best bound, but only to their worst bound. While increasing the objective function value, these decisions are often less probable to lead to an infeasible subproblem, so that this variant aims mainly at finding a feasible solution.

On the other hand, the fourth variant tries to cause many changes by the fixings—similar to variant two—, so it changes the bound corresponding to the current node in the topological order by fixing the variable to the reverse bound. However, it only applies this fixing if this helps to improve the objective function value, i.e., if the variable is fixed to is its best bound. This allows for better solutions, while hopefully still fixing enough variables to have reasonable LNS solving times.

After each fixed variable, two rounds of domain propagation are performed (line 3) to identify consequences of the fixing.

There are arguments for each of these rules and indeed, none of them dominates the other but rather do they complement each other. Therefore, the variable bound heuristic is run up to four times in a row, once with each rule.

5 Computational results

In this section, we present computational experiments showing the effect of the clique and variable bound heuristics. We used an implementation based on the academic MIP solver SCIP 3.1.1 [2] with SoPlex 2.0.1 [34] as underlying LP solver. All results were obtained on a cluster of 3.2 GHz Intel Xeon X5672 CPUs with 12 MB cache and 48 GB main memory. Each job was run exclusively on one node with a time limit of 3600 seconds.

Our experiments were performed on the MMMC test set which contains all instances from the last three MIPLIB benchmark sets [12, 3, 22] as well as the Cor@l test set [14]. We removed duplicates and the instances lrn, neos-1058477, neos-847051, and npmv07 because they caused numerical troubles with both variants. This left us with a total of 495 instances.

Each heuristic is implemented in a primal heuristics 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. We decided not to run

Algorithm 3: `variable_bound_fixing`

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

output: - subproblem $\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I})$ to solve as sub-MIP
- Bool `inf` – `TRUE` if selected assignement was detected to be infeasible

begin
  `// 1. topological ordering`
  $b_1, \ldots, b_k \leftarrow$ bounds of variables in almost topological order;
  `// 2. process bound in topological order`
  for $i \in \{1, \ldots, k\}$ **do**
    $x_j \leftarrow$ variable represented by b_i;
    `// skip continuous variables`
    if $j \notin \mathcal{I}$ **then continue** ;
    `// variable is already fixed`
    if $\tilde{\ell}_j = \tilde{u}_j$ **then continue** ;
    `//` b_i `represents lower bound`
    if $b_i = lb(x_j)$ **then**
      if ***not*** `obj` ***or*** $c_j < 0$ **then**
        `// tighten the regarded bound`
        if *`tighten`* **then**
          $\tilde{\ell}_j \leftarrow \tilde{u}_j$;
        `// fix to the regarded bound`
        else
          $\tilde{u}_j \leftarrow \tilde{\ell}_j$;
    `//` b_i `represents upper bound`
    else
      if ***not*** `obj` ***or*** $c_j \geq 0$ **then**
        `// tighten the regarded bound`
        if *`tighten`* **then**
          $\tilde{u}_j \leftarrow \tilde{\ell}_j$;
        `// fix to the regarded bound`
        else
          $\tilde{\ell}_j \leftarrow \tilde{u}_j$;
    `// perform 2 rounds of domain propagation`
    $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}), \texttt{inf}) =$ `domain_propagation`$(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}), 2)$;
    if *`inf`* **then**
      return $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}),$ *`TRUE`*$)$;
  return $(\mathcal{P}(c, A, b, \tilde{\ell}, \tilde{u}, \mathcal{I}),$ *`FALSE`*$)$;
end

	affected models				unaffected models		
heuristic	instances	found	time	share (%)	instances	time	share (%)
clique	259	139	1.19	3.43	236	0.03	0.25
vbound	127	64	1.63	7.12	368	0.00	0.01
both	309	176	1.67	5.80	186	0.04	0.30

Table 1: Root node statistics for the clique and variable bound heuristic individually and jointly on the MMMC test set (495 instances).

our global structure-based heuristics on local bounds, since the computational overhead of LNS heuristics is more significant than for, e.g., simple rounding or diving heuristics.

During the fixing phase of both heuristics, we apply all default domain propagation algorithms of SCIP, but limit the number of domain propagation rounds performed after a variable fixing by 2. This typically suffices to identify most implied bound changes and infeasibilities while avoiding spending too much time for propgation. In order to limit the effort spent within the heuristics, we use working limits for the subsequent LP and sub-MIP solving. First, we aim at having a significantly easier problem after the fixing phase. To this end, we run the heuristic only if at least 30 % of the integer variables were fixed. 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 [1]. 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.

In our first experiment, we evaluated the effectiveness of the new heuristics and their running times. All other SCIP heuristics were disabled for this experiment. The results are presented in Table 1 for the two heuristics individually, as well as for both of them together. In the columns labeled "affected models", we list the number of instances on which the fixing rate of 30 % was reached, the number of instances on which a solution was found, and the average running time of the heuristic on these models. Note that the classification is done individually for all three variants and the sets of affected instances differ significantly so that the times should not be compared directly. Therefore, we also show the running time compared to the overall presolving and root node processing time in column "share (%)". Additionally, we also present the number of unaffected instances as well as the average running time and running time share on these instances in columns "unaffected models".

Our first observation is that the needed structures occur regularly: More than half of the instances contain a large enough clique structure to reach the desired fixing rate; on the other hand, the variable bound graph is sufficiently large for more than 25 % of the instances. Both heuristics succeed in finding a feasible solution on more than every second instance with the appropriate structure. Together, they find solutions for more than one third of the complete test set.

The running times on the affected instances are below two seconds on average for both heuristics individually as well as jointly. When compared to the root node processing time including presolving, the relative running time of the variable bound heuristic is twice as high as that of the clique heuristic which is consequential since it runs four different fixing rules (some of which might be fast because they do not lead to the desired fixing rate). On average over all affected instances, the running time of both heuristics together was less than 6 % of the overall root node solving time. This is more than reasonable for heuristics with a joint success rate of more than 50 %.

	MMMC (495 instances)					all optimal (299)	
variant	solved	time first	avg. gap	nodes	time (s)	nodes	time (s)
default	300	11.3	6.93 %	5892	340.4	1459	66.0
+ struct	305	10.6	6.58 %	5626	328.9	1347	63.0

Table 2: Solution process statistics for SCIP with default settings and with additional structural heuristics (clique and variable bound heuristic).

On the other hand, the running times on unaffected models are very small. The variable bound heuristic spends almost no time, while computing the clique partition in the clique heuristic can need up to half a second on some instances. On average, however, the time needed is still quite small and both heuristics together increase the root node processing time by only 0.3 % on average for unaffected models.

In order to evaluate the impact of the newly proposed structure-based heuristics on the solving process, we performed another experiment which is summarized in Table 2. We compare the default settings of SCIP—with all default SCIP heuristics—to a variant with both structural heuristics enabled. Again, we observe an improvement by the structural heuristics. With clique and variable bound heuristic, we are able to solve 6 instances that were not solved before, while only one instance cannot be solved anymore. We reduce the time to finding the first solution (column "time first") by 6 % as well as the solving time by 3 % in the shifted geometric mean[1]. The average primal gap[2] (on instances for which a feasible solution is known) is decreased by 5 % as well, showing the consistent improvement in terms of solutions found and their quality. When looking only at those 299 instances solved to optimality by both variants, the time reduction amounts to 5 % and we see that also the number of branch-and-bound nodes needed to prove optimality is reduced by 8 % in the shifted geometric mean.

6 Conclusions and outlook

In this paper, we presented two primal heuristics which are based on global structures available within MIP solvers, namely the clique table and the variable bound graph.

Based on these structures, we derive variable fixings which are applied iteratively with intermediate domain propagation rounds. The LP relaxation of the resulting subproblem is then solved. If the rounded LP solution is not feasible, the subproblem 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.

When applied carefully, these heuristics find solutions for more than one third of the instances in standard MIP benchmark sets. They slightly speed up the overall solving process and help to solve more instances to optimality. Therefore, clique and variable bound heuristic will be part of the next SCIP release.

[1] For a definition and discussion of the shifted geometric mean, see [1, Appendix A]. We use shifts of 10 and 100 for time and nodes, respectively.

[2] We compute the average primal gap by means of the primal integral [7] as $P(t_{max})/t_{max}$ with t_{max}= 3600 seconds, $P(x) = \int_{t=0}^{x} \gamma(t)\,dt$ and $\gamma(t)$ the primal gap at time t.

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). The authors would like to thank the anonymous reviewer for helpful comments on the paper.

References

[1] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007)

[2] Achterberg, T.: SCIP: Solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)

[3] Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters **34**(4), 1–12 (2006). DOI 10.1016/j.orl.2005.07.009

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

[5] Achterberg, T., Wunderling, R.: Mixed integer programming: Analyzing 12 years of progress. In: Facets of Combinatorial Optimization, pp. 449–481. Springer (2013)

[6] Berthold, T.: Primal heuristics for mixed integer programs. Diploma thesis, Technische Universität Berlin (2006)

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

[8] Berthold, T.: Heuristic algorithms in global MINLP solvers. Ph.D. thesis, Technische Universität Berlin (2014)

[9] Berthold, T.: RENS – the optimal rounding. Mathematical Programming Computation **6**(1), 33–54 (2014)

[10] Berthold, T., Hendel, G.: Shift-and-propagate. Journal of Heuristics (2015). To appear

[11] Bixby, R.E.: A brief history of linear and mixed-integer programming computation. Documenta Mathematica pp. 107–121 (2012)

[12] Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Optima (58), 12–15 (1998)

[13] Borndörfer, R., Grötschel, M., Jäger, U.: Planning problems in public transit. In: M. Grötschel, K. Lucas, V. Mehrmann (eds.) Production Factor Mathematics, pp. 95–121. Springer Berlin Heidelberg (2010)

[14] COR@L: MIP Instances (2014). `http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/`

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

[16] Danna, E., Rothberg, E., Pape, C.L.: Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming **102**(1), 71–90 (2004)

[17] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**(1-3), 23–47 (2003)

[18] Fischetti, M., Lodi, A.: Heuristics in mixed integer programming. In: J.J. Cochran, L.A. Cox, P. Keskinocak, J.P. Kharoufeh, J.C. Smith (eds.) Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc. (2010). Online publication

[19] Ghosh, S.: DINS, a MIP improvement heuristic. In: M. Fischetti, D.P. Williamson (eds.) Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Proceedings, *LNCS*, vol. 4513, pp. 310–323. Springer Berlin Heidelberg (2007)

[20] Heinz, S., Ku, W.Y., Beck, J.: Recent improvements using constraint integer programming for resource allocation and scheduling. In: C. Gomes, M. Sellmann (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, *Lecture Notes in Computer Science*, vol. 7874, pp. 12–27. Springer Berlin Heidelberg (2013)

[21] Johnson, E.L., Padberg, M.W.: Degree-two inequalities, clique facets, and biperfect graphs. North-Holland Mathematics Studies **66**, 169–187 (1982)

[22] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation **3**(2), 103–163 (2011)

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

[24] Lee, E., Lewis, D.: Integer programming for telecommunications. In: M. Resende, P. Pardalos (eds.) Handbook of Optimization in Telecommunications, pp. 67–102. Springer US (2006)

[25] Lodi, A.: Mixed integer programming computation. In: M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, L.A. Wolsey (eds.) 50 Years of Integer Programming 1958-2008, pp. 619–645. Springer Berlin Heidelberg (2010)

[26] Lodi, A.: The heuristic (dark) side of MIP solvers. In: E.G. Talbi (ed.) Hybrid Metaheuristics, *Studies in Computational Intelligence*, vol. 434, pp. 273–284. Springer Berlin Heidelberg (2013)

[27] Marchand, H., Wolsey, L.A.: Aggregation and mixed integer rounding to solve MIPs. Operations Research **49**(3), 363–371 (2001). DOI 10.1287/opre.49.3.363.11211

[28] Pochet, Y., Wolsey, L.A.: Production planning by mixed integer programming. Springer Science & Business Media (2006)

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

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

[31] Salvagnin, D.: Detecting and exploiting permutation structures in MIPs. In: H. Simonis (ed.) Integration of AI and OR Techniques in Constraint Programming, *Lecture Notes in Computer Science*, vol. 8451, pp. 29–44. Springer Berlin Heidelberg (2014)

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

[33] Winkler, M.: Presolving for pseudo-Boolean optimization problems. Diploma thesis, Technische Universität Berlin (2014)

[34] Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin (1996)