TIMO BERTHOLD AND GREGOR HENDEL

Shift-And-Propagate

Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

ZIB-Report 13-03 (January 2013)

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

Shift-And-Propagate

Timo Berthold[*] and Gregor Hendel[†]

Abstract

For mixed integer programming, recent years have seen a growing interest in the design of general purpose primal heuristics for use inside complete solvers. Many of these heuristics rely on an optimal LP solution. Finding this may itself take a significant amount of time.

The presented paper addresses this issue by the introduction of the Shift-and-Propagate heuristic. Shift-and-Propagate is a pre-root primal heuristic that does not require a previously found LP solution. It applies domain propagation techniques to quickly drive a variable assignment towards feasibility. Computational experiments indicate that this heuristic is a powerful supplement of existing rounding and propagation heuristics.

Keywords: primal heuristic, mixed integer programming, domain propagation, rounding

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

1 Introduction

Considering the increasing number of industrial applications, there is a large commercial interest in solving *mixed integer programs* (MIPs). It is well-known that, apart from complete solving methods, general-purpose *primal heuristics* like the feasibility pump [18] are often able to find high-quality solutions rapidly. Besides their application as standalone procedures, primal heuristics have become a substantial component of state-of-the-art solvers for mixed integer programming [10, 15]. The presented paper introduces a new start heuristic that is designed for the application inside a MIP solver.

Definition 1. *Let $m, n \in \mathbb{N}$, $\mathcal{I} \cup \mathcal{C} = \mathcal{N}$ be a partition of the variable index set $\mathcal{N} := \{1, \ldots, n\}$, and l and u be lower and upper bound vectors on the variables with $l_j \in \mathbb{R} \cup \{-\infty\}$ and $u_j \in \mathbb{R} \cup \{\infty\}$ for all $j \in \mathcal{N}$, respectively. Let further $A \in \mathbb{R}^{m,n}$ be a real matrix, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ be real vectors. A* mixed integer program *(MIP) P is defined as follows:*

$$
\begin{aligned}
\text{Minimize} \quad & c^T x \\
s.\,t. \quad & Ax \le b \\
& l \le x \le u \\
& x_j \in \mathbb{Z} && \text{for } j \in \mathcal{I} \\
& x_j \in \mathbb{R} && \text{for } j \in \mathcal{C}
\end{aligned}
$$

The special case that $\mathcal{C} = \mathcal{N}$ is refered to as a *linear program* (LP). Further, we call a solution that fulfills all of the linear constraints, but not necessarily all integrality constraints, *LP-feasible*. We denote the vector corresponding to the i-th row of matrix A by a_i.

[*]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, berthold@zib.de
[†]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, hendel@zib.de

In today's market, one can find a large variety of both commercial [35, 36, 37] and non-commercial [34, 38, 40] MIP solving software that is capable of solving many MIPs of practical relevance to proven optimality. *Primal heuristics* are algorithms that aim at finding a feasible solution quickly; however, they are incomplete algorithms that are not guaranteed to succeed. Primal heuristics can be used as standalone procedures if only a feasible solution needs to be found within short time. At least as importantly, they are employed as subroutines inside complete solvers. Each of Cplex, Gurobi, SCIP and CBC feature a double-digit quantity of primal heuristics.

Recent years have seen several publications on general-purpose primal heuristics for MIPs, including [3, 4, 7, 8, 9, 11, 12, 17, 19, 20, 23, 24, 28, 29, 31, 33]. For an overview, see [10, 21, 22]. When we use the term *heuristics* in this paper, we always refer to primal heuristics for mixed integer programming. Further, with a slight abuse of notation, we will use the abbreviation MIP for mixed integer programming as well as for *a* MIP being a single mixed integer programming instance.

Heuristics can be further classified w. r. t. their specific purpose within the solving process and the techniques they apply, see [10, 25]. In this paper, we focus on rounding and propagation heuristics that are applied inside a complete solver. While rounding heuristics apply rounding strategies to make an LP-feasible solution feasible, propagation heuristics, also known as probing heuristics [1], use domain propagation techniques (see Section 3) to reduce the search space and drive a partially assigned solution towards feasibility. In contrast to diving heuristics [10, 17], propagation heuristics do not solve LPs to ensure feasibility of the linear constraints. Note that, although rounding heuristics are very fast procedures, they can only be applied after the LP relaxation of a given MIP has been solved, which sometimes can already take a long time, compare, e. g., the XXL instances from MIPLIB2010 [26].

The goal of this paper is to introduce a new propagation heuristic, called Shift-and-Propagate, and to compare it against rounding heuristics and two improvement heuristics when used inside a complete MIP solver. Both improvement heuristics employ a light version of propagation. The common feature of all these heuristics is that they do not solve LPs (or even sub-MIPs, as, e. g., RINS [17]) by themselves.

In the spirit of [33], the primal heuristic presented in this paper is intended to be a component of a complete solver rather than a standalone procedure, which makes it different from most other heuristics presented in the literature. For this purpose, it is designed to be a very quick procedure, which might sacrifice success on some instances to achieve a good trade-off between the number of found solutions and average running time. It does not require an LP solution as a starting point and can therefore be applied earlier during the solver's search than many other heuristics.

The remainder of the paper is organized as follows. We will first recall some rounding and improvement heuristics known from the literature. All of them are already implemented in SCIP—the solver that we will use for our computational experiments. Then, we give an overview of MIP domain propagation techniques. After that, we introduce key ideas and implementation details of the Shift-and-Propagate heuristic, including individual discussions on two of its main components: the shifting value selection and the variable ordering. Finally, computational results are presented.

2 Rounding and improvement heuristics

Primal heuristics, in particular those that only employ computationally inexpensive procedures such as rounding and logical deductions (propagation), are an important component of state-of-the-art MIP solvers. In this section, we describe rounding heuristics and two simple improvement heuristics that are known from the literature [4, 33] and will serve as a comparison for our newly proposed heuristic later.

2.1 Rounding heuristics

The goal of rounding heuristics is to convert an LP-feasible solution z^* into a feasible solution for the MIP by applying rounding strategies to the set of fractional variables $F := \{j \in \mathcal{I} : z_j^* \notin \mathbb{Z}\}$.

Most of the rounding heuristics described in this section use the notion of *up-* and *down-locks*. For a MIP, we call the number of positive coefficients $\overline{\Lambda}_j := |\{i\colon a_{ij} > 0\}|$ the *up-locks* of the variable x_j; the number of negative coefficients is called the *down-locks* $\underline{\Lambda}_j$ of x_j, see, e.g., [10]. This is motivated by the following observation: let z^* be an LP-feasible solution. If we shift a variable x_j up or down, at most $\overline{\Lambda}_j$ or $\underline{\Lambda}_j$ constraints can be violated, respectively. If a variable $j \in F$ satisfies $\overline{\Lambda}_j = 0$ or $\underline{\Lambda}_j = 0$, it can be trivially rounded up or down, respectively: Let z^* be an LP-feasible solution and $\hat{z}$ be the solution obtained by rounding j into the direction of zero locks. Then it holds for all constraints that $a_i^T \hat{z} \leq a_i^T z^* \leq b_i$ and hence $\hat{z}$ is LP-feasible.

The Simple Rounding heuristic [10] uses this to produce feasible solutions by rounding variables $z_j \leftarrow \lceil z_j^* \rceil$ if $\overline{\Lambda}_j = 0$, or $z_j \leftarrow \lfloor z_j^* \rfloor$ if $\underline{\Lambda}_j = 0$. It will terminate either with a feasible solution or after a variable $j \in F$ was found with $\overline{\Lambda}_j > 0$ and $\underline{\Lambda}_j > 0$.

ZI Round [33] reduces the *integer infeasibility* of an LP-feasible solution step-by-step by shifting fractional values towards integrality, but not necessarily rounding them. For each fractional variable $j \in F$, the heuristic calculates bounds for both possible rounding directions of z_j^* such that the obtained solution stays LP-feasible. z_j^* is shifted by the corresponding bound into the direction which reduces the *fractionality* $\min\{z_j^* - \lfloor z_j^* \rfloor, \lceil z_j^* \rceil - z_j^*\}$ most. The set of fractional variables might be processed several times by ZI Round. It either terminates with a feasible solution or aborts if the integer infeasibility could not be decreased anymore or if a predefined iteration limit has been reached.

In contrast to Simple Rounding and ZI Round, Rounding [10] also performs roundings, which potentially lead to a violation of some linear constraints, trying to recover from this infeasibility by further roundings later on. The solutions that can be found by Rounding are a superset of the ones that can be found by Simple Rounding. Like Simple Rounding, the Rounding heuristic takes up- and down-locks of an integer variable with fractional LP solution value z_j^* into account. As long as no linear constraint is violated, the algorithm iterates over the fractional variables and applies a rounding into the direction of fewer locks, updating the *activities* Az of the LP rows after each step, with z being the partially rounded LP solution. If there is a violated linear constraint, hence $a_i^T z > b_i$ for some i, the heuristic will try to find a fractional variable that can be rounded in a direction such that the violation of the constraint is decreased, using the number of up- and down-locks as a tie breaker. If no rounding can decrease the violation of the constraint, the procedure aborts.

2.2 Improvement heuristics

In addition to the described rounding heuristics, we will also employ two "natural" improvement heuristics, called 1-Opt [2] and 2-Opt [25].

Taking a MIP-solution z as input, 1-Opt determines for every integer variable $j \in \mathcal{I}$ the shift $\delta_j \in \mathbb{Z}$ with maximum $|\delta_j|$ such that $c_j \delta_j \leq 0$ and $a_i^T x + a_{ij}\delta_j \leq b_i$ for all i, hence feasibility is preserved for all constraints. Shifting any variable j with $|\delta_j| \geq 1$ by δ_j will give an improved solution. All found improving shifts are then sorted by increasing value of $c_j \delta_j$ and executed one by one, except a previously executed shift rendered them infeasible. This is a basic version of *constraint propagation*: variable values are inferred from constraint activities.

The 2-Opt heuristic shifts pairs of variables at a time rather than single variables. In the SCIP implementation of 2-Opt [25], integer variables are sorted lexicographically w. r. t. their columns and then grouped together into smaller blocks where the variables appear with nonzero coefficients in at least a predefined ratio of their rows. During its execution, 2-Opt searches within every such block for variable pairs $\{j_1, j_2\} \subseteq \mathcal{I}$ which allow a shift $\delta_1, \delta_2 = \pm\delta_1$ improving the objective $(c_1\delta_1 + c_2\delta_2 < 0)$ and maintaining the feasibility of the solution. Similarly to 1-Opt, all found improving shifts are sorted w. r. t. to the objective improvement and then applied starting with the most improving shift. This approach can easily be extended to a k-Opt or Lin-Kernighan-like [27] heuristic.

Combining either of these improvement heuristics with any of the rounding heuristics from the previous section yields a greedy algorithm for MIP. Starting from an LP-feasible solution, all presented rounding algorithms apply strategies that favor feasibility over optimality. If a solution is found, it can then be driven towards optimality by the improvement heuristics until it cannot be further improved by switching single variable values or pairs of them.

This combination is a fast and promising approach to find good solutions early during the solving process, in particular for set covering and packing problems, see also Section 3.2.2. Here, the described rounding heuristics will always yield a solution where all or at least the vast majority of variables are set to 1 (for covering) or 0 (for packing). The 1-Opt heuristic greedily flips variables in the order determined by their objective coefficient until it reaches a local optimum.

There exist, however, MIPs for which even solving the LP relaxation is hard, hence the LP solver is unable to find the required LP-solution within reasonable time and thus, none of the above heuristics can be applied in the described way. Examples include the XXL instances from MIPLIB2010 [26]. Therefore, we propose the Shift-and-Propagate heuristic, which does not depend on a previously found LP-feasible solution.

Before the presentation of the heuristic, we recall some MIP domain propagation procedures.

3 Domain propagation

Domain propagation (see, e.g., [13]) denotes the process of inferring sequences of local domain reductions at the current node of the branch-and-bound tree. The goal is to shrink the size of the current subproblem as much as possible at affordable computational cost. This natural idea is known under many different names, e. g., node preprocessing, bound tightening, range reduction, filtering in different communities such as mathematical programming, constraint programming, satisfiability testing, and artificial intelligence.

In this section, we want to give an overview of existing domain propagation rules for linear constraints. The *domain* of a variable j is the set of values within the (local) lower and upper bounds of j,

$$D_j := \{z \in M : l_j \leq z \leq u_j\}$$

for $M \in \{\mathbb{R}, \mathbb{Z}\}$ depending on the variable type of j. In mixed integer programming, domain reductions typically consist of tightened variable bounds l_j, u_j, hence holes in the interval $[l_j, u_j]$ are not considered. Reductions on variable bounds from linear constraint activity were first established in Brearly et al. [16]. Savelsbergh [32] extended these methods by various probing techniques on binary variables and constraints, while Andersen and Andersen [6] exploited further presolving techniques for linear programming.

For the suggested Shift-and-Propagate heuristic, domain propagation is a crucial step. In this section we review which domain propagation rules are provided by the different constraint types in SCIP. For more information and implementation details, see [2].

Note that domain propagation rules are applied at two different stages of the MIP solving process. First, they are applied during preprocessing before branch-and-bound is started, in which case the deductions hold globally for the problem. Second, they are used locally at nodes within the branch-and-bound tree to infer reductions from the branching decisions. In the following, we focus on local propagation during search, and use l, u for local bounds at a branch-and-bound node.

3.1 General linear constraints

Taking into account the domains of all variables which are involved in a particular linear constraint, its *minimum and maximum activity* [16] $\underline{\alpha}$ and $\overline{\alpha}$ are defined by

$$\underline{\alpha} := \min\left\{a_i^T x \colon l \le x \le u\right\} \text{ and } \overline{\alpha} := \max\left\{a_i^T x \colon l \le x \le u\right\}.$$

In the same way, we obtain the *minimum and maximum residual activity* for variable j,

$$\underline{\alpha}_j := \min\left\{a_i^T x - a_{ij}x_j \colon l \le x \le u\right\} \text{ and}$$
$$\overline{\alpha}_j := \max\left\{a_i^T x - a_{ij}x_j \colon l \le x \le u\right\} \text{ resp.,}$$

by excluding the contribution of variable j. By these definitions, it is possible to deduce bounds on the variables that appear in a certain constraint. For a positive row coefficient a_{ij} of variable j in row i, it holds that

$$x_j \le \frac{b_i - \underline{\alpha}_j}{a_{ij}}$$

by which the variable bound can be tightened whenever $u_j > \frac{b_i - \underline{\alpha}_j}{a_{ij}}$. If j is an integer variable, the new upper bound can be rounded down. An analogous inference rule holds for lower bounds in the case of negative coefficients.

Besides the tightening of variable domains, the notion of minimum and maximum activities can also be used to detect the local redundancy or infeasibility of a constraint [16].

3.2 Domain propagation for special classes of linear constraints

For linear constraints of special form, there often exist stronger propagation algorithms. In his thesis [2], Achterberg described special techniques for the following linear constraint classes: variable bounds, knapsack, set covering, set partitioning, and set packing.

Of course, all of them could be propagated by algorithms for general linear constraints, but their special structure allows for a more efficient implementation of the propagation routines. In this section, we briefly describe propagation algorithms for knapsack and set covering constraints.

3.2.1 Knapsack constraints

For *knapsack* constraints, all involved variables have to be *binary* variables and all coefficients $a_{ij} =: w_j$ and the right hand side $\overline{b}$ have to be nonnegative integers, called the *weights* and the capacity, respectively. In [2], a transformation for a more general class of constraints into this special type was shown. With the properties of knapsack constraints, the propagation routines are tuned by the use of integer—instead of floating point—arithmetic. Second to that, the only reduction to be performed is fixing a variable j to 0 if the weighted sum of the variables fixed to 1 and the weight w_j together exceed the right hand side. Let

$$K^* := \{j \in \mathcal{I} \colon w_j > 0 \ \wedge \ x_j \text{ fixed to } 1\}$$

denote the set of variables which are already fixed to 1 and $w^* := \sum_{j \in K^*} w_j$ the sum of their weights. The domain propagation rules then reads

$$(j \text{ unfixed}) \;\wedge\; (w^* + w_j > b) \;\Rightarrow\; (x_j \leftarrow 0)$$

Here, the nonnegativity of the weights and the binary type of all involved variables allows an improved performance by sorting the variables in nonincreasing order of their weights.

SCIP also features methods to extract (negated) *clique-information* about the binary variables of a problem. A (negated) clique is a set of binary variables of which at most one variable can be assigned the value 1 (0) in a feasible solution.

Let therefore $C \subseteq \mathcal{I} \setminus K^* \cup K^0$ denote a negated clique of unfixed variables, $w(C)$ be the sum of weights of variables in C, $j_{\max} := \operatorname{argmax}_{j \in C} w_j$ be a clique variable of maximum weight, and $w_{\min}(C) := w(C) - w_{j_{\max}}$ be the minimum weight of this negated clique in any feasible solution. The following reductions are now possible considering negated clique information:

1. $(w^* + w_{\min}(C) > b) \;\Rightarrow$ (the subproblem is infeasible)
2. $(j \in C \setminus \{j_{\max}\}) \;\wedge\; (w^* + w_{\min}(C) - w_j + w_{j_{\max}} > b) \;\Rightarrow\; (x_j \leftarrow 1)$

3.2.2 Set covering constraints

Set covering constraints are another special class of linear constraints. They have the form

$$x_{j_1} + \cdots + x_{j_k} \geq 1, \text{ for } \{j_1, \ldots j_k\} =: K \subseteq \mathcal{N},$$

where K contains only binary variables. The only domain reduction to be inferred from a set covering constraint is to fix a variable j to 1 if all remaining variables $j' \in K \setminus \{j\}$ have already been fixed to 0. The state-of-the-art algorithm to keep track of the bound changes in this case was introduced in [30]; it is known as *two-watched-literals scheme* and provides a significant speedup in propagation.

4 Shift-And-Propagate

After having reviewed existing inexpensive start heuristics and domain propagation techniques, we now introduce the Shift-and-Propagate heuristic. Recall that solving the root-LP relaxation of a MIP can be very time-consuming. As mentioned before, the purpose of this primal heuristic is finding a feasible MIP solution at the very early stage of the solution process where no information about the root LP solution is available. In addition, it should be computationally cheap, using only domain propagation techniques.

The basic idea is as follows: in each iteration, the heuristic selects an unfixed variable $j \in K$ and a fixing value t_j^* inside the domain of x_j, to which the variable is shifted. Then, domain propagation routines are called for this fixing. If domain propagation detects that fixing $x_j \leftarrow t_j^*$ is infeasible, a one-level backtrack-strategy is applied. Otherwise, the heuristic proceeds with the next unfixed variable. The goal of this heuristic is to find a good start solution, before the root node processing of a MIP solver starts, in particular prior to the first LP being solved. It might then serve as a reference point for improvement heuristics (see Section 2.2) and for inferring further domain reductions (e. g., by propagating the maximum activity of the objective function, see Section 3.1).

The general algorithm is described in Algorithm 1. The main degrees of freedom are the variable selection in line 3, the choice of a promising fixing value (line 4), and the backtrack-strategy including an appropriate stopping criterion (line 7): all of which are discussed in the remainder of this section. We will first discuss different variable orders, then introduce an algorithm to select a best shifting value, and finally present a full version of the Shift-and-Propagate algorithm, including considerations on backtracking. Note that bounds can be changed, in particular variables can be fixed, in line 5 of Algorithm 1.

Algorithm 1: The basic Shift-and-Propagate algorithm

```
   Input  :  MIP problem $P$
   Output :  a feasible solution of $P$, or NULL if search was not successful
 1 $K \leftarrow \mathcal{I}$, $z \leftarrow 0$ ;
 2 while $K \neq \emptyset$ do
 3     Select $j \in K$ ;
 4     Choose $t_j^* \in D_j$ ;
 5     Propagate $D_j \leftarrow \{t_j^*\}$ ;
 6     if propagation detects infeasibility then
 7         Apply backtrack strategy ;
 8     else
 9         $z_j \leftarrow t_j^*$ ;
10         $K \leftarrow K \setminus \{j\}$ ;
11     end
12 end
13 if $z$ is feasible for $P$ then
14     return $z$ ;
15 return NULL;
```

4.1 Implementation details

For the ease of presentation, we assume from now on, w.l.o.g., that all variables have a lower bound of 0, since otherwise a suitable problem transformation can be applied. Variables with finite lower bound are shifted, variables with infinite lower bound but finite upper bound are negated (and shifted), and free variables are decomposed into a negative and a positive part.

Shift-and-Propagate starts with the *zero-assignment* $z \leftarrow 0$ which respects all variable bounds[1]. The row activity of all rows is zero as well. Hence, an assignment is feasible for a row, if and only if it has a positive right hand side $b_i \geq 0$. Subsequent fixing steps $D_j \leftarrow t_j^*$ are then processed as shifts, which only affect those rows in which the variable j is involved in. Instead of updating the row activity explicitly after every shift, activities are maintained implicitly by changing the right hand side.

Continuous variables are not handled directly by the heuristic but treated as row slacks. For each row, we consider a relaxation by subtracting the minimum activity of the continuous variables from the right hand side. If the heuristic finds a solution on the integer variables in this transformed space, a final LP with all integer variables fixed to their heuristic values is solved to obtain values for the continuous variables. There are two advantages of this final LP compared to solving an LP relaxation beforehand. First, the final LP will be smaller and often significantly easier to solve, since all integer variables are fixed and the fixings have been propagated. In

[1]in principle, Shift-and-Propagate could be started from any solution within the variable bounds. In particular, a start assignment could be chosen that is promising either w.r.t. feasibility or w.r.t. objective function value.

particular, for pure IPs, this stage is completely omitted. Second, it is only employed when the heuristic found a consistent assignment for the integer variables and merely the continuous ones need to be adopted.

As for the variable order, the heuristic sorts the variables nonincreasingly w. r. t. the initial number of violated rows they appear in,

$$|\{i \colon a_{ij} \neq 0 \text{ and } b_i < 0\}| \,.$$

We also tested a different variable order using the *importance* of a variable column, which we define as

$$\sum_{i=1}^{m} |a_{ij}| + |\{i \colon a_{ij} \neq 0\}| \,.$$

We compare different variable orders in Section 5.

The choice of a fixing value for the selected variable is obtained by a *best shift selection* which is based on the feasibility state of the row w. r. t. the current assignment. Therefore, we keep track of how many *violated* rows can be made feasible and vice versa by a certain shift.

Definition 4.1. *For an LP-row $i : a_i^T x \leq b_i$ and an unfixed variable $j \in K$, we define*

$$\begin{aligned} \Psi_i^j \,:\, D_j &\to \{-1, 0, 1\} \\ t &\mapsto \begin{cases} 1, & b_i \geq 0, \text{ and } b_i - a_{ij} \cdot t < 0 \\ -1, & b_i < 0, \text{ and } b_i - a_{ij} \cdot t \geq 0 \\ 0, & \text{else} \end{cases} \end{aligned}$$

the row violation function *of row i. The row violation functions of all rows sum up to*

$$\begin{aligned} \Psi^j \,:\, D_j &\to \mathbb{Z} \\ t &\mapsto \sum_{i=1}^{m} \Psi_i^j(t) \end{aligned}$$

the row violation sum function *of j. For a particular $t \in D_j$ we call $\Psi^j(t)$ the* violation balance *of t.*

For every row violation function, it holds that $\Psi_i^j(0) = 0$ and that it changes its value at most once on D_j. The row violation balance $\Psi^j(t)$ is a measure of how the overall feasibility of a partial assignment changes by a particular shift t of the variable. A negative value means that more rows will be made feasible than infeasible by shifting variable j by a value of $t \in D_j$. The function Ψ^j is a step function with at most M_j steps, M_j being the number of nonzeros for variable j. The best shift selection Algorithm 2 searches for a value t^* which minimizes the violation balance,

$$t^* \leftarrow \operatorname*{argmin}_{t \in D_j} \Psi^j(t).$$

Since $\Psi^j(0) = 0$, the heuristic will prefer a shifting value different from 0 if and only if there is a value $t' > 0$ with $\Psi^j(t') < 0$, i. e., if t' is able to really reduce the number of currently violated rows.

In Algorithm 2, the row violation functions are interpreted as tuples

$$(t_i, \Psi_i^j(t_i)) \in D_j \times \{-1, 1\}$$

Algorithm 2: bestShift(P, j, D_j)

```
   Input  : MIP $P$, integer variable $j \in \mathcal{I}$ with domain $D_j$
   Output : Best shift $t^*$ for $j$
 1 $Q \leftarrow \emptyset$ ;
   // collect row violation functions of the variable
 2 foreach row $i$ with $a_{ij} \neq 0$ do
 3     if $b_i < 0$ and $a_{ij} < 0$ then
 4         $t \leftarrow \lceil \frac{b_i}{a_{ij}} \rceil$ ;     // minimum shift of variable $j$ to make row $i$ feasible
 5         if $t \in D_j$ then $Q \leftarrow Q \cup (t, -1)$;
 6     else if $b_i \geq 0$ and $a_{ij} > 0$ then
 7         $t \leftarrow \lfloor \frac{b_i}{a_{ij}} \rfloor + 1$ ;     // minimum shift of variable $j$ to violate row $i$
 8         if $t \in D_j$ then $Q \leftarrow Q \cup (t, 1)$;
 9     end
10 end
11 if $Q = \emptyset$ then return $t^* = 0$ ;
12 $\sigma \leftarrow 0$, $t^* \leftarrow 0$, $t_{\text{before}} \leftarrow 0$, $\Psi^* \leftarrow 0$ ;
   // summation in the right order gives the row violation balance
13 foreach $(t_i, \Psi_i^j(t_i)) \in Q$ in nondecreasing order of $t_i$ do
14     if $t_i > t_{before}$ and $\sigma < \Psi^*$ then
15         $\Psi^* \leftarrow \sigma$, $t^* \leftarrow t_{\text{before}}$ ;
16     $t_{\text{before}} \leftarrow t_i$ ;
17     $\sigma \leftarrow \sigma + \Psi_i^j(t_i)$ ;
18 end
19 if $\sigma < \Psi^*$ then $t^* \leftarrow t_{\text{before}}$ ;     // takes highest step value into account
20 return $t^*$ ;
```

where t_i is the smallest (always positive) value for which the row changes its feasibility state, and $\Psi_i^j(t_i) \in \{-1, 1\}$, depending on the kind of change in the feasibility state. The t_i is called the *step value*. An empty tuple means that no value in the variable domain alters the feasibility state of the row. Algorithm 2 collects the row violation functions for all rows i with nonempty tuples $(t_i, \Psi_i^j(t_i))$ and sorts them in nondecreasing order of the t_i. Since different rows $i \neq i'$ might have the same step value $t_i = t_{i'}$ the sum of row violation functions processed so far is stored in the variable σ and yields the row violation balance $\Psi^j(t_{\text{before}})$ when $t_i \gneq t_{\text{before}}$, i. e., two subsequent step values are different. The method returns a value $t^* \in D_j$ minimizing Ψ^j. The best shift selection could also be used for continuous variables, omitting the rounding in lines 4 and 7 of Algorithm 2. A different implementation of Shift-and-Propagate might keep continuous variables as part of the transformed problem instead of relaxing them, and use the best shift selection to find solution values for all variables.

The complete Shift-and-Propagate procedure is shown in Algorithm 3. It uses the best shift selection as a subroutine to select a promising fixing value for some variable j from the set of unfixed variables K. The fixing is then performed and propagated. An empty domain of some variable after the propagation will cause the algorithm to apply a one-level backtrack. If the attempted shift value was one of the variable bounds, $t_j^* \in \{l_j, u_j\}$, the variable domain is tightened by $D_j \leftarrow D_j \setminus \{t^*\}$ and repropagated. If the repropagation of the shrunk domain also detects an empty domain, the execution method of the heuristic is stopped (line 15). Two cases remain to be handled: either the original shifting value was in the inner of the variable's domain or the propagation of the shrunk domain did not lead to infeasibility. In both cases, the variable is assigned to the set of suspicious variables S in line 17 and thus removed from the set of unfixed variables K (line 21). This ensures that the best shift selection is only performed at most once for every variable. In our implementation, we use a limit on the number of backtracks performed; if it is exceeded, the heuristic stops.

We conclude this section with a small example MIP, for which we describe the course of the algorithm in detail.

Example 4.2. *Let P_{ex} be the following MIP with three integer variables $\mathcal{I} = \{1, 2, 3\}$ and $m = 3$ constraints.*

$$\begin{aligned} \text{Minimize} \quad 0 & \\ 2x_1 - x_2 - x_3 &\leq 1 \\ -x_1 - x_2 &\leq -2 \\ -3x_1 + x_3 &\leq -3 \\ x_1, x_2, x_3 &\in \{0, 1, 2\} \end{aligned}$$

P_{ex} is a pure feasibility problem. The zero assignment is not feasible, because both the second and the third row have a negative right hand side.

The heuristic starts with x_1, x_2, and x_3 appearing in two, one and one violated row, respectively, so the sorting will keep the variables in place and select x_1 to start with.

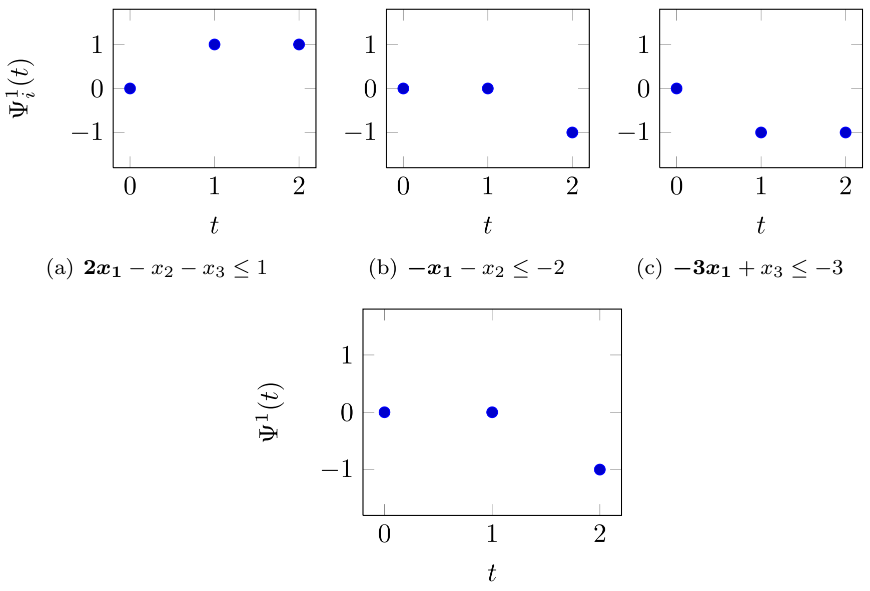
The row violation functions and the row violation sum function for variable x_1 from the example P_{ex} are depicted in Figure 1. The first row will be made (temporarily) infeasible by a shift of 1 or 2, whereas the third will be made feasible by a shift of either 1 or 2. The second row, however, requires a shift of 2 to be made feasible. Therefore, the algorithm will select $t_1^* = 2$ as shifting value for x_1 because it minimizes the violation balance $\Psi^1(2) = -1 < 0 = \Psi^1(1)$. Fixing the domain of x_1 to the single value $D_1 \leftarrow \{2\}$ and subtracting the contribution from the right hand side yields a reduced problem

$$\begin{aligned} -x_2 - x_3 &\leq -3 \\ -x_2 &\leq 0 \\ x_3 &\leq 3 \\ x_2, x_3 &\in \{0, 1, 2\} \end{aligned}$$

The feasibility state of every row has changed by the shift of variable 1. The second and third row are trivially satisfied in this reduced problem. The propagation from Section 3 applied to the first row leads to further reductions: 0 is excluded from the variable domains $D_2 = D_3 = \{1, 2\}$. Since the heuristic requires lower bounds 0 for every variable, the variables are formally replaced by $\bar{x}_{\{2,3\}} \leftarrow x_{\{2,3\}} - 1$, which leads to a changed right hand side vector. The first row now reads

$$\begin{aligned} -\bar{x}_2 - \bar{x}_3 &\leq -1 \\ \bar{x}_2, \bar{x}_3 &\in \{0, 1\}. \end{aligned}$$

A shift of variable $\bar{x}_2$ by 1 is selected by Algorithm 2 and finally makes the row feasible again. Since the right hand side $\bar{b}$ is now nonnegative, the remaining variable $\bar{x}_3$ can be set to 0 without violating a row. Untransforming all bound changes, the heuristic finds the solution $z_{P_{ex}} = (2, 2, 1)$.



(a) $\boldsymbol{2x_1} - x_2 - x_3 \leq 1$ (b) $\boldsymbol{-x_1} - x_2 \leq -2$ (c) $\boldsymbol{-3x_1} + x_3 \leq -3$

Figure 1: The row violation functions for variable x_1 from P_{ex} for every row (smaller pictures) and the resulting row violation sum function $\Psi^1(t)$

5 Computational results

In this section, we compare different variable orders and evaluate the performance of the presented Shift-and-Propagate heuristic. We use SCIP 3.0 with SoPlex 1.7.0 [39] as the underlying LP-solver. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSuse 12.1 with a gcc 4.6.2 compiler. Hyperthreading and Turboboost were disabled. In all experiments, we ran only one job per node to avoid random noise in the measured running time that might be caused by cache-misses if multiple processes share common resources.

For all benchmarks, we chose a test set containing 168 MIP instances from the three publicly available libraries MIPLIB3.0 [14], MIPLIB2003 [5], and MIPLIB2010 [26]. We excluded the three instances ash608gpia-3col, enlight14, and ns1766074 which are infeasible and ex9 which can be solved to optimality by the root LP.

As a measure for the quality of a solution z for an instance P, we consider a gap function defined as follows: let z_{opt} be an optimal (or best known) solution for P. The gap is defined as

$$\Delta(z) := \begin{cases} 0, & \text{if } c^T z = c^T z_{\text{opt}}, \\ \min\{1, \frac{|c^T z - c^T z_{\text{opt}}|}{\max\{|c^T z|, |c^T z_{\text{opt}}|\}}\}, & \text{if } z \neq \emptyset, \\ 1, & \text{else} \end{cases}$$

By the choice of $\Delta(\cdot)$, every solution has a gap $\Delta(z) \leq 100\%$. If no solution was found, the gap is worst possible, $\Delta(\emptyset) = 100\%$.

In Section 4, two variable orders are proposed: The first order based on the importance of the variables, and the second one based on the initial number of infeasible rows of a variable. In a first experiment, we compared these two different approaches and a random sorting. The importance-based and the violation-based sorting are addressed by the symbols $|\cdot|$ and $|\{b < 0\}|$,

Algorithm 3: Shift-and-Propagate

```
Input  : A MIP $P$ with constraint matrix $A$ and right hand side $b$
Output : a feasible solution $z$ for $P$, or NULL
1  $P_{\text{orig}} \leftarrow P$ ;
2  Relax continuous variables s. t. $\mathcal{C} = \emptyset$;
3  $K \leftarrow \mathcal{I}$;
4  Sort $K$ w. r. t. to the number of initially violated rows ;
5  $S \leftarrow \emptyset$, $z \leftarrow 0$ ;      // $S$ contains variables whose fixing led to cutoffs
6  while $K \neq \emptyset$ and $b \ngeq 0$ do
7      Transform $P$ s. t. $l = 0$ ;      // has to be ensured each round
8      Select next $j \in K$ ;
9      $t_j^* \leftarrow$ bestShift($P, j, D_j$) ;
10     propagate $D_j \leftarrow \{t_j^*\}$ ;
11     if $\exists k \in K\colon D_k = \emptyset$ then
12         backtrack ;      // if empty domain, undo propagation, unfix $j$
13         if $t_j^* \in \{l_j, u_j\}$ then
14             propagate $D_j \leftarrow D_j \setminus \{t_j^*\}$ ;
15             if $\exists k \in K\colon D_k = \emptyset$ then return NULL;
16         end
17         $S \leftarrow S \cup \{j\}$;
18     else
19         $z_j \leftarrow t_j^*$ ;
20     end
21     $K \leftarrow K \setminus (\{j \in \mathcal{I}\colon |D_j| = 1\} \cup S)$ ;
22 end
23 if $b \geq 0$ then
24     foreach $j \in K \cup S$ do
25         $z_j \leftarrow 0$ ;      // all unfixed variables are set to 0
26     end
27     $z_{\text{orig}} \leftarrow$ retransform $z$ ;      // undo all transformation steps from line 7
28     if $P_{orig}$ contains continuous variables then
29         Fix integer variables to $z_{\text{orig}}$ and solve the remaining LP ;
30         if LP infeasible then return NULL;
31     end
32     return $z_{\text{orig}}$ ;
33 end
34 return NULL;
```

Table 1: Overview of the different variable sortings

setting	$\bar{\Delta}$ (%)	#sols	t_{heur} (s)
$\lvert\{b<0\}\rvert\downarrow$	84.73	88	1.21
$\lvert\cdot\rvert\downarrow$	84.98	84	1.20
random	85.95	84	1.17
$\lvert\{b<0\}\rvert\uparrow$	87.66	83	1.17
$\lvert\cdot\rvert\uparrow$	86.85	81	1.20
Best	78.61	102	1.11

respectively, together with an arrow indicating the sense in which the variables are processed: $\downarrow$ for nonincreasing and $\uparrow$ for nondecreasing. The symbol $|\{b < 0\}| \downarrow$, e. g., stands for "sorting w. r. t. the number of initially violated rows, nonincreasing".

For all settings, we disabled cutting plane routines in order to avoid the LP relaxation to be re-solved and to make sure that the primal solutions were indeed obtained by the Shift-and-Propagate heuristic.

The performance of the five variable sortings is shown in Table 1 (see Table 3 in the appendix to see results on particular instances) and compared with respect to three criteria: The average solution gap $\bar{\Delta}$, the absolute number of solutions obtained (#sols), and the geometric mean of the time spent by the heuristic t_{heur} (s). The average gap ranges from 84.7% to 87.7% and the number of found solutions differs by up to 7. The setting $|\{b < 0\}| \downarrow$ scores best regarding the number of found solutions and also the average gap, but is the slowest setting w. r. t. its geometric mean time.

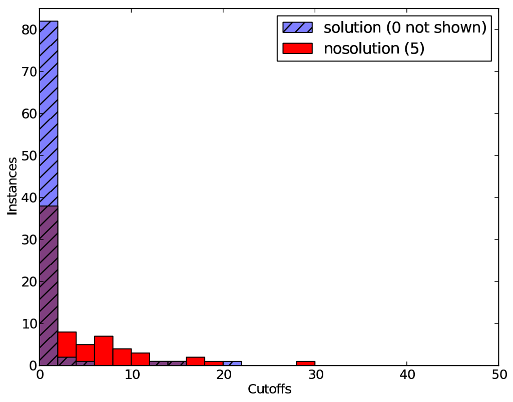
A closer look at the results on specific instances shows more variety than the overall results: A feasible solution for rail507 was found with two of the five settings, namely $|\cdot| \uparrow$ and $|\{b < 0\}| \uparrow$, which took 13.78 s and 14.42 s, respectively. This seems moderate compared to other settings, which terminate unsuccessfully after up to 3600 s (the time limit) with $|\{b < 0\}| \downarrow$. This is particularly undesirable knowing that the LP solve only takes 10 s on this instance.

Reasons for long running times on some instances are the propagation itself, combined with a huge number of backtracking operations due to decisions which lead to cutoffs, and/or the number of iterations of the internal LP solve, which can take itself a long time if the number of continuous variables is large.

Table 1 has a sixth setting named Best presenting the union of the five settings, choosing the quickest setting which reaches the best gap in an instance. The total number of instances for which Best succeeds is 102, which is 14 more than the best real setting in this respect. Best reaches a gap of 78.61 which is considerably better compared to the real settings.

Figure 2 presents histograms of how many cutoffs were effectively produced until a feasible solution was found (blue bar), or until the heuristic terminated without a solution (red bar). Some instances are not presented in the table for reasons of normalization of the horizontal axis, those numbers are indicated in the legend. We show these histograms for two different settings: $|\{b < 0\}| \downarrow$, and random below. These charts present two facts: The number of instances on which the heuristic finds a solution without or with nearly no backtracking (0 or 1 cutoff, the leftmost bar) is higher for the $|\{b < 0\}| \downarrow$ than for the random order by almost 10 instances. The bars for the informed sorting lie underneath those for the random sorting, indicating that the sorting method produces less cutoffs on the average.

The sorting method $|\{b < 0\}| \downarrow$ has the advantage to produce more feasible solutions directly or with at most one cutoff on this testset than the other methods. We will use this method for



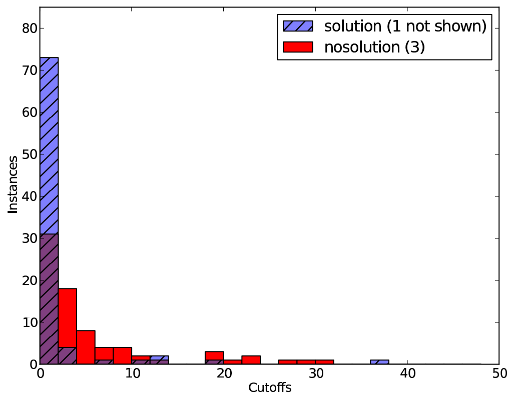


Figure 2: Distribution of the instances over the number of cutoffs Shift-and-Propagate produced during its search for two different variable sortings: $|\{b < 0\}| \downarrow$ and random

the next comparison experiment. Furthermore, the fact that solutions were only obtained after a small number of cutoffs suggest that a small absolute limit on the number of cutoffs as, e. g., 10, will only slightly decrease the number of produced solutions but trigger a quicker termination on a significant amount of instances for which the heuristic is not successful. In particular, for the instance rail507, using a cutoff limit of 10 leads to a running time of 8.25 s instead of hitting the time limit.

The second experimental setup compares the number of found solutions and the average solution quality obtained with the rounding and improvement heuristics from Section 2 to the quality of the Shift-and-Propagate heuristic after processing the root node. The first setting *RandI* uses only rounding and improvement heuristics from Section 2. The second setting *SandP* uses Shift-and-Propagate as only primal heuristic. A third setting *Both* is a combination of those heuristics. Neither Shift-and-Propagate nor rounding heuristics take the objective function into account, but improvement heuristics do. It can therefore not be expected that *SandP* outperforms *RandI* in terms of solution quality. The hope is rather that it is beneficial in terms of found solutions and that we can observe a combined effect in the *Both* setting.

All instances are presented in Table 4 in the appendix, together with their optimal or best known solution. For each setting, the table depicts the achieved primal bound as well as the solving time (which includes presolving and the processing of the root node) and the time spent on heuristics.

Table 2: Results from the root node solve for three different settings

setting	$\bar{\Delta}$ (%)	#sols	t (s)
SandP	85.05	85	2.40
RandI	81.40	60	2.36
Both	70.27	96	2.41

In Table 2, we present the results obtained for these different settings. The column $\bar{\Delta}$ shows the average gap for all instances from the test set. In the column #sols, the total number of instances is shown for which a solution was found. The last column shows the geometric mean of the overall solving time. The table shows that the use of Shift-and-Propagate leads to a primal feasible solution on 85 instances, whereas rounding and improvement heuristics alone can only contribute solutions to 60 instances of the problem set. Both settings combined find a solution on 96 instances or 60 % more than without Shift-and-Propagate. The average gap obtained by SandP was 85.05%, compared to an average gap of 81.40% when using RandI. Here, instances for which no solution could be found are accounted for by a gap of 100%. The best result w. r. t. the average gap is the combined setting, Both, which finishes the root node with an average gap of 70.27%. The increase of the overall geometric mean solving time is 0.05 s or 2.1%, from 2.36 s with RandI heuristics alone, to 2.41 for the setting Both.

Figure 3 shows the distribution of solution qualities over all instances from 0 to 100 in steps of 20. The "none" bar gives the number of instances for which the corresponding setting did not provide any solution. One can see that the solution quality on individual instances is often worse than for rounding and improvement heuristics which produce more solutions with a gap between 0 and 40 than Shift-and-Propagate. The third setting combines both the higher number of solutions found with Shift-and-Propagate and the better gap obtained with rounding and improvement heuristics.

The experiment reveals that Shift-and-Propagate outperforms the existing rounding and improvement heuristics in terms of the number of found solutions, but is inferior w. r. t. the objective quality. For both measures, the best performance was achieved by a combined setting, which increased the number of found solutions by 60%, decreased the average gap by 14% and took only 2% more time, compared to a setting that uses only rounding and improvement heuristics. As a motivation for Shift-and-Propagate, we mentioned the XXL instances from MIPLIB2010. Nine of these instances could be loaded into SCIP given the limitation of 48 GB RAM. Using a time limit of 24 hours, Shift-and-Propagate found a solution for five instances, the rounding and improvement heuristics only for three. Given the small size of the test set, this result is of course of limited conclusiveness.

We conclude that Shift-and-Propagate is a valuable extension of the primal heuristic portfolio of SCIP. It is by now employed by default in SCIP; the complete source code can be obtained at http://scip.zib.de.

6 Conclusions

In this paper, we have introduced Shift-and-Propagate, a pre-root primal heuristic for mixed integer programming that alternately shifts variables to promising fixing values in order to make linear constraints feasible, and propagates the variables fixings to get tighter domains for choosing the fixing values. It differs from other recently proposed MIP heuristics in that it is does not require a feasible LP solution as a starting point and that it is specifically designed as a quick

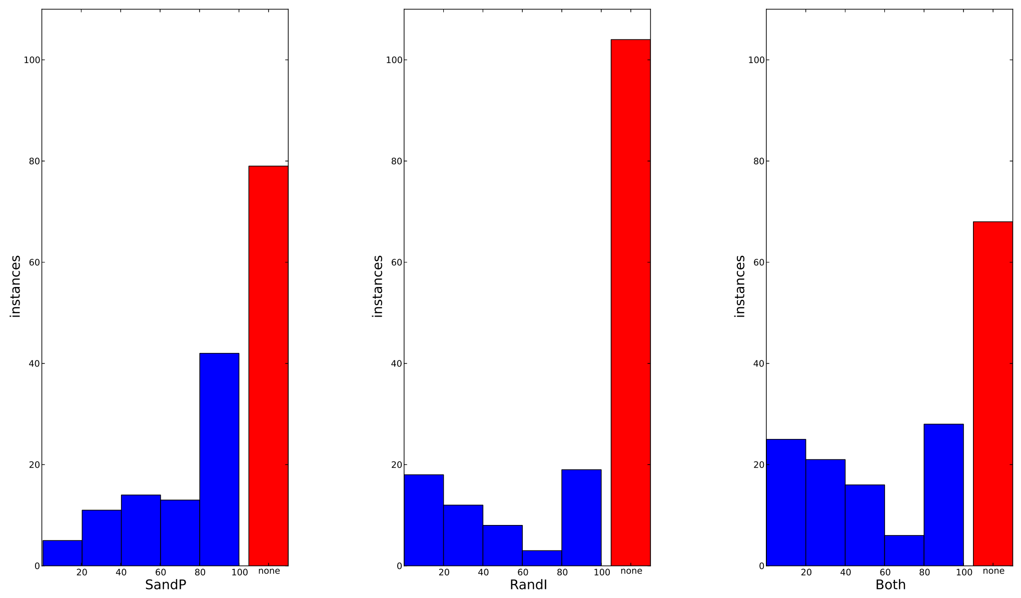


Figure 3: Distribution of solution quality after root node processing

start heuristic inside a complete solver. The main contributions are the presentation of a quick and reliable procedure to select a shifting variable and an analysis of different variable orders, which cover the two main degrees of freedom in the heuristic.

We conducted two experiments, which revealed that Shift-and-Propagate alone can find solutions on more instances than three rounding heuristics and two combinatorial improvement heuristics together. Combining the existing rounding and improvement heuristics with Shift-and-Propagate increases the number of instances on which a feasible solution is found during the root node by 60 %. Furthermore, the average gap to the optimal or best known solution could be significantly reduced.

The computational results indicate, that the pre-root heuristic Shift-and-Propagate nicely complements existing LP-based root node heuristics; it is now one of the default heuristics applied in SCIP.

Acknowledgements

Many thanks to Christina Burt for her valuable comments.

References

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

[2] Achterberg, T.: Constraint integer programming. Ph.D. thesis, TU Berlin (2007)

[3] Achterberg, T., Berthold, T.: Improving the feasibility pump. Discrete Optimization **Special Issue 4**(1), 77–86 (2007)

[4] Achterberg, T., Berthold, T., Hendel, G.: Rounding and propagation heuristics for mixed integer programming. In: D. Klatte, H.J. Lüthi, K. Schmedders (eds.) Operations Research Proceedings 2011, pp. 71–76. Springer Berlin Heidelberg (2012)

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

[6] Andersen, E., Andersen, K.: Presolving in linear programming. Mathematical programming **71**, 221–245 (1995)

[7] Balas, E., Ceria, S., Dawande, M., Margot, F., Pataki, G.: Octane: A new heuristic for pure 0-1 programs. Operations Research **49** (2001)

[8] Balas, E., Schmieta, S., Wallace, C.: Pivot and shift - a mixed integer programming heuristic. Discrete Optimization **1**(1), 3–12 (2004)

[9] Bertacco, L., Fischetti, M., Lodi, A.: A feasibility pump heuristic for general mixed-integer problems. Discrete Optimization **Special Issue 4**(1), 77–86 (2007)

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

[11] Berthold, T.: Heuristics of the branch-cut-and-price-framework SCIP. In: J. Kalcsics, S. Nickel (eds.) Operations Research Proceedings 2007, pp. 31–36. Springer-Verlag (2008)

[12] Berthold, T., Feydy, T., Stuckey, P.J.: Rapid learning for binary programs. In: A. Lodi, M. Milano, P. Toth (eds.) Proc. of CPAIOR 2010, *LNCS*, vol. 6140, pp. 51–55. Springer (2010)

[13] Bessiere, C.: Constraint propagation. In: F. Rossi, P. van Beek, T. Walsh (eds.) Handbook of Constraint Programming, chap. 3, pp. 29–83. Elsevier (2006)

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

[15] Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice – closing the gap. In: M. Powell, S. Scholtes (eds.) Systems Modelling and Optimization: Methods, Theory, and Applications, pp. 19–49. Kluwer Academic Publisher (2000)

[16] Brearley, A., Mitra, G., Williams, H.: Analysis of mathematical programming problems prior to applying the simplex algorithm. Mathematical Programming **8**, 54–83 (1975)

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

[18] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming A **104**(1), 91–104 (2005)

[19] Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Mathematical Programming C **1**, 201–222 (2009)

[20] Ghosh, S.: DINS, a MIP improvement heuristic. In: M. Fischetti, D.P. Williamson (eds.) Integer Programming and Combinatorial Optimization (IPCO 2007), *LNCS*, vol. 4513, pp. 310–323 (2007)

[21] Glover, F., Laguna, M.: General purpose heuristics for integer programming – part I. Journal of Heuristics **2**(4), 343–358 (1997)

[22] Glover, F., Laguna, M.: General purpose heuristics for integer programming – part II. Journal of Heuristics **3**(2), 161–179 (1997)

[23] Glover, F., Løkketangen, A., Woodruff, D.L.: Scatter search to generate diverse MIP solutions. OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research (2000)

[24] Hansen, P., Mladenović, N., Urošević, D.: Variable neighborhood search and local branching. Computers and Operations Research **33**(10), 3034–3045 (2006)

[25] Hendel, G.: New rounding and propagation heuristics for mixed integer programming. Bachelor's thesis, TU Berlin (2011)

[26] 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)

[27] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the travelling-salesman problem. Operations Research **21**, 498–516 (1973)

[28] Løkketangen, A.: Heuristics for 0-1 mixed integer programming. Handbook of Applied Optimization (2002)

[29] Løkketangen, A., Glover, F.: Solving zero/one mixed integer programming problems using tabu search. European Journal of Operations Research **106**, 624–658 (1998)

[30] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: ANNUAL ACM IEEE DESIGN AUTOMATION CONFERENCE, pp. 530–535. ACM (2001)

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

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

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

[34] COIN-OR branch-and-cut MIP solver. https://projects.coin-or.org/Cbc

[35] FICO Xpress-Optimizer. http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx

[36] GUROBI Optimizer. http://www.gurobi.com/products/gurobi-optimizer/gurobi-overview

[37] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

[38] SCIP. Solving Constraint Integer Programs. http://scip.zib.de/

[39] SoPlex. An open source LP solver implementing the revised simplex algorithm. http://soplex.zib.de/

[40] SYMPHONY. development home page. https://projects.coin-or.org/SYMPHONY

Appendix

Table 3: Results of different variable sortings w. r. t. to the primal bound and the heuristic time. Bold face indicates better solutions values, hence the setting chosen for the oracle setting Best in Table 1 which prefers lower gaps and uses the time as a tie-breaker.

	$\vert\{b<0\}\vert\downarrow$		$\vert\cdot\vert\downarrow$		random		$\vert\{b<0\}\vert\uparrow$		$\vert\cdot\vert\uparrow$	
Problem Name	c^Tz	t (s)	c^Tz	t (s)	c^Tz	t (s)	c^Tz	t (s)	c^Tz	t (s)
10teams	–	0.02	–	0.02	–	0.01	–	0.01	–	0.02
30n20b8	–	0.04	–	0.19	–	0.09	–	0.11	–	0.03
a1c1s1	–	0.03	–	0.05	–	0.04	–	0.03	–	0.02
acc-tight5	–	0.00	–	0.01	–	0.01	–	0.01	–	0.01
aflow30a	4606.0	0.01	**4280.0**	0.01	–	0.01	4606.0	0.01	4606.0	0.01
aflow40b	8300.0	0.04	**7672.0**	0.02	–	0.02	8300.0	0.04	8300.0	0.04
air03	**6.2e+05**	0.04	6.6e+05	0.04	7.2e+05	0.04	1.2e+06	0.07	1.2e+06	0.07
air04	–	0.03	–	0.03	–	0.03	–	0.03	–	0.04
air05	–	0.03	–	0.03	–	0.08	–	0.04	–	0.04
app1-2	–	9.69	–	10.44	–	9.98	–	9.69	–	10.48
arki001	–	0.01	–	0.04	–	0.01	–	0.01	–	0.03
atlanta-ip	–	0.10	–	0.13	–	0.11	–	0.11	–	0.11
beasleyC3	–	0.01	–	0.01	–	0.02	–	0.01	–	0.01
bell3a	–	0.00	–	0.00	–	0.00	–	0.00	–	0.01
bell5	–	0.00	–	0.00	**9.1e+06**	0.00	–	0.01	–	0.00
bab5	–	2.06	–	2.18	–	1.82	–	0.59	–	0.48
biella1	**9.5e+07**	1.23	1.2e+08	1.24	3.2e+08	0.37	–	0.13	–	0.18
bienst2	–	0.00	–	0.01	**66**	0.01	–	0.00	–	0.00
binkar10_1	11244.2	0.01	11596.1	0.02	11505.7	0.01	11244.2	0.02	**1e+04**	0.02
blend2	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
bley_xl1	515.0	0.03	575.0	0.05	**465.0**	0.03	480.0	0.05	–	0.03
bnatt350	–	0.01	–	0.01	–	0.02	–	0.01	–	0.01
cap6000	-87646.0	0.06	-2e+05	0.11	**-2.3e+05**	0.09	-1.8e+05	0.05	-77813.0	0.06
core2536-691	–	197.52	–	194.44	**12280.0**	0.66	2e+04	39.20	–	293.35
cov1075	**56**	0.01	**56**	0.01	63	0.01	**56**	0.01	**56**	0.01
csched010	–	0.01	–	0.01	–	0.03	–	0.02	–	0.02
dano3mip	–	0.53	**807.6**	0.34	974.3	0.20	–	0.53	847.8	0.23
danoint	–	0.01	**66**	0.01	–	0.02	–	0.01	–	0.02
dcmulti	–	0.01	–	0.01	–	0.00	–	0.00	–	0.01
dfn-gwin-UUM	–	0.01	–	0.00	–	0.01	–	0.01	–	0.00
disctom	–	0.09	–	0.09	–	0.15	–	0.09	–	0.08
ds	**1308.2**	0.48	1308.2	0.50	2090.6	0.49	5418.6	0.49	5418.6	0.49
dsbmip	–	0.35	–	0.33	–	0.33	–	0.28	–	0.33
egout	667.1	0.00	663.5	0.00	**624.9**	0.00	667.1	0.00	626.6	0.00
eil33-2	**1321.7**	0.04	**1321.7**	0.04	2736.8	0.04	5050.2	0.03	5050.2	0.03
eilB101	**2427.3**	0.02	**2427.3**	0.02	3196.7	0.02	5e+03	0.02	5e+03	0.01
enigma	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
enlight13	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
fast0507	1.2e+05	0.47	1.2e+05	0.47	69159.0	0.48	8276.0	0.45	**8276.0**	0.43
fiber	–	0.00	–	0.01	–	0.01	–	0.00	–	0.01
fixnet6	–	0.01	–	0.01	–	0.01	–	0.01	–	0.02
flugpl	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
gen	**1.1e+05**	0.00	1.1e+05	0.01	1.2e+05	0.01	1.1e+05	0.01	–	0.01
gesa2-o	–	0.03	–	0.02	**1.3e+08**	0.02	1.3e+08	0.02	–	0.02
gesa2	**4.7e+07**	0.02	9.2e+07	0.02	–	0.02	1.5e+08	0.02	1.5e+08	0.02
gesa3	**5.9e+07**	0.02	–	0.02	–	0.03	1.5e+08	0.02	1.5e+08	0.02
gesa3_o	–	0.02	–	0.01	–	0.01	–	0.01	–	0.01
glass4	–	0.01	–	0.00	–	0.01	–	0.01	–	0.01
gmu-35-40	-5e+04	0.02	**-5e+04**	0.01	-5e+04	0.02	-5e+04	0.02	**-5e+04**	0.01
gt2	–	0.00	–	0.01	**3e+05**	0.00	–	0.00	–	0.00
harp2	-5.2e+07	0.01	–	0.01	–	0.00	**-5.2e+07**	0.00	–	0.01
iis-100-0-cov	46	0.02	**46**	0.01	64	0.01	88	0.02	88	0.02
iis-bupa-cov	98	0.03	**98**	0.02	224.0	0.02	315.0	0.02	315.0	0.02
iis-pima-cov	**92**	0.05	**92**	0.05	398.0	0.04	700.0	0.04	700.0	0.04
khb05250	1.6e+08	0.01	1.6e+08	0.01	**1.4e+08**	0.01	1.6e+08	0.01	1.6e+08	0.01
lectsched-4-obj	**277.0**	0.03	–	0.02	–	0.02	–	0.03	–	0.03
liu	**6450.0**	0.02	**6450.0**	0.02	6450.0	0.03	6450.0	0.03	6450.0	0.03
l152lav	–	0.02	–	0.01	–	0.02	–	0.03	–	0.04
lseu	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00

continued on next page

Table 3 continued

	$\lvert\{b<0\}\rvert\downarrow$		$\lvert\cdot\rvert\downarrow$		random		$\lvert\{b<0\}\rvert\uparrow$		$\lvert\cdot\rvert\uparrow$	
Problem Name	c^Tz	t (s)	c^Tz	t (s)	c^Tz	t (s)	c^Tz	t (s)	c^Tz	t (s)
m100n500k4r1	**0**	0.00	**0**	0.00	**0**	0.00	**0**	0.00	0	0.01
macrophage	1409.0	0.02	**1e+03**	0.02	1562.0	0.03	1582.0	0.02	1581.0	0.02
manna81	0	0.05	**0**	0.04	**0**	0.04	**0**	0.04	0	0.05
map18	0	0.19	0	0.19	0	0.19	0	0.19	**0**	0.18
map20	0	0.19	0	0.19	0	0.19	0	0.19	**0**	0.18
markshare1	**7286.0**	0.00	**7286.0**	0.00	**7286.0**	0.00	7286.0	0.01	**7286.0**	0.00
markshare2	**10512.0**	0.00	**10512.0**	0.00	**10512.0**	0.00	10512.0	0.01	**10512.0**	0.00
mas74	**1.5e+05**	0.00	1.5e+05	0.01	1.5e+05	0.01	1.5e+05	0.01	1.5e+05	0.01
mas76	1.5e+05	0.01	1.5e+05	0.01	**1.5e+05**	0.00	1.5e+05	0.01	1.5e+05	0.01
mcsched	4.8e+05	0.06	4.8e+05	0.07	**4.3e+05**	0.07	4.8e+05	0.10	4.8e+05	0.05
mik-250-1-100-1	0	0.02	**0**	0.01	**0**	0.01	**0**	0.01	**0**	0.01
mine-166-5	0	0.03	0	0.03	**0**	0.01	0	0.02	0	0.02
mine-90-10	0	0.03	**0**	0.01	**0**	0.01	0	0.03	**0**	0.01
misc03	–	0.00	–	0.00	–	0.01	–	0.00	–	0.00
misc06	**13951.9**	0.01	13951.9	0.02	13951.9	0.02	13951.9	0.03	**13951.9**	0.01
misc07	–	0.00	–	0.00	–	0.01	–	0.00	–	0.01
mitre	1.6e+05	0.03	1.3e+05	0.03	1.4e+05	0.04	1.5e+05	0.02	**1.2e+05**	0.02
mkc	0	0.21	0	0.19	0	0.12	0	0.21	**0**	0.04
mod008	1452.0	0.00	783.0	0.00	839.0	0.00	**498.0**	0.01	536.0	0.00
mod010	–	0.02	–	0.02	–	0.02	–	0.03	–	0.02
mod011	**0**	0.05	0	0.06	0	0.06	0	0.06	**0**	0.05
modglob	3.6e+07	0.01	**3.6e+07**	0.00	**3.6e+07**	0.00	**3.6e+07**	0.00	**3.6e+07**	0.00
momentum1	**3.6e+05**	0.11	–	0.13	–	0.06	–	0.08	4.8e+05	0.11
momentum2	–	0.13	–	0.16	–	0.20	–	0.24	–	0.12
momentum3	–	2.91	–	3.91	–	2.16	–	2.59	–	2.18
msc98-ip	–	0.06	–	0.06	–	0.11	–	0.05	–	0.09
mspp16	–	40.05	–	44.84	–	20.68	–	16.58	**385.0**	40.61
mzzv11	0	0.14	0	0.15	0	0.14	**0**	0.08	**0**	0.08
mzzv42z	0	0.19	0	0.38	0	0.27	0	0.16	**0**	0.14
n3div36	2.5e+06	0.69	3.6e+06	0.75	**1.8e+06**	0.80	2.9e+06	0.65	5.5e+06	0.74
n3seq24	9.7e+07	7.87	1.4e+08	8.45	**7.8e+07**	2.44	–	29.05	–	43.57
n4-3	–	0.06	3e+07	0.05	–	0.06	**3e+07**	0.06	–	0.04
neos-1109824	–	0.02	–	0.02	–	0.03	–	0.01	–	0.02
neos-1337307	–	0.03	–	0.13	–	0.31	–	0.06	–	0.06
neos-1396125	–	0.01	–	0.01	–	0.01	–	0.01	–	0.01
neos13	-28	0.96	**-28**	0.95	-28	0.96	**-28**	0.95	**-28**	0.95
neos-1601936	–	0.09	–	0.08	–	0.07	–	0.06	–	0.07
neos18	19	0.01	**18**	0.02	61	0.02	62	0.04	62	0.03
neos-476283	–	21.46	411.9	11.31	509.7	13.22	**411.9**	11.04	411.9	11.30
neos-686190	–	0.08	–	0.05	–	0.06	–	0.07	–	0.06
neos-849702	–	0.01	–	0.02	–	0.01	–	0.02	–	0.02
neos-916792	–	0.17	–	0.18	–	0.34	–	0.17	–	0.44
neos-934278	4.7e+05	0.41	**1.8e+05**	0.15	3.4e+05	0.11	3.6e+05	0.20	4.7e+05	0.48
net12	–	0.04	**296.0**	0.09	–	0.05	–	0.05	–	0.04
netdiversion	**4.9e+06**	20.10	–	6.83	–	16.27	–	48.24	–	9.32
newdano	–	0.00	–	0.01	**80**	0.01	–	0.01	–	0.00
noswot	**-5**	0.00	**-5**	0.00	**-5**	0.00	–	0.01	–	0.00
ns1208400	–	0.06	–	0.06	–	0.05	–	0.04	–	0.04
ns1688347	–	0.02	–	0.04	–	0.01	–	0.03	–	0.01
ns1758913	-236.8	2.09	-102.3	2.11	-226.7	2.84	-236.8	2.04	**-460.5**	6.03
ns1830653	–	0.02	–	0.03	–	0.03	–	0.03	–	0.03
nsrand-ipx	1.3e+06	0.05	2.7e+06	0.05	7.2e+05	0.05	8.5e+05	0.05	**1.6e+05**	0.06
nw04	29430.0	0.25	**29430.0**	0.24	57134.0	7.72	1.4e+05	19.05	1.4e+05	16.94
opm2-z7-s2	0	0.22	0	0.12	**0**	0.06	0	0.23	0	0.09
opt1217	0	0.01	**0**	0.00	**0**	0.00	**0**	0.00	0	0.01
p0033	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
p0201	12855.0	0.01	13210.0	0.00	**10815.0**	0.01	–	0.01	11260.0	0.01
p0282	9.1e+05	0.00	1.1e+06	0.00	6.5e+05	0.01	8.8e+05	0.01	**6.5e+05**	0.01
p0548	–	0.01	–	0.00	–	0.01	–	0.01	–	0.01
p2756	**6595.0**	0.06	1.3e+05	0.03	51211.0	0.06	1.5e+05	0.05	19419.0	0.08
pg5_34	0	0.02	0	0.02	**0**	0.01	0	0.02	0	0.02
pigeon-10	0	0.01	0	0.01	0	0.01	**0**	0.00	**0**	0.00
pk1	731.0	0.01	**731.0**	0.00	**731.0**	0.00	**731.0**	0.00	**731.0**	0.00
pp08a	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
pp08aCUTS	–	0.00	–	0.00	–	0.01	–	0.00	–	0.00
protfold	**-11**	0.04	–	0.03	–	0.03	–	0.03	–	0.03
pw-myciel4	22	0.01	22	0.01	**19**	0.01	–	0.02	–	0.01
qiu	–	0.01	–	0.02	–	0.01	–	0.02	–	0.02
qnet1	2.2e+05	0.01	2.3e+05	0.01	4e+05	0.02	**1.7e+05**	0.02	4.3e+05	0.02

continued on next page

Table 3 continued

Problem Name	$\lvert\{b<0\}\rvert \downarrow$		$\lvert\cdot\rvert \downarrow$		random		$\lvert\{b<0\}\rvert \uparrow$		$\lvert\cdot\rvert \uparrow$	
	$c^T z$	t (s)	$c^T z$	t (s)	$c^T z$	t (s)	$c^T z$	t (s)	$c^T z$	t (s)
qnet1_o	**78355.2**	0.01	2.6e+05	0.02	3.8e+05	0.02	4.1e+05	0.01	4.1e+05	0.02
rail507	–	3599.02	–	3599.54	–	2209.66	8276.0	1.86	**7674.0**	1.42
ran16x16	6e+03	0.01	6e+03	0.01	6712.0	0.01	**6e+03**	0.00	6e+03	0.01
reblock67	0	0.01	0	0.01	0	0.01	0	0.01	**0**	0.00
rd-rplusc-21	–	1.52	–	0.25	–	0.28	–	1.10	–	0.40
rentacar	–	0.39	–	0.56	–	0.45	–	0.51	–	0.44
rgn	**445.0**	0.00	445.0	0.01	**445.0**	0.00	**445.0**	0.00	**445.0**	0.00
rmatr100-p10	817.0	0.07	**763.0**	0.07	772.0	0.07	817.0	0.07	975.0	0.07
rmatr100-p5	1414.0	0.14	**1374.0**	0.13	1555.0	0.14	1414.0	0.14	2e+03	0.17
rmine6	0	0.05	**0**	0.01	0	0.02	0	0.06	**0**	0.01
rocII-4-11	–	0.03	–	0.03	–	0.03	–	0.02	–	0.02
rococoC10-001000	2.1e+05	0.03	–	0.01	**34348.0**	0.03	5e+04	0.03	54116.0	0.03
roll3000	–	0.04	–	0.06	–	0.02	–	0.02	–	0.04
rout	2375.2	0.01	2375.2	0.01	2375.2	0.01	2375.2	0.02	**2375.2**	0.00
satellites1-25	97	0.11	**79**	0.10	–	0.12	80	0.14	–	0.10
set1ch	**1e+05**	0.01	**1e+05**	0.01	1.1e+05	0.00	**1e+05**	0.01	**1e+05**	0.01
seymour	1269.0	0.02	1269.0	0.03	**1204.0**	0.03	1276.0	0.04	1276.0	0.03
sp97ar	2.6e+10	0.19	4.6e+10	0.19	2.3e+10	0.18	2.7e+10	0.20	**7.8e+09**	0.22
sp98ic	1.1e+10	0.20	1.6e+10	0.20	6.2e+09	0.22	8.2e+09	0.18	**5.2e+09**	0.21
sp98ir	5.2e+08	0.03	4.4e+08	0.03	–	0.03	–	0.05	**4.4e+08**	0.03
stein27	23	0.01	23	0.00	**21**	0.00	23	0.00	23	0.00
stein45	38	0.00	38	0.00	**37**	0.00	39	0.00	39	0.01
stp3d	–	0.90	–	1.17	–	1.93	–	0.89	–	12.95
swath	**713.2**	0.06	–	0.11	–	0.13	928.6	0.10	1e+03	0.08
t1717	–	0.13	–	0.12	–	0.09	–	0.09	–	0.10
tanglegram1	33625.0	0.25	33564.0	0.24	**32825.0**	0.30	33807.0	0.24	33807.0	0.24
tanglegram2	4172.0	0.06	**4158.0**	0.05	4241.0	0.05	4269.0	0.05	4267.0	0.05
timtab1	–	0.01	–	0.00	–	0.01	–	0.01	–	0.00
timtab2	–	0.00	–	0.00	–	0.01	–	0.01	–	0.00
tr12-30	–	0.02	–	0.02	–	0.03	–	0.02	–	0.02
triptim1	–	0.17	–	0.18	–	0.23	–	0.24	–	0.61
unitcal_7	–	0.12	–	0.11	–	0.12	–	0.10	–	1.04
vpm1	**23**	0.01	–	0.00	–	0.00	**23**	0.01	24	0.00
vpm2	–	0.00	–	0.00	–	0.01	–	0.00	–	0.00
vpphard	–	0.37	–	0.21	**2e+04**	5.39	–	7.41	–	4.71
zib54-UUE	–	0.02	–	0.02	–	0.04	–	0.02	–	0.02

Table 4: Results of the root experiment for all three settings in terms of heuristic/root solving time and objective value for all 163 instances. Bold face indicates better primal solutions values.

Problem Name		SandP		RandI		Both	
	$c^T z_{\text{opt}}$	$c^T z$	t (s)	$c^T z$	t (s)	$c^T z$	t (s)
10teams	924.0	–	0.00/0.18	–	0.00/0.32	–	0.02/0.33
30n20b8	302.0	–	0.02/9.60	–	0.01/9.59	–	0.04/9.55
a1c1s1	11503.4	–	0.02/0.27	–	0.02/0.28	–	0.03/0.29
acc-tight5	0	–	0.00/2.14	–	0.01/2.00	–	0.01/2.15
aflow30a	1158.0	**4606.0**	0.01/0.19	–	0.00/0.09	**4606.0**	0.03/0.21
aflow40b	1168.0	**8300.0**	0.03/0.81	–	0.00/0.78	**8300.0**	0.07/0.84
air03	3.4e+05	**6.2e+05**	0.04/5.08	–	0.02/5.05	**6.2e+05**	0.06/5.32
air04	56137.0	–	0.02/3.63	–	0.00/3.51	–	0.05/3.55
air05	26374.0	–	0.04/1.22	–	0.00/1.11	–	0.04/1.20
app1-2	-41	–	9.70/24.08	**-23**	0.07/13.82	–	9.86/24.23
arki001	7.6e+06	–	0.01/0.38	–	0.00/0.39	–	0.01/0.38
atlanta-ip	90	–	0.10/18.18	–	0.06/18.06	–	0.15/18.27
beasleyC3	754.0	–	0.00/0.07	**951.0**	0.05/0.19	**951.0**	0.05/0.17
bell3a	8.8e+05	–	0.00/0.01	**9.2e+05**	0.00/0.00	**9.2e+05**	0.01/0.02
bell5	9e+06	–	0.00/0.01	–	0.00/0.03	–	0.00/0.03
bab5	-1.1e+05	–	2.05/18.84	–	0.03/16.92	–	2.13/19.19
biella1	3.1e+06	9.5e+07	1.23/5.70	–	0.00/4.50	**9.5e+07**	1.86/6.30
bienst2	55	–	0.01/0.02	–	0.00/0.01	–	0.01/0.05
binkar10_1	6742.2	11244.2	0.02/0.13	–	0.00/0.12	**11244.2**	0.01/0.13
blend2	7.6	–	0.01/0.04	–	0.00/0.04	–	0.01/0.04
bley_xl1	190.0	515.0	0.05/351.94	–	0.00/310.20	**275.0**	0.05/311.14
bnatt350	0	–	0.01/0.75	–	0.00/0.74	–	0.02/0.75

continued on next page

Table 4 continued

		SandP		RandI		Both	
Problem Name	$c^T z_{\text{opt}}$	$c^T z$	t (s)	$c^T z$	t (s)	$c^T z$	t (s)
cap6000	-2.5e+06	-87646.0	0.06/0.73	-2.5e+06	0.16/0.83	**-2.5e+06**	0.20/0.98
core2536-691	689.0	–	1.56/13.46	**795.0**	0.09/12.15	**795.0**	1.65/13.68
cov1075	20	56	0.01/0.30	**26**	0.00/0.31	**26**	0.02/0.31
csched010	408.0	–	0.01/0.27	–	0.00/0.27	–	0.01/0.27
dano3mip	687.7	–	0.53/22.28	–	0.03/21.43	–	0.55/22.10
danoint	66	–	0.01/0.60	–	0.00/0.51	–	0.01/0.60
dcmulti	1.9e+05	–	0.01/0.03	–	0.01/0.04	–	0.01/0.03
dfn-gwin-UUM	38752.0	–	0.01/0.03	**2.3e+05**	0.00/0.03	**2.3e+05**	0.00/0.03
disctom	-5e+03	–	0.05/1.81	–	0.01/1.57	–	0.06/1.75
ds	94	**1308.2**	0.51/21.53	–	0.08/21.30	**1308.2**	1.10/22.18
dsbmip	-305.2	–	0.16/0.42	–	0.01/0.37	–	0.35/0.59
egout	568.1	667.1	0.00/0.01	**625.3**	0.00/0.01	**625.3**	0.00/0.00
eil33-2	934.0	**1321.7**	0.03/0.50	–	0.01/0.46	**1321.7**	0.09/0.55
eilB101	1216.9	**2427.3**	0.02/0.41	–	0.00/0.41	**2427.3**	0.04/0.43
enigma	0	–	0.00/0.01	–	0.00/0.01	–	0.00/0.01
enlight13	71	–	0.00/0.02	–	0.00/0.02	–	0.01/0.02
fast0507	174.0	1.2e+05	0.48/13.71	**240.0**	0.57/13.87	257.0	0.94/14.22
fiber	4.1e+05	–	0.01/0.04	–	0.00/0.04	–	0.02/0.04
fixnet6	4e+03	–	0.01/0.04	**4536.0**	0.02/0.05	**4536.0**	0.03/0.07
flugpl	1.2e+06	–	0.00/0.01	–	0.00/0.00	–	0.00/0.01
gen	1.1e+05	**1.1e+05**	0.01/0.06	–	0.00/0.05	**1.1e+05**	0.02/0.07
gesa2-o	2.6e+07	–	0.00/0.05	–	0.00/0.10	–	0.02/0.12
gesa2	2.6e+07	4.7e+07	0.02/0.16	1.9e+08	0.01/0.13	**4.6e+07**	0.05/0.18
gesa3	2.8e+07	5.9e+07	0.02/0.14	1.9e+08	0.00/0.12	**5.9e+07**	0.02/0.14
gesa3_o	2.8e+07	–	0.02/0.11	–	0.00/0.09	–	0.02/0.10
glass4	1.2e+09	–	0.01/0.04	–	0.00/0.04	–	0.01/0.04
gmu-35-40	-2.4e+06	-5e+04	0.01/0.22	**-1.5e+06**	0.00/0.21	**-1.5e+06**	0.01/0.20
gt2	21166.0	–	0.00/0.01	–	0.00/0.01	–	0.00/0.01
harp2	-7.4e+07	-5.2e+07	0.01/0.18	–	0.01/0.18	**-5.6e+07**	0.06/0.24
iis-100-0-cov	29	46	0.02/0.57	**35**	0.01/0.49	36	0.02/0.57
iis-bupa-cov	36	98	0.01/1.13	48	0.01/1.30	**47**	0.03/1.30
iis-pima-cov	33	92	0.05/1.26	44	0.03/1.50	**42**	0.05/1.39
khb05250	1.1e+08	1.6e+08	0.01/0.03	**1.3e+08**	0.00/0.01	**1.3e+08**	0.01/0.03
lectsched-4-obj	4	–	0.02/1.13	–	0.01/1.29	–	0.02/1.14
liu	1132.0	**6450.0**	0.04/0.10	–	0.00/0.05	**6450.0**	0.05/0.12
l152lav	4722.0	–	0.02/0.15	–	0.00/0.15	–	0.02/0.17
lseu	1120.0	–	0.00/0.01	–	0.00/0.01	–	0.00/0.01
m100n500k4r1	-25	0	0.01/0.03	**-18**	0.00/0.03	**-18**	0.02/0.04
macrophage	374.0	1409.0	0.03/0.13	609.0	0.02/0.13	**458.0**	0.03/0.15
manna81	-13164.0	0	0.04/0.26	**-13162.0**	0.07/0.19	**-13162.0**	0.11/0.31
map18	-847.0	0	0.19/7.88	**0**	0.17/7.11	**0**	0.30/8.00
map20	-922.0	0	0.19/7.37	**0**	0.18/6.51	**0**	0.32/7.62
markshare1	1	7286.0	0.00/0.01	**125.0**	0.00/0.01	**125.0**	0.00/0.01
markshare2	1	10512.0	0.00/0.00	**161.0**	0.00/0.00	**161.0**	0.00/0.01
mas74	11801.2	**1.5e+05**	0.01/0.02	–	0.00/0.01	**1.5e+05**	0.02/0.02
mas76	4e+04	**1.5e+05**	0.01/0.02	–	0.00/0.01	**1.5e+05**	0.01/0.01
mcsched	2.1e+05	**4.8e+05**	0.06/0.79	–	0.01/0.80	**4.8e+05**	0.08/0.81
mik-250-1-100-1	-66729.0	**0**	0.01/0.04	**0**	0.01/0.02	**0**	0.01/0.04
mine-166-5	-5.7e+08	0	0.03/1.66	**-7.3e+06**	0.02/1.79	**-7.3e+06**	0.03/1.75
mine-90-10	-7.8e+08	0	0.04/1.14	**-1.6e+07**	0.02/1.13	**-1.6e+07**	0.03/1.24
misc03	3360.0	–	0.00/0.04	–	0.00/0.04	–	0.00/0.05
misc06	12850.9	**13951.9**	0.02/0.13	–	0.00/0.13	**13951.9**	0.03/0.13
misc07	2810.0	–	0.00/0.09	–	0.00/0.11	–	0.01/0.11
mitre	1.2e+05	1.6e+05	0.03/4.94	–	0.00/5.03	**1.4e+05**	0.04/4.97
mkc	-563.8	0	0.21/0.49	**0**	0.11/0.40	**0**	0.25/0.53
mod008	307.0	1452.0	0.00/0.01	**308.0**	0.00/0.02	**308.0**	0.01/0.02
mod010	6548.0	–	0.02/0.12	–	0.01/0.20	–	0.02/0.21
mod011	-5.5e+07	0	0.05/0.50	**-4.3e+07**	0.04/0.48	**-4.3e+07**	0.07/0.53
modglob	2.1e+07	3.6e+07	0.00/0.02	**2.1e+07**	0.00/0.01	**2.1e+07**	0.00/0.02
momentum1	1.1e+05	**3.6e+05**	0.11/5.42	–	0.02/5.47	**3.6e+05**	0.12/5.47
momentum2	12314.2	–	0.13/11.70	–	0.01/11.43	–	0.13/11.07
momentum3	2.4e+05	–	1.82/387.71	–	0.05/375.27	–	1.76/379.42
msc98-ip	2e+07	–	0.07/6.41	–	0.01/6.35	–	0.08/6.24
mspp16	363.0	–	40.01/758.94	–	0.83/737.06	–	40.85/764.70
mzzv11	-21718.0	0	0.13/41.75	**0**	0.02/41.60	**0**	0.15/41.85
mzzv42z	-20540.0	0	0.19/9.44	**0**	0.10/9.39	**0**	0.24/9.42
n3div36	1.3e+05	2.5e+06	0.67/3.96	1.8e+05	0.38/3.70	**1.6e+05**	1.09/4.40
n3seq24	52200.0	9.7e+07	8.09/46.82	–	0.22/40.31	**1.6e+05**	8.96/48.01
n4-3	9e+03	–	0.06/0.34	**14275.0**	0.00/0.06	**14275.0**	0.06/0.19

continued on next page

Table 4 continued

		SandP		RandI		Both	
Problem Name	$c^T z_{\text{opt}}$	$c^T z$	t (s)	$c^T z$	t (s)	$c^T z$	t (s)
neos-1109824	378.0	–	0.02/0.86	–	0.00/0.86	–	0.01/0.98
neos-1337307	-2e+05	–	0.03/2.50	–	0.00/2.49	–	0.04/2.48
neos-1396125	3e+03	–	0.01/1.39	–	0.00/1.47	–	0.01/1.34
neos13	-95	**-28**	0.95/3.15	–	0.03/2.31	**-28**	0.99/3.17
neos-1601936	3	–	0.08/4.98	–	0.00/5.05	–	0.08/4.85
neos18	16	**19**	0.01/0.16	57	0.01/0.26	**19**	0.03/0.26
neos-476283	406.4	–	21.91/84.89	–	0.02/63.30	–	21.51/84.03
neos-686190	6730.0	–	0.09/0.33	–	0.01/0.26	–	0.08/0.33
neos-849702	0	–	0.01/1.10	–	0.01/1.16	–	0.02/1.15
neos-916792	32	–	0.17/0.87	–	0.00/0.80	–	0.17/0.92
neos-934278	260.0	**4.7e+05**	0.40/19.17	3.4e+10	0.04/18.93	**4.7e+05**	0.47/19.32
net12	214.0	–	0.04/3.94	–	0.02/3.96	–	0.05/4.05
netdiversion	242.0	**4.9e+06**	22.09/675.36	–	0.13/648.13	**4.9e+06**	28.70/674.29
newdano	66	–	0.01/0.03	–	0.00/0.02	–	0.00/0.04
noswot	-41	-5	0.00/0.01	–	0.00/0.01	**-6**	0.00/0.02
ns1208400	2	–	0.07/2.30	–	0.00/2.33	–	0.06/2.23
ns1688347	27	–	0.02/5.13	–	0.01/5.23	–	0.02/5.27
ns1758913	-1454.7	–	1.36/131.32	–	0.04/130.64	–	1.47/134.89
ns1830653	20622.0	–	0.01/0.66	–	0.00/0.58	–	0.02/0.82
nsrand-ipx	51200.0	1.3e+06	0.05/1.50	–	0.01/1.27	**70720.0**	0.09/1.42
nw04	16862.0	**29430.0**	0.26/11.70	–	0.08/11.61	**29430.0**	0.93/12.37
opm2-z7-s2	-1e+04	0	0.23/13.68	**-3685.0**	0.14/13.61	**-3685.0**	0.33/13.88
opt1217	-16	**0**	0.00/0.02	–	0.00/0.03	**0**	0.01/0.04
p0033	3089.0	–	0.00/0.01	–	0.00/0.00	–	0.00/0.01
p0201	7615.0	12855.0	0.00/0.06	–	0.00/0.06	**11295.0**	0.02/0.06
p0282	2.6e+05	9.1e+05	0.01/0.03	**2.7e+05**	0.00/0.02	**2.7e+05**	0.00/0.03
p0548	8691.0	–	0.01/0.04	–	0.00/0.04	–	0.02/0.05
p2756	3124.0	6595.0	0.05/0.34	–	0.01/0.29	**5e+03**	0.07/0.34
pg5_34	-14339.4	0	0.02/0.17	**0**	0.01/0.16	**0**	0.02/0.19
pigeon-10	-9e+03	**0**	0.01/0.05	**0**	0.01/0.04	**0**	0.01/0.05
pk1	11	**731.0**	0.00/0.01	–	0.00/0.01	**731.0**	0.00/0.01
pp08a	7350.0	–	0.01/0.01	**14600.0**	0.00/0.01	**14600.0**	0.00/0.01
pp08aCUTS	7350.0	–	0.00/0.02	**16630.4**	0.00/0.01	**16630.4**	0.02/0.02
protfold	-31	–	0.02/1.44	–	0.00/1.49	–	0.03/1.43
pw-myciel4	10	**22**	0.01/1.02	–	0.00/1.07	**22**	0.01/0.98
qiu	-132.9	–	0.00/0.10	**1805.2**	0.00/0.10	**1805.2**	0.01/0.11
qnet1	16029.7	2.2e+05	0.01/0.24	–	0.01/0.24	**26659.3**	0.04/0.26
qnet1_o	16029.7	78355.2	0.02/0.06	28462.1	0.01/0.05	**17842.7**	0.01/0.06
rail507	174.0	–	8.25/20.69	**216.0**	0.43/13.02	**216.0**	8.98/21.27
ran16x16	3823.0	6e+03	0.01/0.02	**4333.0**	0.01/0.02	**4333.0**	0.03/0.04
reblock67	-3.5e+07	0	0.02/0.91	**-2.6e+06**	0.01/0.84	**-2.6e+06**	0.03/0.85
rd-rplusc-21	1.7e+05	–	1.49/58.94	–	0.00/57.58	–	1.47/58.81
rentacar	3e+07	–	0.58/1.13	–	0.02/1.16	–	0.39/0.96
rgn	82	**445.0**	0.01/0.02	–	0.00/0.00	**445.0**	0.01/0.02
rmatr100-p10	423.0	**817.0**	0.07/2.52	–	0.01/2.59	**817.0**	0.08/2.50
rmatr100-p5	976.0	**1414.0**	0.14/4.17	–	0.01/4.16	**1414.0**	0.15/4.23
rmine6	-457.2	0	0.05/1.54	**-90**	0.01/1.35	**-90**	0.07/1.53
rocII-4-11	-6.7	–	0.02/6.18	–	0.01/6.17	–	0.04/6.15
rococoC10-001000	11460.0	2.1e+05	0.03/0.31	27042.0	0.03/0.33	**24044.0**	0.05/0.34
roll3000	12890.0	–	0.03/0.61	–	0.01/0.62	–	0.04/0.60
rout	1077.6	2375.2	0.01/0.09	**2375.2**	0.00/0.08	**2375.2**	0.01/0.09
satellites1-25	-5	**97**	0.10/39.36	–	0.00/37.12	**97**	0.12/39.79
set1ch	54537.8	**1e+05**	0.01/0.03	1.1e+05	0.02/0.04	**1e+05**	0.02/0.05
seymour	423.0	1269.0	0.03/1.19	**496.0**	0.03/1.04	500.0	0.06/1.18
sp97ar	6.6e+08	2.6e+10	0.18/5.67	–	0.05/5.57	**9.7e+08**	0.39/5.66
sp98ic	4.5e+08	1.1e+10	0.19/3.80	**5.4e+08**	0.22/3.94	8.3e+08	0.42/3.93
sp98ir	2.2e+08	5.2e+08	0.03/1.19	–	0.00/1.18	**3.1e+08**	0.04/1.38
stein27	18	23	0.00/0.01	**19**	0.00/0.00	**19**	0.01/0.01
stein45	30	38	0.01/0.02	33	0.00/0.01	**32**	0.00/0.02
stp3d	493.7	–	0.81/2215.49	–	0.21/2206.18	–	1.19/2239.59
swath	467.4	**713.2**	0.06/0.24	–	0.02/0.20	**713.2**	0.33/0.52
t1717	1.7e+05	–	0.08/6.82	–	0.06/6.86	–	0.14/6.77
tanglegram1	5182.0	33625.0	0.25/2.27	7798.0	0.72/2.76	**5406.0**	0.85/2.79
tanglegram2	443.0	4172.0	0.05/0.36	2122.0	0.15/0.38	**535.0**	0.16/0.49
timtab1	7.6e+05	–	0.00/0.03	–	0.00/0.02	–	0.00/0.02
timtab2	1.1e+06	–	0.00/0.06	–	0.00/0.05	–	0.01/0.05
tr12-30	1.3e+05	–	0.02/0.12	–	0.00/0.10	–	0.02/0.12
triptim1	23	–	0.17/111.54	–	0.03/112.41	–	0.21/132.15
unitcal_7	2e+07	–	0.10/17.19	–	0.03/17.86	–	0.14/17.25

continued on next page

Table 4 continued

Problem Name	$c^T z_{\mathsf{opt}}$	SandP $c^T z$	SandP t (s)	RandI $c^T z$	RandI t (s)	Both $c^T z$	Both t (s)
vpm1	20	**23**	0.00/0.00	24	0.01/0.02	**23**	0.01/0.01
vpm2	14	–	0.00/0.03	–	0.00/0.03	–	0.00/0.02
vpphard	5	–	0.27/11.66	–	0.05/11.54	–	0.34/11.67
zib54-UUE	1e+07	–	0.02/0.21	**1.8e+07**	0.00/0.21	**1.8e+07**	0.02/0.21