Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

GERALD GAMRATH, BENJAMIN HILLER, JAKOB WITZIG

Reoptimization Techniques in MIP Solvers

This paper is to appear in the Proceedings of the 14th International Symposium on Experimental Algorithms (SEA 2015) held June 29 – July 1, 2015, in Paris, France. The final publication is available at http://www.springerlink.com.

ZIB Report 15-24 (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

Reoptimization Techniques for MIP Solvers *

Gerald Gamrath Benjamin Hiller Jakob Witzig

Zuse Institute Berlin
Takustraße 7
D–14195 Berlin, Germany.
gamrath,hiller,witzig@zib.de

April 9, 2015

Abstract

Recently, there have been many successful applications of optimization algorithms that solve a sequence of quite similar mixed-integer programs (MIPs) as subproblems. Traditionally, each problem in the sequence is solved from scratch. In this paper we consider reoptimization techniques that try to benefit from information obtained by solving previous problems of the sequence. We focus on the case that subsequent MIPs differ only in the objective function or that the feasible region is reduced. We propose extensions of the very complex branch-and-bound algorithms employed by general MIP solvers based on the idea to "warmstart" using the final search frontier of the preceding solver run. We extend the academic MIP solver SCIP by these techniques to obtain a reoptimizing branch-and-bound solver and report computational results which show the effectiveness of the approach.

1 Introduction

In the last decades many powerful decomposition- and reformulation-based techniques for solving hard optimization problems were developed, e.g., column generation and Lagrangian relaxation. These methods decompose a problem into a master problem and several subproblems which are repeatedly solved to update the master problem. Frequently, the subproblems solved in successive iterations differ only in the cost vector, reflecting updated information from the master problem. It is a natural idea to exploit this property in order to improve the running time of the overall algorithm for solving the master problem. Methods to achieve this are known as *reoptimization techniques*. They have been investigated in the context of decomposition methods, e.g., in the context of Lagrangian relaxation [20], column generation [8], and for generic branch-and-bound [17].

*This paper is to appear in the Proceedings of the 14th International Symposium on Experimental Algorithms (SEA 2015) held June 29 – July 1, 2015, in Paris, France. The final publication is available at http://www.springerlink.com.

In the literature, reoptimization techniques have been investigated for polynomial solvable problems mainly, e.g., the shortest path problem [21] or the min cost flow problem [12]. This is partly due to the fact that traditionally, decomposition methods have been applied such that the resulting subproblems are (pseudo)polynomially solvable. More recently, *Mixed Integer Programs* (MIPs) have been used as subproblems, e.g., for cut generation [11, 10] or in generic decomposition schemes [22, 14] and corresponding solvers [15, 9]. The resulting subproblems are solved by standard MIP solvers, which are very sophisticated branch-and-bound algorithms. Thus there is a need for reoptimization techniques in MIP solvers to benefit from the knowledge obtained in previous iterations.

One of the first investigations on reoptimizing MIPs was done by Güzelsoy and Ralphs [23, 16]. They addressed sequences of MIPs that differ only in the right-hand side. Their approach is mainly based on duality theory, which they employed to develop techniques for "warm starting" using dual information obtained through primal algorithms. Our approach to reoptimizing MIPs is similar to the One-Tree algorithm [6] for generating multiple solutions of a single MIP. Similar techniques have also been used in [18] to benefit from a preliminary restricted branching phase when solving a single MIP instance.

In this paper, we propose a reoptimizing variant of the general LP-based branch-and-bound algorithm used by modern MIP solvers. It is based on the idea [17] to "continue" the search at the last known search frontier of the branch-and-bound tree. As the performance of state-of-the-art MIP solvers is based to a substantial extend on exploiting dual information, we introduce a mechanism to deal with this. This mechanism is in particular applied to cope with strong branching. It is intuitively clear that continuing the solving process is a poor idea if the objective function has changed a lot or the search frontier is rather huge. To deal with the first situation, we use a similarity measure for objective functions to decide whether to reoptimize or to start from scratch. Moreover, we propose heuristics to start with a reduced search frontier that is still based on the previous one. Our ideas have been carefully implemented using the MIP solver `SCIP` [25, 2]. We test our reoptimization techniques on sequences arising from the generic column generation solver `GCG` [15] and on instances of the k-constrained shortest path problem arising from a ship navigation problem[1]. More details and computational results can be found in the master thesis of the last author[27].

The paper is outlined as follows. Sec. 2 provides a summary of the relevant ingredients of a state-of-the-art MIP solver (i.e., `SCIP`) together with an in-depth motivation to reoptimization for MIPs. Sec. 3 presents our technical contributions summarized above. Computational results are discussed in Sec. 4. Finally, Sec. 5 concludes the paper.

[1]Thanks to Mirai Tanaka from Tokyo Institute of Technology and Kazuhiro Kobayashi from National Maritime Research Institute Japan for providing the instances.

2 Mixed Integer Programming and Reoptimization

In this paper we consider mixed integer linear programs (MIPs) of the form

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

with objective function $c \in \mathbb{R}^n$, constraint matrix $A \in \mathbb{R}^{m \times n}$ and constraint right-hand sides $b \in \mathbb{R}^m$, variable lower and upper bounds $\ell, u \in \bar{\mathbb{R}}^n$ where $\bar{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$, and a subset $\mathcal{I} \subseteq \mathcal{N} = \{1, \ldots, n\}$ of variables which need to be integral in every feasible solution. In the remainder of the paper, we focus on mixed-binary programs, i.e., MIPs with $\ell_i = 0, u_i = 1$ for all $i \in \mathcal{I}$.

When omitting the integrality restrictions, we obtain the *linear program (LP)*

$$z_{LP} = \min\{c^T x : Ax \geq b, \ell \leq x \leq u\}. \tag{2}$$

It is a relation of the corresponding MIP and provides a lower bound on its optimum, i.e., $z_{LP} \leq z_{MIP}$. This fact plays an important role in the most widely used method to solve general MIPs, the LP-based branch-and-bound method [19, 5]. It is a divide-and-conquer method which starts by solving the LP relaxation of the problem to compute a lower bound and a solution candidate $x^\star$. If $x^\star$ fulfills the integrality restrictions, the problem is solved to optimality, if not, it is split into (typically) two disjoint subproblems, thereby removing $x^\star$ from both feasible LP regions. Typically, a fractional variable $x_i, i \in \mathcal{I}$ with $x_i^\star \notin \mathbb{Z}$ is selected and the restrictions $x_i \geq \lceil x_i^\star \rceil$ and $x_i \leq \lfloor x_i^\star \rfloor$ are added to the two subproblems, respectively. This step is called branching. While this process is iterated, we store and update the best solution $\bar{x}$ found so far whenever one of the subproblems has an integral LP solution. The key observation is that a subproblem can be disregarded when its lower bound is not smaller than the objective value of $\bar{x}$. This is called bounding. This leads to the following cases that need to be distinguished when regarding a subproblem:

1. the LP relaxation is infeasible: the subproblem can be disregarded
2. $x^\star$ is integral: the subproblem is solved to optimality
3. $c^T x^\star \geq c^T \bar{x}$: the current subproblem can be disregarded due to bounding
4. else: the current subproblem is split (branching).

The branch-and-bound process is typically illustrated as a tree. The root node represents the original problem and the two subproblems created by the branching step correspond to two child nodes being created for the current node.

This general scheme is extended by various algorithms to enhance the performance (see [1]). We briefly sketch those advanced components that we need to handle specifically in the context of reoptimization. One of these components is *presolving*, which is done before the branch-and-bound process starts and analyzes the problem, removes redundancies and tightens the formulation. A reduced form of this, called *domain propagation*, is also done during the branch-and-bound phase before the LP relaxation of a node is solved. For more details, we refer to [24, 1]. Finally, the decision on which variable to branch is of high importance for a fast convergence. It is typically supported by a technique called

strong branching [3]. Strong branching precomputes lower bounds for potential child nodes of a candidate variable by solving auxiliary LPs with the branching bound change added. Besides providing very accurate lower bound predictions, it can also deduce bound changes or even infeasibility of the current node, if one or both of the regarded children for a candidate variable are infeasible or their lower bound exceeds the upper bound.

All these reductions can be divided into two classes: *primal* and *dual reductions*. The former are based only on feasibility arguments and remove only infeasible parts of the search space. In contrast to that, dual reductions are based on an optimality argument and may exclude feasible solutions as long as they retain at least one optimal solution. Therefore, dual reductions are not necessarily valid anymore if the objective is changed and we need to treat them with care in the reoptimization context.

2.0.1 Reoptimization for general MIP

In recent years, there is a growing interest in reoptimization techniques for MIP solvers. One of the major applications is the repeated solution of pricing problems within generic branch-cut-and-price solvers. Those solvers are based on a Dantzig-Wolfe decomposition [7] of the given original problem. The resulting problem is solved by a column generation approach, i.e., the LP relaxation is solved with just a subset of the variables and improving variables are searched for by solving the pricing problem, which is a MIP whose objective function depends on the current LP solution. For more details on generic branch-cut-and-price, we refer to [22, 14, 13]. Since the pricing problem is solved repeatedly with updated objective function, this would greatly benefit from an effective reoptimization technique and is thus our main application in the following.

Another application of reoptimization is the computation of the k best solutions of binary programs. This can be accomplished by iteratively solving the problem to optimality and excluding each optimal solution $\bar{x}$ for the next iteration by a *logic-or constraint*.

Definition 1 (logic-or constraint)**.** *Let $\mathcal{B} \subseteq \mathcal{I}$ be a set of binary variables and $C^-, C^+ \subseteq \mathcal{B}$ disjoint subsets. A* logic-or constraint *has the form*

$$\sum\nolimits_{i \in C^-} x_i + \sum\nolimits_{j \in C^+} (1 - x_j) \geq 1. \tag{3}$$

As a shortcut, we use the notation $x(C^-, C^+) := \sum_{i \in C^-} x_i + \sum_{j \in C^+} (1 - x_j)$. As can easily be seen, the logic-or constraint with $C^- = \{i \in \mathcal{I} \mid \bar{x}_i = 0\}$ and $C^+ = \{i \in \mathcal{I} \mid \bar{x}_i = 1\}$ forbids the optimal solution $\bar{x}$ and no other solution. Logic-or constraints play an important role in the remainder of this paper. Note that the sets C^- and C^+ do not need to be a partition of the variable set, but can also cover only a part of the variables in order to exclude all solutions with these variables set to the given values.

While there are more efficient ways to compute the k best solutions for a fixed k, the iterated approach proves useful if the limit k is decided during the optimization process. An example for this case is a mixed-integer nonlinear ship navigation problem investigated by Mirai Tanaka and Kazuhiro Kobayashi [26]. They repeatedly solve a MIP relaxation of the problem to compute a lower bound and a solution candidate. For this candidate, exact costs are computed

and it is excluded from the MIP relaxation. The procedure continues until the MIP lower bound exceeds the cost of the best solution found. Therefore, the number k of best solutions needed cannot be determined a priori and an iterated approach is preferred. In contrast to the reoptimization for branch-cut-and-price, the difference between two iterations is not in the objective function, but the additional constraint excluding the previously computed optimum.

In the following section, we will discuss how these two applications of reoptimization cases can be handled within a state-of-the-art MIP solver.

3 Extending SCIP to a Reoptimizing Branch-and-Bound Solver

For our approach we follow the ideas of [17]. Consider the sequence of MIPs

$$(P_i) \qquad \min\left\{c_i^T x \mid Ax \geq b, \ell \leq x \leq u, x_i \in \mathbb{Z}\right\} \quad \text{for all } i \in \mathcal{I} \tag{4}$$

for a given sequence of objective vectors $(c_i)_{i\in I}$, for some index set $I = \{1, \ldots, m\}$. Moreover, let $\mathcal{S}$ be the function mapping an optimization problem $\mathcal{P} \subseteq P_i$ to its set of feasible solutions X. Solving one problem of this sequence with a standard LP-based branch-and-bound algorithm up to a given stopping criterion, e.g., optimality, provides search space dividing subsets:

- a set of subproblems with an infeasible LP relaxation,
- a set $\mathcal{P}_{obj}$ of subproblems that have been either pruned due to bounding or are solved to optimality,
- a set Σ of all feasible solutions found so far, and
- the set $\mathcal{P}_o$ of as-yet unprocessed subproblems,

such that $X = \mathcal{S}(\mathcal{P}_o) \cup \mathcal{S}(\mathcal{P}_{obj}) \cup \Sigma$ holds, where $\mathcal{S}(\mathcal{P}_o)$ and $\mathcal{S}(\mathcal{P}_{obj})$ denote the set of solutions of $\mathcal{P}_o$ and $\mathcal{P}_{obj}$, respectively. Usually, the set $\mathcal{P}_o$ is empty at the end of the computation and there are open nodes only if the solving process terminates due to a stopping criterion different to optimality, e.g., time limit.

We call the solving process of (P_i) an iteration. To summarize the basic idea consider two iterations $i, i+1$. The authors of [17] proposed that solving P_{i+1} can be started at the last search frontier of iteration i. The set of leaf nodes generated in iteration i can be divided into the sets of unprocessed nodes $(\mathcal{P}_o)$ and nodes pruned because their dual bound exceeds the best known primal bound $(\mathcal{P}_{obj})$. Additionally, all feasible solutions found during the solving process of P_i are collected in Σ. Obviously, by changing the objective function c_i to c_{i+1} from iteration i to $i+1$, subproblems that are discarded since the LP relaxation is infeasible cannot become feasible. Thus, an optimal solution of P_i has to be in $\mathcal{S}(\mathcal{P}_o)$, $\mathcal{S}(\mathcal{P}_{obj})$ or Σ. Therefore, in the next iteration, we restore all subproblems in $\mathcal{P}_o \cap \mathcal{P}_{obj}$ as children of the root node. Additionally, all solutions from Σ are added to the solution pool. Note that in case additional constraints were added, we can apply this method as well but need to check these solutions for feasibility.

Additionally, we define a set $\mathcal{P}_f \subseteq \mathcal{P}_{obj}$ which contains all subproblems with an integral LP solution. We use this set in Sec. 3.2 where we present two heuristics which operate on the set of feasible nodes $\mathcal{P}_f$ only.

3.1 Handling Dual Information

Most state-of-the-art MIP solvers use, in addition to branch-and-bound, various preprocessing and domain propagation techniques. In connection with reoptimization, we have to be very careful when using these techniques. Dual methods provide bound changes based on the current objective function. Thus, the pruned part, i.e., the subproblem discarded by fixing variables, could contain feasible solutions which might be of interest after changing the objective function. Hence, it is necessary to disable all dual methods or treat them with special care. In the following we focus on binary and mixed binary programs of the form (1), where $\{l_i, u_i\} = \{0, 1\}$ for all $i \in \mathcal{I}$. Let us denote the set of binary variables by $\mathcal{B}$. Moreover, consider a subproblem $\mathcal{P}$ of P and a method D based on dual information. Calling D for $\mathcal{P}$ fixes variables $C = C^- \cup C^+ \subseteq \mathcal{B}$ to 0 and 1, respectively, where $C^- = \{i \in C \mid x_i = 0\}$ and $C^+ = \{i \in C \mid x_i = 1\}$. There are two ways to deal with these fixings. On the one hand, we can remember the subproblem $\mathcal{P}$ as before calling D and forget all decisions in the subtree induced by $\mathcal{P}$. The disadvantage of this approach is that some methods using dual information, e.g., strong branching, provide the most advantage at the root node. Thus, revoking these decisions will lead to the original problem $\mathcal{P}$. Hence, we do not benefit from reoptimizing. On the other hand, we can ensure that the pruned part may be reconstructed in the next iteration. In this paper we will follow the latter approach. The idea is to split problem $\mathcal{P} \subseteq P$ into two nodes at the beginning of the next iteration. The first node corresponds to $\mathcal{P}$ with all fixings $C^- \cup C^+$; the second node corresponds to $\mathcal{P}$ with an additional logic-or constraint $\mathfrak{C}$ depending on C^- and C^+. This logic-or constraint ensures that at least one variable get value different from the fixings $C^- \cup C^+$.

Theorem 2. *Let P be a binary or mixed binary problem, $\mathcal{P} \subseteq P$ a subproblem, and $C^-, C^+ \subseteq \mathcal{B}$ disjoint sets of binary variables. A complete and disjoint representation of the solution space $\mathcal{S}(\mathcal{P})$ of $\mathcal{P}$ is given by $\mathcal{S}(\mathcal{P}_\mathfrak{F}) \cup \mathcal{S}(\mathcal{P}_\mathfrak{C})$, where*

$$\mathcal{S}(\mathcal{P}_\mathfrak{F}) = \mathcal{S}(\mathcal{P}) \cap \{x \in \mathbb{R}^n \mid x_i = 0 \; \forall i \in C^- \text{ and } x_j = 1 \; \forall j \in C^+\} \tag{5}$$

$$\mathcal{S}(\mathcal{P}_\mathfrak{C}) = \mathcal{S}(\mathcal{P}) \cap \{x \in \mathbb{R}^n \mid x(C^-, C^+) \geq 1\}, \tag{6}$$

i.e., $\mathcal{S}(\mathcal{P}_\mathfrak{C})$ contains all solutions with at least one variable $x_i = 1$ or $x_j = 0$ for $i \in C^-$ and $j \in C^+$.

Proof. We only have to focus on variables x_i with $i \in C^+ \cup C^-$. Assume $\mathcal{S}(\mathcal{P}_\mathfrak{F})$ and $\mathcal{S}(\mathcal{P}_\mathfrak{C})$ are not disjoint. Let $x \in \mathcal{S}(\mathcal{P}_\mathfrak{F}) \cap \mathcal{S}(\mathcal{P}_\mathfrak{C})$. By definition of $\mathcal{S}(\mathcal{P}_\mathfrak{F})$ the solution x has to fulfill $x_i = 0$ and $x_j = 1$, for all $i \in C^-$ and $j \in C^+$. Thus, the constraint $x(C^-, C^+) \geq 1$ is violated by x. Hence, $x \notin \mathcal{S}(\mathcal{P}_\mathfrak{C})$ and $\mathcal{S}(\mathcal{P}_\mathfrak{F}) \cap \mathcal{S}(\mathcal{P}_\mathfrak{C}) = \emptyset$. Finally, consider $x \in \mathcal{P}$ and assume $x \notin \mathcal{S}(\mathcal{P}_\mathfrak{F}) \cup \mathcal{S}(\mathcal{P}_\mathfrak{C})$. Since $x \notin \mathcal{S}(\mathcal{P}_\mathfrak{F})$ at least one variable $i \in C^-$ and $j \in C^+$ needs to be different from $x_i = 0$ and $x_j = 1$, respectively. Hence, by the definition of $\mathcal{S}(\mathcal{P}_\mathfrak{C})$ it follows $x \in \mathcal{S}(\mathcal{P}_\mathfrak{C})$, which is a contradiction. □ □

3.2 Heuristics

In the following we present a primal heuristic which is fitted to column generation and two heuristics for reducing the size of the search frontier that needs to be reoptimized. The latter heuristics – so-called compression heuristics –

are needed because we have to solve the whole stored search frontier to prove optimality. Hence, a small search frontier of "good quality" would be desirable.

3.2.1 A Primal Heuristic: Trivialnegation

Consider a binary or mixed binary optimization problem $\mathcal{P}$ with n variables and two objective vectors $c, \tilde{c} \in \mathbb{R}^n$. Furthermore, let $x^\star \in \mathcal{P}$ be an optimal solution with $c^T x^\star = \min_{x \in \mathcal{P}} c^T x$. The impact of variable $i \in \mathcal{B}$ w.r.t. c and $\tilde{c}$ is defined by

$$\Psi_i = \begin{cases} 1 & \text{if } \operatorname{sgn}(c_i) \cdot \operatorname{sgn}(\tilde{c}_i) \leq 0 \text{ and } c_i \neq 0 \vee \tilde{c}_i \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Based on the impact Ψ_i of a variable $i \in \mathcal{B}$ we construct a (not necessarily feasible) solution candidate $\tilde{x}^B$ for each subset $B \subseteq \mathcal{B}$ by setting $\tilde{x}_i^B = 1 - x_i^\star$ if $\Psi_i = 1$ and $\tilde{x}_i^B = x_i^\star$ otherwise.

3.2.2 Compressing the Search Tree

Consider a search tree with nodes V and the set of leaf nodes $V_{leaf} \subseteq V$ resulting from solving a (mixed) binary optimization problem consisting of n variables with a branch-and-bound algorithm. Each node $v \in V$ corresponds to a subproblem

$$(\mathcal{P}^v) \quad \min\{c^T x : A^v x \geq b^v, \ell^v \leq x \leq u^v, x_i \in \mathbb{Z} \text{ for all } i \in \mathcal{I}\}, \tag{7}$$

where the constraint set $A^v x \geq b^v$ contains all constraints $Ax \geq b$ plus other constraints added during the branch-and-bound procedure, e.g., logic-or constraints. For each subproblem the set of binary variables can be partitioned into a set of unfixed variables, i.e., $\ell_i^v < u_i^v$, and the sets

$$X_v^0 = \{i \in \mathcal{B} \mid u_i^v = 0\} \quad \text{and} \quad X_v^1 = \{i \in \mathcal{B} \mid \ell_i^v = 1\}.$$

The key idea of our compression heuristics is to find a set of subproblems $\mathcal{P}^r, r \in \mathcal{R}$ of the form (7), such that each leaf $v \in V_{leaf}$ is represented by a unique r (for short: $r \succ v$), i.e., $\mathcal{S}(\mathcal{P}^v) \subseteq \mathcal{S}(\mathcal{P}^r)$. Moreover, we classify the quality of a representative by the loss of information w.r.t. the represented leaves

$$loss(V_{leaf}, r) = \sum\nolimits_{v \in V_{leaf} \,:\, r \succ v} \left| \left(X_v^0 \cup X_v^1\right) \setminus \left(X_r^0 \cup X_r^1\right) \right| \tag{8}$$

and the quality of a set $\mathcal{R}$ by the sum of losses of information for its representatives $r \in \mathcal{R}$.

To keep the search frontier small we use two heuristics. The first heuristic is called *largest representative* and reduces the search frontier to two nodes. In this paper we describe the basic idea only and refer to [27] for more details. Consider a subset of leaf nodes $W \subseteq V_{leaf}$. Based on an arbitrary node $v \in W$ we construct a representative r iteratively. First, we set $X_r^0 = X_v^0$ and $X_r^1 = X_v^1$. Afterwards, we add each node $w \in W$ greedily to the set of represented nodes as long as r and w have at least one common fixed variable and we update

$$X_r^0 \leftarrow X_r^0 \cap X_w^0 \quad \text{and} \quad X_r^1 \leftarrow X_r^1 \cap X_w^1.$$

The compressed search frontier is then given by $\mathcal{P}^r_{\mathfrak{F}}$ and $\mathcal{P}^r_{\mathfrak{C}}$, where

$$\mathcal{S}(\mathcal{P}^r_{\mathfrak{F}}) = \mathcal{S}(\mathcal{P}) \cap \{x \in \mathbb{R}^n \mid x_i = 0 \,\forall i \in X^0_r \text{ and } x_i = 1 \,\forall i \in X^1_r\}$$
$$\mathcal{S}(\mathcal{P}^r_{\mathfrak{C}}) = \mathcal{S}(\mathcal{P}) \cap \{x \in \mathbb{R}^n \mid x(X^0_r, X^1_r) \geq 1\}.$$

Since we are interested in finding good solutions fast, we run the heuristic on the set of feasible nodes $\mathcal{P}_f$ only. Moreover, to ensure that the compressed search frontier is as good as possible we run the procedure for each node $v \in \mathcal{P}_f$ and choose the representation with minimal loss of information.

The second heuristic is called *weak compression* and we have to distinguish between trees where the only difference between nodes are the sets of fixed variables ($A^v = A$) and trees with additionally added constraints, e.g., logic-or constraints. Due to page limitation we give the basic ideas only and refer to [27] for a complete description. Consider a tree without added constraints, its search frontier V_{leaf} and a subset of leaf nodes $W \subseteq V_{leaf}$. For each node $v \in W$ let v be a representative for itself. This leads to $|W|$ disjoint representatives. To ensure that the representation is complete, i.e., no feasible solution gets lost, we need to construct a subproblem covering $\mathcal{S}(\mathcal{P}) \setminus \bigcup_{v \in W} \mathcal{S}(\mathcal{P}^v)$. Such a representative can be constructed as before using Theorem 2, i.e., by including a logic-or constraint for each $v \in W$. Therefore, a complete representation of the search frontier is given by $\mathcal{R} = \{\mathcal{P}^{v_1}, \ldots, \mathcal{P}^{v_k}, \mathcal{P}_{\mathfrak{C}_W}\}$, where $W = \{v_1, \ldots, v_k\}$ and

$$\mathcal{S}(\mathcal{P}_{\mathfrak{C}_W}) = \mathcal{S}(\mathcal{P}) \cap \bigcap\nolimits_{v \in W} \{x \in \mathbb{R}^n \mid x(X^0_v, X^1_v) \geq 1\}.$$

If the search tree consists of nodes containing additional constraints, this construction cannot be adapted while guaranteeing completeness and disjointness. Thus, we restrict ourselves to the case $|W| = 1$. Assume $W = \{v\} \subseteq V_{leaf}$ with at least one fixed variable and potentially added logic-or constraints to handle dual reductions in previous iterations (see Sec. 3.1). Each of these constraints corresponds to disjoint variable sets $C_l = C^-_l \cup C^+_l \subset \mathcal{B}$. For guaranteeing a complete and disjoint representation we have to reconstruct subproblems which were cut off by the added constraints, for an illustration see Figure 1. Therefore, a complete and disjoint representation is given by $\mathcal{R} = \{\mathcal{P}^v_{\mathfrak{C}}, \mathcal{P}^{v,1}_{\mathcal{F}}, \mathcal{P}^{v,2}_{\mathcal{F}}, \ldots, \mathcal{P}^{v,k}_{\mathcal{F}}, \mathcal{P}^{v,k}_{\mathfrak{C}}\}$ (drawn solid in Figure 1), where

$$\mathcal{S}(\mathcal{P}^v_{\mathcal{F}}) = \mathcal{S}(\mathcal{P}) \cap \{x \in \mathbb{R}^n \mid x_i = 0 \,\forall i \in X^0_v \text{ and } x_j = 1 \,\forall j \in X^1_v\},$$
$$\mathcal{S}(\mathcal{P}^v_{\mathfrak{C}}) = \mathcal{S}(\mathcal{P}) \cap \{x \in \mathbb{R}^n \mid x(X^0_v, X^1_v) \geq 1\},$$
$$\mathcal{S}(\mathcal{P}^{v,1}_{\mathcal{F}}) = \mathcal{S}(\mathcal{P}^v_{\mathcal{F}}) \cap \{x \in \mathbb{R}^n \mid x_i = 0 \,\forall i \in C^-_1 \text{ and } x_j = 1 \,\forall j \in C^+_1\},$$
$$\mathcal{S}(\mathcal{P}^{v,1}_{\mathfrak{C}}) = \mathcal{S}(\mathcal{P}^v_{\mathcal{F}}) \cap \{x \in \mathbb{R}^n \mid x(C^-_1, C^+_1) \geq 1\},$$

and for all $l = 2, \ldots, k$

$$\mathcal{S}(\mathcal{P}^{v,l}_{\mathcal{F}}) = \mathcal{S}(\mathcal{P}^{v,l-1}_{\mathfrak{C}}) \cap \{x \in \mathbb{R}^n \mid x_i = 0 \,\forall i \in C^-_l \text{ and } x_j = 1 \,\forall j \in C^+_l\},$$
$$\mathcal{S}(\mathcal{P}^{v,l}_{\mathfrak{C}}) = \mathcal{S}(\mathcal{P}^{v,l-1}_{\mathfrak{C}}) \cap \{x \in \mathbb{R}^n \mid x(C^-_l, C^+_l) \geq 1\}.$$

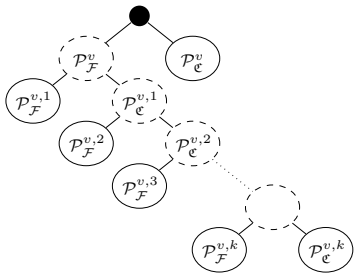

Figure 1: Construction of the weak compression heuristic.

3.3 Similarity of Objective Functions

If two objective functions are quite similar – whatever similar means at this point – we expect the resulting search trees to be similar as well. On the other hand, if two objective functions are quite different the resulting search trees might be very different, too. An immediate consequence is that continuing the solving process at the last search frontier might need much more effort than solving the problem from scratch. The following criterion can be used to estimate the similarity of the resulting search trees a priori.

Definition 3 (Similarity of Objective Functions)**.** *Let $c, \tilde{c} \in \mathbb{R}^n$ be two objective functions. Then the similarity of c and $\tilde{c}$ is given by*

$$\Lambda(c, \tilde{c}) = \frac{\langle c, \tilde{c} \rangle}{\|c\|_2 \|\tilde{c}\|_2}.$$

This corresponds to the cosine between the two objective vectors. We apply reoptimization only if the similarity measure is above a certain threshold (per default 0.8), since reoptimizing the search frontier in spite of very different objective functions seems to be not promising.

4 Computational Results

For evaluating our reoptimization approach we performed computational experiments on vertex coloring instances and on instances of the k-constrained shortest path problem. For the latter problem the sequence consists of k identical objective functions and the consecutive subproblems differ in exactly one logic-or constraint, cf. Sec. 2.0.1 and [26]. For generating sequences of objective functions for the vertex coloring problem we use a subset of the `COLOR02/03/04` [4] test set and the generic branch-cut-and-price solver `GCG` [15] which was introduced in [13]. The pricing subproblems that `GCG` needs to solve during the column generation process (see Sec. 2.0.1) only differ in the objective function. In order to test `SCIP` and `ReoptSCIP` on these problems, we write out the sequence of pricing problems of a `GCG` run, which avoids side-effects that a different optimal solution computed by any of the two solvers might have.

We performed all tests on an Intel Xeon CPU E3-1290 V2 @ 3.70GHz with 16GB RAM. Our implementation is based on `SCIP` 3.1.0.1 using `CPLEX` 12.6 as LP solver. We limited the solving time of each sequence by 3600 seconds and used the technique described in Sec. 3.1 in order to handle dual reductions

	variant			vertex coloring					k-constrained shortest path				
algorithm	TN	LR	WC	solved	faster	slower	nodes	time	solved	faster	slower	nodes	time
SCIP				47/47	0	37	3,896	47	66/66	0	60	1,143	55
	•			47/47	28	6	3,775	43	–	–	–	–	–
ReoptSCIP				47/47	33	4	6,590	38	66/66	48	11	1,310	38
	•			47/47	32	5	6,602	38	–	–	–	–	–
		•		47/47	36	1	4,909	33	66/66	57	1	1,112	35
			•	47/47	36	1	4,825	32	66/66	56	0	1,149	30
		•	•	–	–	–	–	–	66/66	57	3	1,125	36
	•	•	•	47/47	37	0	4,493	28	–	–	–	–	–

Table 1: Computational results on the vertex coloring (left) and k-constrained shortest path (right) test sets.

obtained by strong branching. The benefits obtained by dual presolving and propagation are less substantial and are outweighted by the enlargement of the search frontier they cause when applying the technique from Sec. 3.1. Therefore, we disabled all dual presolving and propagation techniques. We compute averages of solved nodes and solving time by using the shifted geometric mean [1, Appendix A] with shift $s = 10$ for running times and $s = 100$ for nodes.

In our computational experiments, we skip reoptimizing the search frontier if two consecutive objective functions are not similar enough, i.e., if the similarity is less than 0.8 (see Sec. 3.3). Furthermore, we solve the problem from scratch if the search frontier consists of more than 2000 nodes, because proving optimality needs more effort for a larger search frontier and thus the chance increases that solving from scratch succeeds faster. In order to shrink the search frontier, we run the largest representative (LR) heuristic exclusively on the set of feasible nodes $\mathcal{P}_f$ and we compress the search tree only if the determined representation is better than the last compression of a previous round, i.e., if the loss of information decreases or the number of fixings of the representative increases. We run the weak compression (WC) heuristic for one node only, since in almost every iteration the search tree includes nodes with added constraints. We choose the node with the largest dual bound in the previous iteration and demand that the number of fixed variables is at least one and not less than the number of fixed variables in the last weak compression. Additionally, we do not want to use the same node for weak compression twice in a row.

In Table 1 we compare the results of our reoptimization approach with plain `SCIP` on both test sets. To this end, we compare different variants, e.g., `SCIP` in combination with the trivialnegation (TN) heuristic or `ReoptSCIP` in combination with a single one or all three presented heuristics, as indicated in column "variant". Note that the trivialnegation heuristic is not applied to the k-constrained shortest path instances, since the objective function does not change. We compare the number of instances that are solved within the time limit and those solved significantly (at least 5%) faster or slower than plain `SCIP`. In a slight abuse of notation, we count an instance to be solved faster by plain `SCIP` if no other variant solved it at least 5% faster. Finally, we list the average number of branch-and-bound nodes needed to solve a sequence and the average solving time in seconds.

Our computational experiments show that `SCIP` does not solve any instance

faster than our new reoptimizing version `ReoptSCIP`. The best results on the vertex coloring instances can be achieved by using all three heuristics together. As opposed to that, the best results on the k-constrained shortest path instances can be achieved by using the weak compression heuristic only. Using the trivial-negation heuristic in combination with plain `SCIP` provides much more benefit than in combination with `ReoptSCIP`. This is caused by the number of nodes that need to be reoptimized to prove the optimality. Thus, the benefit obtained from constructing an optimal solution is consumed by proving its optimality, i.e., reoptimizing the search frontier. The compression heuristics can speed up the solving process of the vertex coloring instances by a factor of approximately 1.4. Thus, the constructed representatives are of good quality, i.e., they provide good dual and primal bounds. A pleasant observation is that the number of branch-and-bound nodes `ReoptSCIP` needs to solve the test sets is not much more than that of `SCIP`, which we did not expect. Summing up, we can state that using reoptimization improves the performance of `SCIP` significantly on our test sets.

5 Conclusions

We presented a reoptimization approach for MIP solvers which is based on a reconstruction of the branch-and-bound tree. We are able to handle dual reductions and introduced heuristics for compressing the tree. Applied carefully within `SCIP`, this method is able to reduce the solving time for two applications significantly. Therefore, it will be part of the next `SCIP` release.

Acknowledgements The work for this article has been supported by the *Research Campus Modal* funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM). The second author thanks the DFG for their support within project A04 in CRC TRR154. The authors would like to thank the anonymous reviewers for helpful comments on the paper.

References

[1] Achterberg, T.: Constraint Integer Programming. Ph.D. thesis, TU Berlin (2007)

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

[3] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: On the solution of traveling salesman problems. Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung Extra Volume ICM III, 645–656 (1998)

[4] COLOR02/03/04. `http://mat.gsia.cmu.edu/COLOR03/`

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

[6] Danna, E., Fenelon, M., Gu, Z., Wunderling, R.: Generating multiple solutions for mixed integer programming problems. In: Integer Programming and Combinatorial Optimization. LNCS, vol. 4513, pp. 280–294 (2007)

[7] Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Operations Research 8(1), 101–111 (1960)

[8] Desrochers, M., Soumis, F.: A reoptimization algorithm for the shortest path problem with time windows. European J. Oper. Res. 35(2), 242–254 (1988)

[9] DIP – Decomposition for Int. Programming. `https://projects.coin-or.org/Dip`

[10] Fischetti, M., Lodi, A.: Mipping closures: An instant survey. Graphs and Combinatorics 23, 233—-243 (2007)

[11] Fischetti, M., Lodi, A.: Optimizing over the first Chvátal closure. Math. Program., Ser. B 110, 3–20 (2007)

[12] Frangioni, A., Manca, A.: A computational study of cost reoptimization for min cost flow problems. INFORMS Journal on Computing 18(1) (2006)

[13] Gamrath, G.: Generic branch-cut-and-price. Master's thesis, TU Berlin (2010)

[14] Gamrath, G., Lübbecke, M.E.: Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In: Experimental algorithms, LNCS, vol. 6049, pp. 239–252. Springer (2010)

[15] GCG – Generic Column Generation. `http://www.or.rwth-aachen.de/gcg/`

[16] Güzelsoy, M.: Dual methods in mixed integer linear programming. Ph.D. thesis, Lehigh University, Bethlehem, Pennsylvania, USA (2009)

[17] Hiller, B., Klug, T., Witzig, J.: Reoptimization in Branch-and-Bound Algorithms with an Application to Elevator Control. In: Experimental Algorithms. LNCS, vol. 7933, pp. 378–389. Springer, Rome, Italy (2013)

[18] Karzan, F.K., Nemhauser, G.L., Savelsbergh, M.W.P.: Information-based branching schemes for binary linear mixed integer problems. Mathematical Programming Computation 1(4), 249–293 (2009)

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

[20] Létocart, L., Nagih, A., Plateau, G.: Reoptimization in Lagrangian methods for the 0–1 quadratic knapsack problem. Comput. Oper. Res. 39(1), 12–18 (2012)

[21] Miller-Hooks, E., Yang, B.: Updating paths in time-varying networks given arc weight changes. Transportation Science 39(4), 451–464 (2005)

[22] Ralphs, T.K., Galati, M.V.: Decomposition in integer linear programming. In: Karlof, J.K. (ed.) Integer Programming: Theory and Practice. CRC Press (2006)

[23] Ralphs, T.K., Güzelsoy, M.: Duality and warm starting in integer programming. In: The Proceedings of the 2006 NSF Design, Service, and Manufacturing Grantees and Research Conference (2006)

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

[25] SCIP – Solving Constraint Integer Programs. `http://scip.zib.de/`

[26] Tanaka, M., Kobayashi, K.: MISOCP formulation and route generation algorithm for ship navigation problem. Tech. Rep. 2013-8, Tokyo Inst. of Technology (2013)

[27] Witzig, J.: Reoptimization Techniques for MIP Solvers. Master's thesis, TU Berlin (2014)

Appendix

Algorithm 1 illustrates the reoptimizing LP-based branch-and-bound solver introduced in Sec. 3. In Table 2 and 3 we present detailed computational results of the tested instances. Both tables contain the number of generated nodes and needed solving time for plain `SCIP` as well as the different variants. The solving times of instances that are solved at least 5% faster by one of our variants than by plain `SCIP` are colored blue. On the other hand, the solving time `SCIP` needs for an instance is colored blue if no reoptimizing version solves the instances at least 5% faster. The last two lines of each table show the shifted geometric means of solved nodes, solving times, and the number of instances which are solved faster or slower compared to plain `SCIP`.

Algorithm 1: A reoptimizing LP-based branch-and-bound solver which is able to handle decisions based on dual information.

Input : Objective function $c\colon \mathcal{S} \to \mathbb{R}$; upper bound U; sets $\mathcal{P}'_{obj}$, $\mathcal{P}'_o$, Σ', $C^{-'}$, and $C^{+'}$.
Output: Sets $\mathcal{P}_{obj}$, $\mathcal{P}_o$, C^-, and C^+; set of solutions Σ.

$\mathcal{P}_{obj} \leftarrow \emptyset$, $\mathcal{P}_f \leftarrow \emptyset$, $\mathcal{P}_o \leftarrow \emptyset$, $\Sigma \leftarrow \Sigma'$
// Initialization
$\mathcal{P}_o \leftarrow \mathcal{P}'_o$
for $\mathcal{P}^v \in \mathcal{P}'_{obj}$ **do**
  if $C^{-'}(v) \cup C^{+'}(v) \neq \emptyset$ **then**
    Generate subproblems $\mathcal{P}^v_{\mathfrak{C}}$, $\mathcal{P}^v_{\mathfrak{F}}$. // split node
    $\mathcal{P}_o \leftarrow \mathcal{P}_o \cup \{\mathcal{P}^v_{\mathfrak{C}}, \mathcal{P}^v_{\mathfrak{F}}\}$
  else
    $\mathcal{P}_o \leftarrow \mathcal{P}_o \cup \{\mathcal{P}^v\}$
for $x \in \Sigma$ **do**
  $U \leftarrow c(x)$ if $c(x) < U$.
// Standard branch-and-bound
while $\mathcal{P}_o \neq \emptyset$ **do**
  Choose $\mathcal{P}^v \in \mathcal{P}_o$, $\mathcal{P}_o \leftarrow \mathcal{P}_o \setminus \{\mathcal{P}^v\}$
  Solve the LP relaxation of subproblem $\mathcal{P}^v$.
  if *the LP relaxation is feasible* **then**
    Let $x^\star_{LP}$ be the optimal solution of the LP relaxation.
    if $c(x^\star_{LP}) < U$ **then**
      if $x^\star_{LP}$ *is integral* **then**
        $\Sigma \leftarrow \Sigma \cup \{x^\star_{LP}\}$, $U \leftarrow c(x^\star_{LP})$, $\mathcal{P}_{obj} \leftarrow \mathcal{P}_{obj} \cup \{v\}$
      else
        Split $\mathcal{P}^v$ into subproblems $\mathcal{P}^{v_1}, \ldots, \mathcal{P}^{v_l}$
        $\mathcal{P}_o \leftarrow \mathcal{P}_o \cup \{\mathcal{P}^{v_1}, \ldots, \mathcal{P}^{v_l}\}$
    else
      $\mathcal{P}_{obj} \leftarrow \mathcal{P}_{obj} \cup \{\mathcal{P}^v\}$
  else if *dual reductions performed at* v **then**
    $\mathcal{P}_{obj} \leftarrow \mathcal{P}_{obj} \cup \{\mathcal{P}^v\}$
    Save variable sets $C^-(v)$ and $C^+(v)$. // store dual bound tightenings
  else
    Discard $\mathcal{P}^v$. // LP relaxation infeasible

	SCIP				ReoptSCIP									
	default		TN		default		TN		LR		WC		TN+LR+WC	
instance	nodes	time [s]	nodes	time [s]	nodes	time [s]	nodes	time [s]	nodes	time [s]	nodes	time [s]	nodes	time [s]
1-FullIns_3	102	≤ 1	94	≤ 1	156	≤ 1	156	≤ 1	130	≤ 1	121	≤ 1	121	≤ 1
1-FullIns_4	803	22	775	17	1,886	8	1,756	9	1,762	8	1,636	9	1,387	6
1-FullIns_5	24,769	2046	21,201	1889	118,346	1393	186,747	1917	79,069	1191	167,173	1822	44,836	679
1-Insertions_4	1,089	12	1,047	11	2,182	6	2,182	6	1,234	4	1,181	4	1,228	4
1-Insertions_5	6,528	318	5,760	286	66,090	302	47,529	278	7,330	117	12,888	158	6,564	90
2-FullIns_3	153	≤ 1	151	≤ 1	324	≤ 1	324	≤ 1	199	≤ 1	213	≤ 1	181	≤ 1
2-FullIns_4	4,181	175	4,134	158	137,120	354	137,120	357	5,368	55	5,156	51	4,697	40
2-Insertions_3	201	≤ 1	201	≤ 1	225	≤ 1	225	≤ 1	225	≤ 1	258	≤ 1	253	≤ 1
2-Insertions_4	10,355	210	10,099	196	25,275	154	25,231	157	18,933	111	10,992	76	12,222	75
3-FullIns_3	165	2	165	≤ 1	288	≤ 1	288	≤ 1	248	≤ 1	260	≤ 1	245	≤ 1
3-FullIns_4	2,865	324	2,961	288	26,290	229	21,617	217	5,182	126	8,336	155	6,432	115
3-Insertions_3	441	4	441	3	696	2	696	2	707	2	842	2	677	2
3-Insertions_4	1,436	108	1,458	82	1,834	55	1,836	54	2,352	59	3,197	73	3,073	63
4-FullIns_3	276	5	280	3	423	2	423	2	423	2	423	2	423	2
4-Insertions_3	785	10	783	8	1,081	5	1,081	5	1,179	5	1,066	5	1,068	4
4-Insertions_4	839	33	839	10	839	18	839	18	839	19	839	19	839	16
5-FullIns_3	340	9	338	5	1,067	4	1,067	4	689	3	643	4	482	3
DSJC125.1	759,419	624	757,657	650	764,521	425	764,521	414	752,876	415	761,473	427	753,073	374
DSJC125.5	558,296	995	557,922	1091	522,576	714	522,576	712	527,878	671	555,781	671	554,871	517
DSJR500.1	410,327	737	412,169	776	480,119	552	479,923	514	461,593	553	457,870	540	457,692	423
anna	154	3	154	≤ 1	154	≤ 1	154	≤ 1	154	≤ 1	154	≤ 1	154	≤ 1
ash331GPIA	13,964	353	16,329	325	83,259	418	68,624	374	79,827	433	38,978	302	53,072	274
david	103	≤ 1	103	≤ 1	113	≤ 1	113	≤ 1	110	≤ 1	111	≤ 1	110	≤ 1
games120	155	3	155	≤ 1	155	≤ 1	155	≤ 1	155	≤ 1	155	≤ 1	155	≤ 1
huck	85	≤ 1	85	≤ 1	89	≤ 1	89	≤ 1	89	≤ 1	89	≤ 1	89	≤ 1
jean	91	≤ 1	91	≤ 1	91	≤ 1	91	≤ 1	91	≤ 1	91	≤ 1	91	≤ 1
le450_15b	694,976	3115	678,331	3047	698,760	1970	696,948	2023	700,005	2085	698,952	2189	699,526	1634
miles250	643	3	665	2	510	2	510	≤ 1	517	≤ 1	516	≤ 1	551	≤ 1
mug88_1	205,487	46	205,196	46	297,350	50	297,350	51	263,050	45	286,418	50	250,933	40
mug88_25	233,089	49	234,389	49	314,243	47	314,243	46	295,759	44	294,026	44	292,287	39
mug100_1	1,827,890	319	1,841,520	334	1,987,330	236	1,983,208	231	1,932,110	279	1,945,297	239	2,016,122	221
mug100_25	735,511	140	732,559	144	858,763	112	858,763	118	830,125	113	847,569	115	822,923	102
myciel3	33	≤ 1	33	≤ 1	33	≤ 1	33	≤ 1	33	≤ 1	33	≤ 1	33	≤ 1
myciel4	92	≤ 1	92	≤ 1	133	≤ 1	133	≤ 1	133	≤ 1	121	≤ 1	121	≤ 1
myciel5	510	4	404	3	478	≤ 1	473	≤ 1	601	≤ 1	540	≤ 1	390	≤ 1
myciel6	8,159	64	4,275	54	18,761	55	23,173	50	4,344	20	4,683	22	3,955	21
myciel7	139,701	1372	93,164	1323	118,441	812	112,597	818	43,645	570	44,157	487	81,555	541
queen5_5	40	≤ 1	40	≤ 1	40	≤ 1	40	≤ 1	40	≤ 1	40	≤ 1	40	≤ 1
queen6_6	196	≤ 1	196	≤ 1	258	≤ 1	258	≤ 1	223	≤ 1	246	≤ 1	223	≤ 1
queen7_7	802	4	798	3	1,292	2	1,727	2	928	2	880	2	806	2
queen8_12	206	8	206	4	374	3	374	2	374	3	374	3	374	2
queen8_8	1,352	7	1,352	5	10,918	8	10,918	9	2,844	4	2,353	4	1,903	3
queen9_9	13,824	32	13,608	30	38,858	40	38,858	39	37,623	36	17,311	24	13,897	19
queen10_10	72,610	138	72,586	133	103,903	92	103,903	94	88,315	115	60,539	74	59,455	66
queen11_11	80,966	171	80,964	167	136,964	136	136,962	138	137,176	147	100,915	119	93,087	99
queen12_12	66,341	150	67,132	146	94,055	119	94,055	116	94,365	125	80,929	114	84,537	96
queen13_13	28,109	155	28,109	142	32,355	65	32,355	64	27,402	63	29,423	66	27,402	51
γ_s	3,896.59	47	3,775.17	43	6,590.29	38	6,602.86	38	4,909.35	33	4,825.56	32	4,493.53	28
faster/slower (of 47)		0/37		30/5		32/5		33/4		35/2		36/1		37/0

Table 2: Detailed computational results on the Vertex Coloring test set.

		SCIP		ReoptSCIP							
				default		LR		WC		LR+WC	
instance	k	nodes	time [s]	nodes	time [s]	nodes	time [s]	nodes	time [s]	nodes	time [s]
005_010_015	10	46	$\leq$ 1	49	$\leq$ 1	49	$\leq$ 1	58	$\leq$ 1	58	$\leq$ 1
	20	246	$\leq$ 1	210	$\leq$ 1	210	$\leq$ 1	235	$\leq$ 1	235	$\leq$ 1
	30	582	$\leq$ 1	489	$\leq$ 1	489	$\leq$ 1	554	$\leq$ 1	536	$\leq$ 1
	40	1,138	3	863	$\leq$ 1	863	$\leq$ 1	1,025	2	981	2
	50	2,082	4	1,292	2	1,292	2	1,647	2	1,483	2
	100	10,240	12	5,822	8	6,297	7	9,417	10	8,621	8
005_020_030	10	65	$\leq$ 1	57	$\leq$ 1	57	$\leq$ 1	63	$\leq$ 1	63	$\leq$ 1
	20	309	2	266	$\leq$ 1	266	$\leq$ 1	273	2	250	$\leq$ 1
	30	697	4	648	2	648	2	594	2	599	2
	40	1,383	6	1,116	4	1,116	4	1,148	4	1,170	4
	50	2,293	9	1,660	5	1,660	5	1,694	6	1,869	6
	100	10,347	29	6,365	17	6,418	18	7,481	20	8,092	21
005_030_040	10	46	$\leq$ 1	45	$\leq$ 1	45	$\leq$ 1	51	$\leq$ 1	51	$\leq$ 1
	20	292	3	256	2	256	2	273	3	273	3
	30	802	6	660	5	660	5	654	4	654	5
	40	1,490	9	1,273	7	1,273	7	1,446	7	1,501	7
	50	2,504	13	2,033	10	2,033	10	2,548	10	2,654	10
	100	12,010	40	7,829	32	9,558	31	11,732	31	12,749	35
005_040_050	10	32	$\leq$ 1	28	$\leq$ 1	28	$\leq$ 1	30	$\leq$ 1	30	$\leq$ 1
	20	264	4	180	3	180	3	251	4	251	3
	30	746	7	454	6	454	5	556	6	581	6
	40	1,584	12	900	8	900	8	1,113	9	1,397	9
	50	2,574	16	1,486	11	1,739	12	2,162	13	2,399	14
	100	12,048	51	7,076	35	7,548	39	11,028	41	10,773	39
005_050_060	10	62	9	60	6	60	6	67	6	67	7
	20	311	20	274	9	274	9	345	11	345	10
	30	990	37	647	14	647	15	878	21	878	21
	40	1,869	57	1,171	21	1,171	22	1,668	31	1,668	29
	50	3,087	75	1,848	28	1,848	30	2,705	41	2,705	41
	100	13,803	223	7,464	92	8,687	100	12,200	134	12,148	139
005_150_200	10	82	21	97	20	97	21	102	21	102	22
	20	356	47	684	42	684	45	371	40	371	40
	30	822	78	1,956	74	1,956	78	824	64	854	67
	40	1,534	115	3,928	114	3,347	113	1,466	93	1,666	100
	50	2,497	151	6,503	157	4,152	143	2,348	131	2,631	129
	100	10,629	407	28,795	470	12,526	353	9,500	349	9,569	335
005_200_250	10	82	39	74	15	74	16	94	18	94	21
	20	364	82	321	21	321	23	387	27	387	28
	30	846	132	720	29	720	34	862	38	866	46
	40	1,558	189	1,387	41	1,387	48	1,601	54	1,605	63
	50	2,666	253	2,329	58	2,329	66	2,731	51	2,669	91
	100	11,589	666	9,654	191	9,611	233	12,295	273	11,657	309
010_050_070	10	31	6	49	5	49	5	61	5	61	7
	20	235	15	318	13	318	13	315	12	315	14
	30	690	27	1,135	25	1,135	26	779	22	779	24
	40	1,707	42	3,153	45	2,026	40	1,597	33	1,625	37
	50	3,381	59	6,824	73	3,315	56	2,638	47	2,908	53
	100	16,825	174	174,543	813	16,304	187	18,403	156	14,652	522
010_100_140	10	80	25	118	25	118	23	86	22	86	24
	20	348	56	1,221	63	781	54	388	34	388	54
	30	938	96	3,218	114	1,519	92	1,127	73	1,095	85
	40	1,814	160	5,928	176	2,716	139	2,068	117	1,991	127
	50	2,950	224	9,430	246	4,387	189	3,292	90	3,078	176
	100	14,640	701	69,998	1030	16,333	525	15,137	273	13,143	144
010_150_200	10	59	51	31	13	35	14	34	7	34	16
	20	333	139	179	40	201	44	221	21	221	42
	30	821	254	827	95	524	90	735	46	794	96
	40	1,482	377	2,171	184	1,601	158	1,897	71	1,927	150
	50	3,452	535	4,035	302	3,409	247	4,197	111	4,204	247
	100	20,733	1496	17,054	1129	23,263	740	19,913	340	20,756	699
010_200_250	10	78	112	75	34	75	37	84	19	84	40
	20	347	267	298	67	298	71	397	37	397	73
	30	810	445	868	118	878	123	946	62	946	130
	40	1,632	658	2,254	206	2,277	204	1,781	96	1,934	195
	50	3,110	898	5,834	359	5,613	332	3,081	134	3,916	276
	100	21,336	2512	107,068	3314	33,553	1278	157,714	1999	23,828	1060
γ_s		1,143.63	55	1,310.82	38	1,112.85	35	1,149.16	30	1,125.2	36
faster/slower (of 66)			0/59		48/11		57/1		56/0		57/3

Table 3: Detailed computational results on the k-Constrained Shortest Path test set.