TIMO BERTHOLD, GREGOR HENDEL, AND
THORSTEN KOCH

The Three Phases of MIP Solving

The work for this article has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM). The work was supported by the ICT COST Action TD1207 *Mathematical Optimization in the Decision Support Systems for Efficient and Robust Energy Networks*. The research was supported by a Google Faculty Research Award.

ZIB Report 16-78 (December 2016)

Zuse Institute Berlin
Takustrasse 7
D-14195 Berlin-Dahlem

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

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

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

The Three Phases of MIP Solving

Timo Berthold[*]
Gregor Hendel[†]
Thorsten Koch[‡]

December 30, 2016

Modern MIP solvers employ dozens of auxiliary algorithmic components to support the branch-and-bound search in finding and improving primal solutions and in strengthening the dual bound. Typically, all components are tuned to minimize the average running time to prove optimality. In this article, we take a different look at the run of a MIP solver. We argue that the solution process consists of three different phases, namely achieving *feasibility*, *improving* the incumbent solution, and *proving* optimality. We first show that the entire solving process can be improved by adapting the search strategy with respect to the phase-specific aims using different control tunings. Afterwards, we provide criteria to predict the transition between the individual phases and evaluate the performance impact of altering the algorithmic behavior of the MIP solver SCIP at the predicted phase transition points.

1 Introduction

Mixed-integer programming (MIP) formulations are a valuable modeling tool for many decision problems from industry and economic areas. One of the reasons is the availability of powerful, commercial and non-commercial MIP solving software such as CBC [12], FICO XPRESS [34], GUROBI [17], IBM ILOG CPLEX [15], and SCIP [30, 1]. All of them employ an LP-based branch-and-bound [24] that solves a series of LP-relaxations obtained by branching on the integer variables with fractional LP solution values. A typical situation for branch-and-bound is that an optimal solution is found long before the proof of optimality is given and that a first feasible solution is found long before the optimal solution.

The idea of this paper is to partition the solution process into three solving phases, which we name after the specific goal which should be achieved during this phase: First, the solver tries to find a feasible solution during the *feasibility phase*. During the subsequent *improvement phase*, a sequence of solutions with improving objective is generated until the incumbent solution is eventually optimal. During the remaining *proof phase*, the search aims at proving optimality. Two questions must be answered

[*]Fair Isaac Germany GmbH, Takustraße 7, 14195 Berlin, Germany, timoberthold@fico.com
[†]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, hendel@zib.de
[‡]Queen Mary University of London, School of Mathematical Sciences, koch@zib.de

to obtain an adaptive solver that dynamically tunes its behavior w.r.t. different solving phases: How should the solver detect the transition between the improvement and the proof phase and how should it react on the phase transitions?

We mainly focus on the first question and present heuristic criteria for deciding whether a given incumbent solution is already optimal. Such criteria cannot be expected to be exact because the decision problem of proving whether a given solution is optimal is still $\mathscr{N}\mathscr{P}$-complete in general. Concerning the second question, we build upon the results of [19] for chosing adequate parameter tunings for each phase. Our computational results evaluate the impact of altering the solver settings at the predicted phase transition points.

The paper is organized as follows: In Section 2, we give an overview of previous work on prediction for search algorithms. In Section 3, we briefly introduce mixed-integer programs and important components of modern MIP solving software. Furthermore, we give a rough classification of the different components regarding the primal and dual progress of the solution process. We formalize the solving phases described above in Section 4 and discuss computational aspects of the different phases. The main contribution of this paper are heuristic criteria for deciding when the solver should stop searching for better solutions and concentrate on proving optimality. We present two such heuristic criteria that take into account global information of the search progress in Section 5. Finally, we present two computational studies to evaluate the accuracy of the heuristic criteria and their benefits when used inside an adaptive solver in Section 6.

2 Related work

It has been suggested to use different node selection and branching rules as long as no feasible solution has been found during a MIP solve, see [27]. In the notation that we introduce in Section 4, this could be seen as a 2-phase approach, switching branching parameters at the first phase transition.

The exponential nature of the branch-and-bound procedure has motivated research on early estimates of final search attributes such as the search tree size, the total amount of time, or the optimal objective value of the given problem. Most previous contributions have been made in the area of tree size prediction, mostly for general search trees. An application inside branch-and-bound is particularly challenging because the changing primal bound results in occasional pruning of huge parts of the search tree. Knuth [21] suggested to average the individual predictions of repeated random probes down the search tree as an unbiased estimate of the search tree size. A generalization of Knuth's method by Chen [13] introduced the use of stratifiers in order to reduce the variance of the estimate. The concept of stratifiers is also referred to as *type system* [25, 26, 35]. Chen's stratified sampling traverses a partial search tree in breadth-first order. Both Knuth and Chen discuss an additional difficulty of branch-and-bound algorithms for tree size prediction: the absence of knowledge about the optimal objective value of the problem at hand.

Recently, Lelis et al. [25, 26] suggested several extensions to stratified search. The first, a *two-step stratified sampling*, constructs a set of stratified search trees in the manner of Chen's [13] and then simulates a depth-first search to only visit nodes from the previously sampled trees. To better cope with a decreasing objective when new incumbent solutions are found, the authors [25] store additional objective information in the form of histograms. This is inspired by the use of histograms for tree size predictions by Burns et al. [11] in the context of iterative deepening in general search trees. The second approach by Lelis et al. [26] is called *retentive stratified sampling*. It uses auxiliary data on solution paths gathered during previous probes to model more correct pruning behavior of the actual search algorithm. Retentive stratified sampling produces predictions of similar quality without the exhaustive memory requirements of the two-step stratified search [25].

The *weighted backtrack estimate* by Kilby et al. [20] is an online method for binary search trees explored by depth first search. For each probe down the tree until a leaf node at depth d is reached, the estimate for the size of the tree considering only this single probe is $2^{d+1}-1$, while the probability of reaching this particular leaf by randomly choosing whether to go left or right at every depth is 2^{-d}. Kilby et al. combine this into a weighted mean

$$\hat{n} = \sum_{d \in D} \frac{2^{-d}(2^{d+1}-1)}{\sum_p 2^{-d}}$$

over the multiset of reached branching depths D before the algorithm backtracks.

Cornuéjols et al. [14] present an online method for tree-size prediction that uses some features of the shape of the partially explored tree to model a shape function for the whole search tree.

Clearly, every estimate for the final search tree size can be used to extrapolate the remaining solving time until termination, and vice versa. Hence, previous prediction methods for the end of the solution process estimate the end of the third phase in the terminology of Section 4. Estimated solution times or search tree sizes can be used for ranking different algorithms for the same problem, e.g., by running several algorithms in parallel for a limited amount of time (a so-called *racing ramp-up* [31]) and afterwards continuing the search with the algorithm that yielded the smallest tree size estimate. All methods are based on partial information of a search tree of the problem and designed to be early available.

The transition heuristics that we propose in Section 5 are different in that they attempt to recognize the point in time when the incumbent solution is optimal, which can happen long before the end of the solution process. Another difference is that transition heuristics are used to adapt the solver behavior directly during the search.

3 Mixed-integer programming and solving components

A mixed-integer program (MIP) is an optimization problem that minimizes a linear objective function subject to linear constraints over real- and integer-valued variables. Let $n, m \in \mathbb{N}$, $l, u \in \mathbb{Q}^n \cup \{-\infty, \infty\}^n$ denote lower and upper bounds for the variables. Let $A \in \mathbb{Q}^{m \times n}$ be a rational matrix, $b \in \mathbb{Q}^m$, and $c \in \mathbb{Q}^n$. Let further $\mathscr{I} \dot{\cup} \mathscr{C}$ be a partition of $\{1, \ldots, n\}$. We call $\mathscr{C}$ and $\mathscr{I}$ the *continuous* and *integer* variables, respectively. A *mixed-integer program (MIP)* is a minimization problem of the form

$$c^{\text{opt}} := \inf\{c^{\mathsf{T}}x : \; x \in \mathbb{Q}^n, \, Ax \le b, \, l \le x \le u, \, x_j \in \mathbb{Z} \quad \forall j \in \mathscr{I}\}.$$

A vector $y \in \mathbb{Q}^n$ is called a *(feasible) solution* for a MIP P, if it satisfies all linear constraints, bound requirements, and integrality restrictions of P. A solution y^{opt} that satisfies $c^{\mathsf{T}}y^{\text{opt}} = c^{\text{opt}}$ is called *optimal*. A MIP with no integrality restrictions $\mathscr{I} = \emptyset$ is called a *linear program (LP)*. The *LP-relaxation* of a MIP P is defined by dropping the integrality restrictions. By solving its LP-relaxation to optimality, we obtain a lower bound on the optimal objective of P.

The idea of branch-and-bound [24] is simple, yet effective: an optimization problem is recursively split into smaller subproblems, thereby creating a *search tree* and implicitly enumerating all potential assignments of the integer variables. The task of *branching* is to successively divide the given problem instance into smaller subproblems until the individual subproblems are easy to solve. Each node of the search tree represents one of the subproblems. The unprocessed subproblems are referred to as the *node frontier* [22]. The intention of *bounding* is to avoid the complete enumeration. If a subproblem's *dual bound* is greater than or equal to the *primal bound* given by the best solution found so far (the *incumbent*), that subproblem can be pruned. For MIPs, dual bounds are calculated by solving the subproblem's LP relaxation.

In modern MIP solvers such as SCIP or XPRESS, the basic branch-and-bound method is enhanced by various auxiliary algorithms with the purpose of improving the primal or dual convergence of the branch-and-bound method. We call such algorithms *solving components*. Among the most important types of solving components are

1. *Branching rules*: A branching rule represents a scoring mechanism to rank different alternatives how to split the current (sub-)problem further to enforce the LP-relaxation in the created child nodes. For an overview on MIP branching rules, see [4].

2. *Node selection rules*: A node selection rule determines the choice of the next open node from the search tree. Classical node selection rules include depth-first, breadth-first, best-bound and best-estimate [6].

3. *Presolving*: Presolving transforms the given problem instance into an equivalent instance that is (hopefully) easier to solve. Presolving removes redundant constraints or variables and strengthens the LP relaxation by exploiting integrality information. For more details on presolving, see [16].

4. *Cutting plane separation* Cutting planes separate the current LP relaxation solution from the convex hull of the solutions of the MIP. For an overview of computationally useful cutting plane techniques, see [28].

5. *Primal heuristics* Primal heuristics are auxiliary algorithms aimed at providing feasible solutions early during search. They can be classified based on the techniques they apply into rounding, propagation, diving and large neighborhood search heuristics, see [3]. For a recent overview of primal heuristics in MIP and MINLP solvers, see [8].

The integration and execution of MIP solver components inside of a complete solver such as SCIP [30] influences the overall solver performance. Clearly, some of the components mainly affect the primal bound, while others mainly contribute to the dual bound development. As a consequence, special settings for individual solving phases should put different emphases on each of the components.

In order to categorize the components' influence on the primal and dual convergence individually, Hendel [19] conducted an experiment where the solving components of SCIP were deativated one at a time or were replaced by a simple default mechanism (e.g., branching on random variables) on 168 MIP instances from MIPLIB 3.0 [10], MIPLIB 2003 [5], and MIPLIB 2010 [23]. The result of this experiment is visualized in Figure 1 which shows the percentage degradation in the primal and dual bound development compared with default solver settings. Here, *primal* and *dual integrals* [7] are used to measure the component's influence on the primal and dual convergence as follows:

Whenever there is an incumbent solution y, we measure the relative distance between y and the optimal objective value c^{opt} in terms of the primal gap

$$\gamma := \begin{cases} 0, & \text{if } c^{\text{opt}} = c^{\mathsf{T}}y, \\ 100 * \frac{|c^{\mathsf{T}}y - c^{\text{opt}}|}{\max\{|c^{\mathsf{T}}y|, |c^{\text{opt}}|\}}, & \text{if } \text{sig}(c^{\text{opt}}) = \text{sig}(c^{\mathsf{T}}y), \\ 100, & \text{otherwise.} \end{cases}$$

A primal gap of zero means that the incumbent is an optimal solution, although this might not be proven so far because the dual bound for P is less than the optimal objective. Before the first incumbent is found, we define $\gamma := 100$. With this gap definition, the *primal integral* is the integral of the *primal gap function* $\gamma(t)$, i.e., the primal gap as a function of time. We define the *dual gap (function)* and the *dual integral* analogously by replacing the primal bound $c^{\mathsf{T}}y$ with the current dual bound. The distinction between the primal and the dual gap enables us to analyze solving components regarding their influence on the primal and dual solver progress separately.

Not surprisingly, primal heuristics mainly affect the primal bound. Its average degrades by a factor of almost two when primal heuristics are deactivated, while the dual integral becomes only 6.5% worse on average. Branching and cutting plane separation mainly contribute to the development of the dual bound. The impact of node selection and presolving is mixed, both bounds deteriorate significantly if presolving is deactivated or a simple depth first search strategy is used for node selection. Notably, Figure 1 implies that all components have a positive impact on both the dual and the primal integral.

4 MIP solving phases

The main idea addressed in this paper is a partition of the branch-and-bound solving process into a set of phases. Figure 2 illustrates a typical primal and dual gap function for a MIP solving process. We draw the negative of the dual gap function to make the convergence of the primal and dual bound more intuitive. The feasibility phase is finished at t_1^* with the first feasible solution. After three more solutions, an optimal

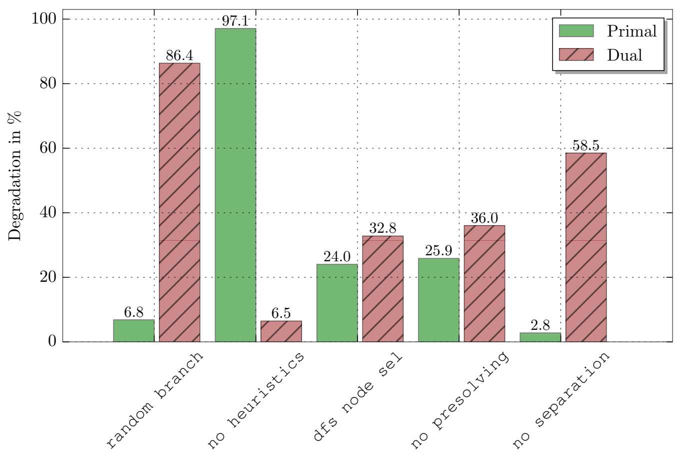

Figure 1: Percentage degradation compared to the SCIP performance with default settings regarding the shifted geometric mean primal and dual integral.

solution is found at t_2^*, such that the remainder of the solution process is dedicated to the proof of optimality. We formalize the tripartition of the solving process as follows:

Definition 1. Let P be a feasible MIP with optimal objective value $c^{\text{opt}} \in \mathbb{Q}$, and let γ, γ^* primal and dual gap functions. Denoting the point in time when the first solution is found by t_1^*, we define the three *solving phases* $\mathscr{P}_1$, $\mathscr{P}_2$, and $\mathscr{P}_3$ of S for P as the following time intervals:

$$\begin{aligned} \mathscr{P}_1 &:= [0, t_1^*[, \\ \mathscr{P}_2 &:= \{t \geq t_1^* : \ \gamma(t) > 0\}, \text{ and} \\ \mathscr{P}_3 &:= \{t \geq t_1^* : \ \gamma(t) = 0, \ \gamma^*(t) > 0\}. \end{aligned}$$

Clearly, the phases are disjoint. Furthermore, if T is the total time spent by a solver for solving P to optimality, the phases are a tripartition of the interval $[0, T]$. Each of the phases emphasizes a different goal of the solving process, so that it seems natural to pursue these goals with different parameter settings, which are tailored to achieve the phase objective as fast as possible.

The objective during the *Feasibility phase* $\mathscr{P}_1$ consists of finding a first feasible solution; the quality of this solution only plays a minor role. Feasible solutions are either provided by a node's LP-relaxation solution or by primal heuristics. The first feasible solution plays an important role for the solving process: First, it indicates the feasibility of the model to the user. Second, the bounding procedure of the branch-and-bound algorithm and some node presolving routines depend on a primal bound. Furthermore, several primal heuristics require a feasible solution as starting point to search for improvements.

After an initial feasible solution was found, the search for an optimal solution is conducted during the *Improvement phase* $\mathscr{P}_2$. During the *Improvement phase*, a sequence of IP-feasible solutions with decreasing objective value is produced until the

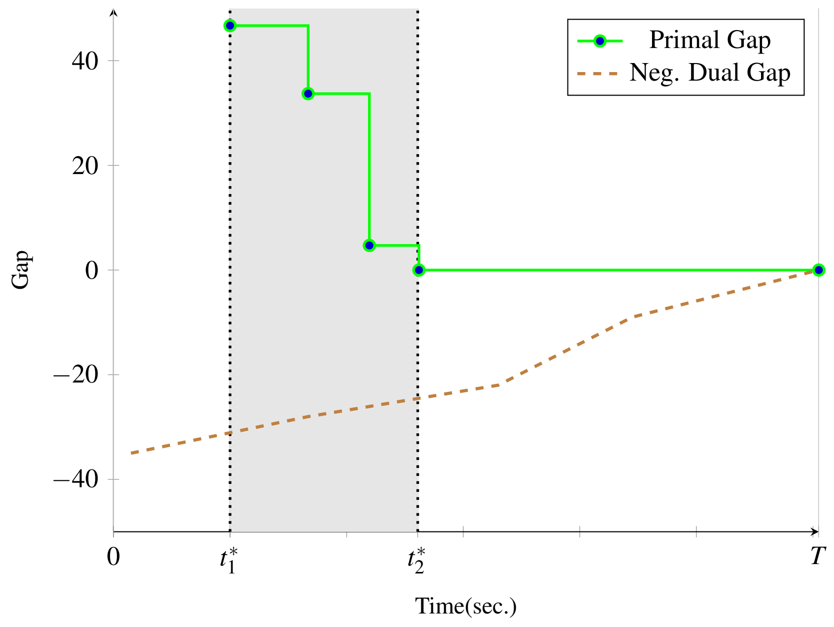

Figure 2: Solving phases of a MIP solution process with phase transition points t_1^* and t_2^*

solver eventually finds an optimal solution. For users of MIP solving software, this is often the most important phase. In many practical applications, MIP models are not required to be solved to proven optimality, reaching a small optimality gap is considered sufficient [7]. The reasons for this are threefold: there might be strict running time limitations, the models often are too large to complete the search and the input data itself might only be based on estimates.

The remaining time, which we call the *Proof phase* $\mathscr{P}_3$, is spent on proving the optimality of the incumbent solution. Such a proof requires the full exploration of the remaining search tree until there are no more open nodes with dual bound lower than the optimal primal objective value.

It is possible for both the *Improvement phase* and the *Proof phase* to be empty. If the first feasible solution is also an optimal one, then $\mathscr{P}_2 = \emptyset$; similarly, if the best possible dual bound is found before an incumbent with this optimal objective value is found, then $\mathscr{P}_3 = \emptyset$. In the special case that the MIP is a pure feasibility problem ($c = 0$), both the *Improvement phase* and *Proof phase* are empty. For the test set which we use for our computational experiments in Section 6, only one of 161 instances is a pure feasibility problem. Further, 11 instances have an empty *Improvement phase* and 34 instances have an empty or almost empty *Proof phase*, their union being 40 instances. Hence, it applies for a majority of 75 % of our test cases that the solving process is indeed partitioned into three nonempty phases by Definition 1.

We refer to the two points in time that mark the boundaries between the phases as *phase transitions*. More precisely, we call t_1^* the *first phase transition*, and we define the *second phase transition* as

$$t_2^* := \sup \mathscr{P}_1 \cup \mathscr{P}_2.$$

The recognition of the second phase transition t_2^* after the *Improvement phase* requires knowledge about the optimality of the current incumbent prior to the termination of the solving process. If $\mathscr{P}_3 \neq \emptyset$, the decision problem of proving that there exists no solution better than $\hat{y}(t_2^*)$ for our input MIP P remains to be solved. This problem is co-$\mathscr{N}\mathscr{P}$-complete in general.

In the next section, we address the problem of how to heuristically estimate the second phase transition during the solving process.

5 Two phase transition heuristics

If $\mathscr{P}_3 \neq \emptyset$, the detection of t_2^* requires knowledge of the optimal solution value. In this section, we present two heuristic criteria to estimate the second phase transition point t_2^* without knowledge of the optimal solution value. These *phase transition heuristics* will be used to switch to settings for the *Proof phase* when the criteria indicate that the current incumbent is optimal.

We present two phase transition heuristics based on a property called *node estimation* [6] of all nodes in the node frontier. The node estimation constitutes an estimate of the objective value of the best attainable solution of a node.

More formally, let $\mathscr{F}_P := \{j \in \mathscr{I} \,:\, (\tilde{y}_P)_j \notin \mathbb{Z}\}$ be the set of fractional variables of an LP solution $\tilde{y}_P$ at node P. For $j \in \mathscr{F}_P$, we define $f_P^-(j) := (\tilde{y}_P)_j - \lfloor (\tilde{y}_P)_j \rfloor$ and $f_P^+(j) := \lceil (\tilde{y}_P)_j \rceil - (\tilde{y}_P)_j$ the *down- and up-fractionality*, respectively, of j. For a branching direction $* \in \{+,-\}$, we denote by Ψ_j^* the average gain per unit fractionality over all prior branchings on j in direction $*$. We estimate the objective gain in branching direction $*$ by the *pseudo-costs* [6] $\Psi_j^* \cdot f_P^*(j)$.

Apart from their use in the selection of the best candidate for branching, pseudo-costs can also be applied to estimate the best solution attainable from a node P. Therefore, we denote by

$$\Psi_P^{\min}(j) := \min\{\Psi_j^- \cdot f_P^-(j), \Psi_j^+ \cdot f_P^+(j)\}$$

the *estimated minimum cost* to make variable $j \in \mathscr{I}$ integer.

Definition 2 (Node estimation [6])**.** The *node estimation* for a node P for which the LP-relaxation has been solved is given by the formula

$$\hat{c}_P = c^{\mathsf{T}} \tilde{y}_P + \sum_{j \in \mathscr{F}_P} \Psi_P^{\min}(j).$$

The rationale behind Definition 2 is to independently consider an estimate of making each fractional variable $j \in \mathscr{F}_P$ integer. In the following, we will be mainly interested in node estimations of open nodes, for which no LP relaxation has been solved so far. In order to determine an estimate of an *open node* Q, we simply subtract the contribution of the branching variable and direction that led to the creation of Q. Let Q be the child of another node P after branching upwards on $j \in \mathscr{F}_P$. An initial estimate of Q can be calculated via

$$\hat{c}_Q = \hat{c}_P - \Psi_P^{\min}(j) + \Psi_j^+ \cdot f_P^+(j),$$

thereby extending Definition 2 to open search nodes.

The node estimation does not account for a possible interplay between variables. This observation makes $\hat{c}_P$ likely to overestimate the actual integer objective value c_P^{opt} of the best attainable solution from the subtree rooted at P. It is, on the other hand, also possible to underestimate c_P^{opt}. Another important aspect concerns a possible degeneracy of the LP-relaxation: Whenever there exist different optima to the node LP-relaxation, they might lead to different estimates.

5.1 The best-estimate transition

We call the minimum node estimation among the set of open nodes $\mathscr{Q}$,

$$\hat{c}_{\mathscr{Q}}^{\min} = \min\{\hat{c}_Q : \; Q \in \mathscr{Q}\}$$

the *best-estimate*. The best-estimate is used as primary criterion for the default node selection in SCIP and is one possible estimate of the optimal objective value of a given MIP, for other possible estimates, we refer to [33]. As our first phase transition heuristic, we propose to switch to the *Proof phase* when the best-estimate exceeds the incumbent objective:

$$t_2^{\text{estim}} := \min\left\{ t \geq t_1^* \; : \; c^{\mathsf{T}} y(t) \leq \hat{c}_{\mathscr{Q}}^{\min}(t) \right\} \tag{1}$$

Note that by requiring $t \geq t_1^*$, we make sure that there is indeed an incumbent solution.

5.2 The rank-1 transition

With an increasing number of explored branch-and-bound nodes, it intuitively becomes less and less likely to encounter a solution better than the current incumbent. Yet, every unprocessed node $Q \in \mathscr{Q}$ has the potential to contain a better solution in the subtree underneath.

Definition 3. Let S be the search tree after termination, and define d_Q as depth and c_Q^{opt} as the (integer) optimal objective value for every node $Q \in S$ (or ∞ if there is no feasible solution for Q). We define the *rank* rg_Q of Q as

$$\text{rg}_Q := \left| \{ Q' \in S : d_{Q'} = d_Q, c_{Q'}^{\text{opt}} < c_Q^{\text{opt}} \} \right| + 1.$$

The rank rg_Q represents the minimum position of node Q in any list $\mathscr{P}^{d_Q}$ that contains all nodes at depth d_Q in nondecreasing order of their optimal solution. The root node P_0 trivially has a rank of 1, because it is the only node at depth 0. Indeed, if S were known in advance, the rank is defined in such a way that an optimal solution can be found by following a path of nodes of rank 1, starting at the root node. If the solving process has not uncovered an optimal solution yet, there exists a node of rank 1 among the open nodes $\mathscr{Q}$. Note, however, that there may even be nodes of rank 1 present in the node frontier although the current incumbent is already optimal.

The second phase transition heuristic is based on the definition (3) of node ranks. As for the best-estimate transition, we use the node estimation (cf. Definition 2) to circumvent the absence of true knowledge about best solutions in the unexplored subtrees. We impose a partial order relation $\prec$ on the nodes of the search tree S:

$$Q' \prec Q \quad \Leftrightarrow \quad Q' \text{ was processed before } Q \; d_{Q'} = d_Q, \quad \forall Q' \neq Q \in S.$$

With this partial order relation, we define the set of rank-1 nodes

$$\mathscr{Q}^{\text{rank-1}} := \{Q \in \mathscr{Q} \,:\; \hat{c}_Q \leq \inf\{\hat{c}_{Q'} : Q' \in S, Q' \prec Q\}\} \tag{2}$$

as the set of all open nodes with a node estimation at least as good as the best evaluated node at the same depth. Note that $\mathscr{Q}^{\text{rank-1}}$ may become empty much earlier than $\mathscr{Q}$, i.e., prior to the termination of the search, as soon as a single node with small node estimation has been processed at every depth of the current tree.

Using the following *rank-1 transition*, we assume that the current incumbent is optimal when $\mathscr{Q}^{\text{rank-1}}$ becomes empty:

$$t_2^{\text{rank-1}} := \min\{t \geq t_1^* \,:\, \mathscr{Q}^{\text{rank-1}}(t) = \emptyset\}. \tag{3}$$

If there is an open node Q at a depth d_Q which was not yet explored by the solving process, it holds that $Q \in \mathscr{Q}^{\text{rank-1}}$ since

$$\hat{c}_Q \leq \inf\{\hat{c}_{Q'} : Q' \in S, Q' \prec Q\} = \inf \emptyset = \infty.$$

The main difference between the best-estimate and the rank-1 transitions is that the rank-1 transition does not directly compare incumbent solution objectives and node estimations. On the one hand, it is an intuitive restriction to only compare nodes of the same depth because node estimations can be assumed to gain precision with increasing depth. Furthermore, it has a computational benefit because our update procedure only needs to compare newly inserted open nodes with other open nodes at the same depth.

For every depth d, we keep track of the minimum node estimate at this particular depth so far, including feasible nodes, i.e. subproblems with feasible LP-relaxation solutions. Every time a node is branched on, its two children are inserted in an array $\mathscr{Q}^d$ of open nodes at their depth d. $\mathscr{Q}^d$ is sorted in nondecreasing order of the node estimations of the nodes. In order to keep the set $\mathscr{Q}^{\text{rank-1}}$ updated, we store for every depth the best-estimate over all nodes already processed, which we update every time a node $Q \in \mathscr{Q}^{\text{rank-1}}$ was selected to be explored next.

6 Computational results

In this section, we first evaluate the potential of the two proposed transitions used with default settings. Second, we show how the use of the proposed transitions together with phase-specific solver settings affects the solution process.

6.1 Accuracy of the proposed phase transitions

In this section, we analyze the accuracy of the proposed phase transition heuristics from Section 5. We based our implementation on SCIP [1] 3.1.0 together with SoPlex [32] 2.0 as LP-solver. All computations were performed on a cluster of 32 computers. Each computer runs with a 64bit Intel Xeon X5672 CPUs at 3.20 GHz with 12 MB cache and 48 GB main memory. The operating system was Ubuntu 14.4. A gcc compiler was used in version 4.8.2. Hyperthreading and Turboboost were disabled. We ran only one job per computer in order to minimize the random noise in the measured running time that might be caused by cache-misses if multiple processes share common resources.

As test library, we use a combined library of MIPLIB 3 [10], MIPLIB 2003 [5], and MIPLIB 2010 [23] after the removal of three infeasible instances. In addition, we excluded the four instances for which, by the time of this writing, the optimal objective value was unknown, so that it is not possible to determine the actual phase transition t_2^*. On the remaining 161 instances we ran SCIP with default settings and a time limit of 2h. We record the solving time in seconds after which a transition criterion was reached for the first time. Before we start checking the transition criteria, we require the search to explore at least 50 branch-and-bound nodes for the node frontier to be meaningfully initialized.

It is noteworthy that the node estimations in SCIP are not updated dynamically together with the pseudo-costs due to running-time considerations, i.e., all nodes keep their initial estimation during the entire time they are in the node queue, although more recent pseudo-cost information on the variables might be available.

The goal of this experiment is to compare the proposed transition points with the actual second phase transition t_2^*. It may happen that $t_2^* > 2h$, i.e., an optimal solution is not found within the time limit, or a transition criterion is not met. Therefore, we first consider instances for which the solver finished the improvement phase within the time limit and at least one of the transition criteria was met.

We compare the relative difference between the two points in time by means of their shifted quotient $(t_2^{\text{crit}} + \tau)/(t_2^* + \tau)$ using a shift of $\tau = 10$ seconds. The use of a shift value compensates for very large or small quotients caused by numbers that are close to one and hence also shifts our attention to harder instances. A shifted quotient larger than one for an instance means that a phase transition heuristic correctly classifies an optimal incumbent solution. A quotient smaller than one, however, is encountered for instances where the transition criterion was met during the *Improvement phase*.

We present Figure 3 to compare the true second phase transition t_2^* and the phase transition t_2^{crit} that we recorded for the phase transitions. The histogram uses a bin width of 0.25 on the logarithm of the shifted quotients. The two bins around one therefore denote the time span shortly before or after t_2^*. We do not show instances which could be solved during the root node. Out of the remaining 147 instances, SCIP finds optimal solutions for 117. The rank-1 and the best-estimate criterion are reached for 91 and 93 of these instances, respectively. Note that the both rank-1 and the best-estimate transitions are trivially met whenever there is no open node left in the tree, i.e., after the search was completed.

We see in the figure that the bars for both transitions are centered around one, the rank-1 transition with 44 instances and the best-estimate transition with 41 instances.

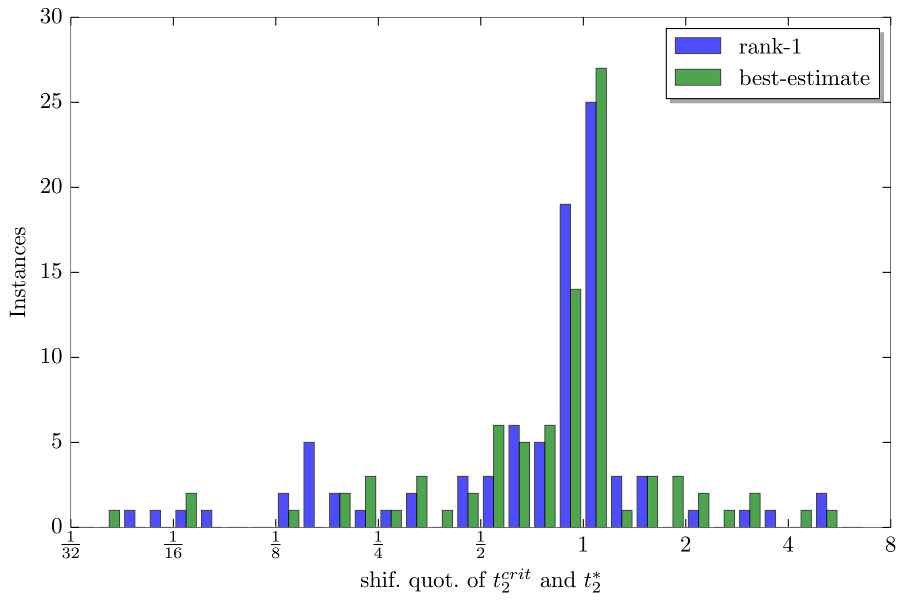

Figure 3: Relative deviation of the proposed transition criteria from the actual second phase transition t_2^* on instances where both values are smaller than 7200 seconds in our test.

Table 1: Contingency tables that group the 36 time limit instances into four categories whether the transition criterion was reached, and whether the incumbent at termination was optimal.

best-estimate transition

	opt.	not opt.	∑
reached	5	9	14
not reached	1	21	22
∑	6	30	36

rank-1 transition

	opt.	not opt.	∑
reached	4	15	19
not reached	2	15	17
∑	6	30	36

Both distributions slightly tend to take negative values, i.e., they tend to underestimate the second phase transition. Except for very few outliers, both distributions show a shifted quotient between $\frac{1}{8}$ and 8. In 75 out of 91 cases, the rank-1 transition approximates the actual phase transition by a factor of 5, the best-estimate transition in 81 out of 93 cases.

Next, we compare the primal-dual gap and the primal gap at the time of transition. Recall that the primal gap takes the optimal objective value into account so that it cannot be computed before the solving process finishes. We consider all instances for which the transition point in time was reached within the time limit. If a transition criterion is met during the proof phase, the primal gap is zero by definition. The average primal-dual gap at the rank-1 transition point is 19.26, while the primal-dual gap at the best-estimate transition point averages to 16.39. The average primal gap regarding the optimal objective value is 7.01 and 5.71 for the rank-1 and the best-estimate transition, respectively. This is in line with the previous observation that the best-estimate transition seems to occur later during the search than the rank-1 transition on average. It is interesting to note that the rank-1 transition occurs more often on instances where the primal gap is already small but is not yet proven by the primal-dual gap. More precisely, the transition point in time occurs on 46 out of 91 instances of our test bed when the primal gap is already smaller than 1 %, but the primal-dual gap is still larger than 1 %. The best-estimate transition achieves this on 38 out of 93 instances.

We now focus on instances where the solution process did not finish within our 2h time limit. For the instance stp-3d, no incumbent solution is found within two hours, so that neither transition criterion is met by definition. Among the remaining 36 instances that hit the time limit with an incumbent, there are 6 for which the incumbent is already optimal. We present in Table 1 two contingency tables grouped into two categories: whether the transition criterion was reached within the time limit or not and whether the incumbent solution at termination was already optimal. The entries on the diagonal represent instances for which the transition gives a correct classification, whereas the anti-diagonal gives the number of false positives and false negatives. Note that we measure only if a criterion was met at some point during the search, not if it was met precisely at termination. For the best-estimate transition, we see that for 5 out of 14 instances for which the transition criterion was reached, SCIP indeed finds an optimal solution. However, when the transition criterion is not reached within the time limit, this is in line with a suboptimal incumbent at termination in 21 out of 22 cases. Based

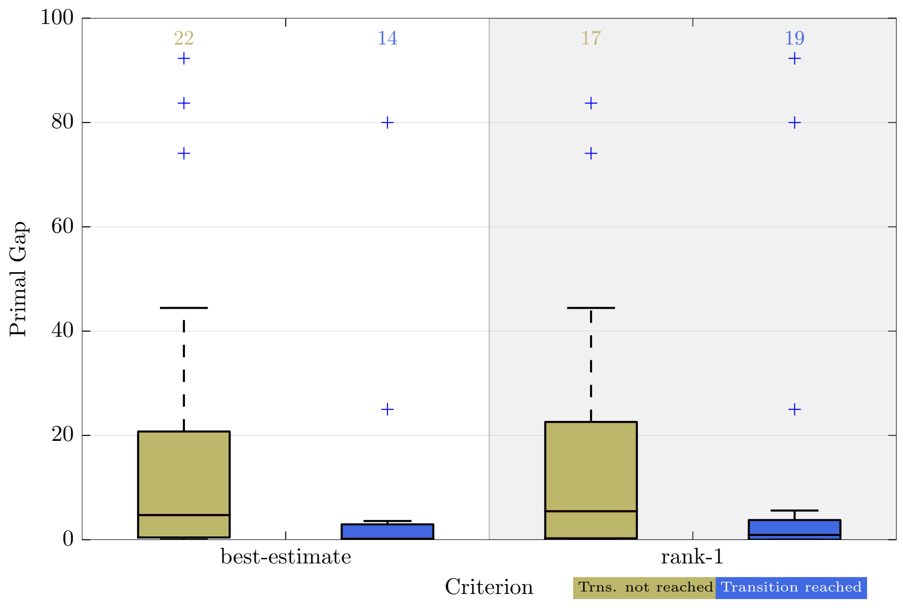

Figure 4: The primal gap at termination for 36 instances which could not be solved within the time limit.

on the best-estimate transition, we can classify 26 out of the 36 instances correctly. The table for the rank-1 transition does not show a similar result. Here, instances with suboptimal incumbent and with optimal incumbent are spread almost evenly across the two groups for this criterion.

As a second comparison, we quantify the incumbent quality at termination by the primal gap. Figure 4 shows the primal gap at termination for both transition criteria. We show two box plots per transition for the groups of instances for which the criterion was not reached (left box) and for which it was reached (right box) within the time limit. The number of instances in each group is shown at the top of the box plot. In contrast to the primal gap statistics presented before, we focus on the primal gap at termination, not on the primal gap at the time of the transition. For instances for which the respective transition criterion was not reached, the median primal gap is 4.7 for the best-estimate and 5.4 for the rank-1 transition. For both transition criteria, there is a clear tendency of the right group towards zero, for which the median values are at 0.17 for best-estimate and at 0.92 for rank-1.

6.2 Using phase transitions to control solver behavior

In an experimental study described in [18], promising settings for each of the three solving phases were determined individually. In this section, we combine those phase settings and the phase transition heuristics to a phase-based MIP solver.

For the *Feasibility phase*, we use a two-level node-selection as follows: We ap-

Table 2: Shifted geometric mean results for t (sec) for the 123 instances solved by all versions and on the subset of 52 hard instances.

		123 instances solved by all			52 hard ($\max t > 200$)		
	(# solv.)	t (sec)	t_{rel}	p	t (sec)	t_{rel}	p
`default`	(124)	90.7	1.000		799.0	1.000	nan
`estim`	(126)	85.0	0.938	0.529	695.2	0.870	0.338
`oracle`	(127)	84.5	0.932	0.004	665.9	0.833	0.000
`rank-1`	(125)	83.3	0.918	0.002	685.0	0.857	0.022

ply uct [29] for the first 31 nodes of the tree and afterwards a depth-first search that restarts periodically from the node with the best estimation. Furthermore, we altered the branching rule to branch exclusively on inference information [2]. This setting constituted the fastest setting in a *Feasibility phase* experiment that also found feasible solutions for all instances within the time limit of 1 hour. During the *Improvement phase*, we employ a setting that uses the default settings of SCIP except for an altered node-selection rule (uct) inside Large Neighborhood Search heuristics.

After a transition criterion for the second phase transition is reached, we continue the node selection of open nodes with a pure depth-first search, which is the fastest method to traverse the remaining tree if the cutoff bound is optimal (which we simply assume in this phase). We also disable all primal heuristics for the remainder of the search, because no further solutions are necessary during the *Proof phase*. Instead, we activate the separation of local cutting planes for the remainder of the search, as can be achieved with an aggressive emphasis on cutting planes in SCIP. With this setting, separation is performed at lower bound defining nodes at every 10'th level of the tree. For a detailed description of the methods involved and a comprehensive list of the used parameters, we refer to [18].

We compare four different versions of our phase-based solver: By `default`, we denote the default settings of SCIP used throughout all three phases. The version `estim` uses the phase transition heuristic best-estimate and `rank-1` the rank-1 transition. Both switch between the settings described above at the transition points t_1^* and t_2^{estim} or $t_2^{\text{rank-1}}$. The version `oracle` uses the exact phase transition point t_2^*, by receiving the optimal solution value as input. `oracle` is used to estimate the potential improvement of a phase-based solver and serves as a reference for the actual improvements of `estim` and `rank-1` which use heuristic predictions. For all runs, we set a time limit of two hours. The data for every instance of the test bed can be found in Table 4–6 in the Appendix.

The first column of Table 2 shows the number of instances that were solved to optimality. `default` could solve 124 instances within the time limit. All other versions solved between 1 and 3 instances more in total. The best version in this respect is `oracle`, which solved 127 instances. In the following, we focus on the subset of 123 instances that were solved by all versions. We restrict ourselves to the subset of solved instances to better compare results regarding both solving time and nodes.

For those 123 instances and a subset of 52 hard instances, the table shows the shifted geometric mean solving time, both absolute and relative compared to `default`.

Table 3: Shifted geometric mean results for the number of branch-and-bound nodes n. Results are restricted to 123 instances for which all versions could finish the solve within two hours, and the subset of 52 hard instances, respectively.

	all instances			hard ($\max t > 200$)		
	n	n_{rel}	p	n	n_{rel}	p
`default`	2565.5	1.000		17179.8	1.000	
`estim`	2454.6	0.957	0.209	15119.3	0.880	0.059
`oracle`	2377.3	0.927	0.000	13935.6	0.811	0.000
`rank-1`	2512.2	0.979	0.221	16577.4	0.965	0.347

An instance is considered hard if at least one solver needed more than 200 sec. We use a shift of $\tau = 10$ seconds. The table also shows p-values obtained from a modified two-sided Wilcoxon signed rank test that uses logarithmic shifted quotients and filtering to quantify if a version differed significantly from the default, see [9] for more details on this test methodology.

On the set of all instances, we observe improvements in the shifted geometric mean solving time for every version compared to `default`. The `oracle` version is 6.8 % faster than `default` in the shifted geometric mean. The improvement is even higher for the `rank-1` version, with which we achieve the highest speed-up over `default` of 8.2 %. These two improvements are accompanied by small p-values of 0.004 and 0.002, whereas the improvement shown for `estim` is not significant according to the Wilcoxon test.

On the hard instances, however, we observe significant improvements of up to 16.7 % with `oracle`, and still 14.3 % with `rank-1`. The version `estim` improves the shifted geometric mean time by 13.0 % but the corresponding p-value of 0.338 does not identify this improvement as significant. The reason for this lies in the fact that the `estim` version extremely improves the time on a few instances, namely acc-tight5 and lectsched-4-obj, for which `estim` achieves speed-up factors of 20 and 3, respectively, as well as six more instances with speed-ups of at least 2. However, disregarding these extreme speed-ups, the `estim` version shows more slow-downs compared to `oracle` and `rank-1`. The latter versions yield fewer extreme but more moderate speed-ups. The instance acc-tight5 is a pure feasibility instance in the sense that the dual bound is already provided by the initial LP relaxation and a feasible solution of this objective needs to be found. Thus, the performance on this instance is greatly affected by our modifications to the settings of SCIP during the *Feasibility phase*. The other phase-transition based settings yield the same speed-up for acc-tight5. Yet, an improvement of 14 % in the shifted geometric mean with `rank-1` is accompanied by a p-value of less than 3 %, which indicates that the rank-1 criterion is the better criterion w.r.t. the solving time.

Table 3 shows the shifted geometric mean results for all versions regarding the number of branch-and-bound nodes until optimality was proven. As in the previous table, we restrict the instances for the node comparison to the subset which could be solved to optimality by all settings within the time limit. The table further shows the results for the 52 hard instances of this set of instances. For the calculation of the mean, we use a shift of 100 nodes. The setting `oracle` significantly improves the overall shifted geometric mean of `default` by 7.3 % and by 18.9 % on the hard instances. For the criteria `estim` and `rank-1`, the obtained node reductions amount to 4.3 % and 2.1 %, respectively. In this case, however, the *p*-column does not indicate either of the improvements as significant. The split into easy and hard instances attributes the observed reductions mainly to the hard instances, where `estim` shows an improvement of 12 % compared to `default`.

Our experiments in the previous section revealed that `estim` transitions occur later during the search than `rank-1`-transitions which makes `estim` a more conservative transition criterion. While it achieves a good performance w.r.t. overall running time, `rank-1` performs even better, in particular when taking the statistic significance of the results into account. In the previous section, we saw that `rank-1` has a tendency to underestimate the point of phase transition. We interpret our results such that switching settings shortly before the second phase transition, i.e. when the solver is about to find the optimal solution, is sufficient, if not preferable, to improve performance. The fact that only one of the two transition heuristics achieves a significant time speed-up shows that the speed-up cannot be attributed only to the changes to the *Feasibility phase*, during which all three versions that employ phase-based settings have exactly the same behavior.

Using the rank-1 phase transition, we obtain a solving time improvement that is similar to the improvement obtained with `oracle`. Comparing the results for an oracle-based phase transition and the phase-transition criteria that we introduced, we conclude that the rank-1 transition is sufficient in practice to achieve a solving-time performance similar to what can be obtained in principle if we could determine the phase transition exactly.

7 Conclusion

In this article, we discussed the partition of a MIP solving process into three phases: *feasibility*, *improvement*, and *proof*. Each of them benefits from different algorithmic components. We introduced and empirically analyzed two criteria to predict the transition between the improvement and the proof phase: the best-estimate transition and the rank-1 transition.

We showed that a phase-based version of the MIP solver SCIP, using the rank-1 transition, improves SCIP's mean running time by 8 % on general MIP instances, and 14 % on hard instances, while simultaneously reducing the number of branch-and-bound nodes. Hence, our computational experiments provide evidence that those easy-to-evaluate criteria correlate sufficiently well with the actual, hard-to-detect phase transition to make use of this approach in practice.

Acknowledgement(s)

The work for this article has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number

05M14ZAM). The work was supported by the ICT COST Action TD1207 *Mathematical Optimization in the Decision Support Systems for Efficient and Robust Energy Networks*. The research was supported by a Google Faculty Research Award.

References

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

[2] T. Achterberg and T. Berthold, *Hybrid Branching*, in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, Lecture Notes in Computer Science, Vol. 5547, Springer, 2009, pp. 309–311.

[3] T. Achterberg, T. Berthold, and G. Hendel, *Rounding and Propagation Heuristics for Mixed Integer Programming*, in *Operations Research Proceedings 2011*, Springer Berlin Heidelberg, 2012, pp. 71–76.

[4] T. Achterberg, T. Koch, and A. Martin, *Branching rules revisited*, Operations Research Letters 33 (2004), pp. 42–54.

[5] T. Achterberg, T. Koch, and A. Martin, *MIPLIB 2003*, Operations Research Letters 34 (2006), pp. 1–12.

[6] M. Bénichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent, *Experiments in mixed-integer programming*, Mathematical Programming 1 (1971), pp. 76–94.

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

[8] T. Berthold, *Heuristic algorithms in global MINLP solvers*, Ph.D. diss., Technische Universität Berlin, 2014.

[9] T. Berthold, G. Gamrath, and D. Salvagnin, *Cloud branching*, Presentation slides from Mixed Integer Programming Workshop at Ohio State University. https://mip2014.engineering.osu.edu/sites/mip2014.engineering.osu.edu/files/uploads/Berthold_MIP2014_Cloud.pdf (2014).

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

[11] E. Burns and W. Ruml, *Iterative-deepening search with on-line tree size prediction*, Annals of Mathematics and Artificial Intelligence 69 (2013), pp. 183–205, Available at http://dx.doi.org/10.1007/s10472-013-9347-9.

[12] *COIN-OR branch-and-cut MIP solver* (2016), https://projects.coin-or.org/Cbc.

[13] P.C. Chen, *Heuristic sampling: A method for predicting the performance of tree searching programs*, SIAM Journal on Computing 21 (1992), pp. 295–315, Available at http://dx.doi.org/10.1137/0221022.

[14] G. Cornuéjols, M. Karamanov, and Y. Li, *Early estimates of the size of branch-and-bound trees*, INFORMS Journal on Computing 18 (2006), pp. 86–96, Available at http://dx.doi.org/10.1287/ijoc.1040.0107.

[15] *IBM ILOG CPLEX Optimizer* (2016), http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[16] G. Gamrath, T. Koch, A. Martin, M. Miltenberger, and D. Weninger, *Progress in presolving for mixed integer programming*, Mathematical Programming Computation 7 (2015), pp. 367–398.

[17] *GUROBI Optimizer* (2016), http://www.gurobi.com/products/gurobi-optimizer/gurobi-overview.

[18] G. Hendel, *Empirical analysis of solving phases in mixed integer programming*, Master thesis, Technische Universität Berlin (2014).

[19] G. Hendel, *Enhancing MIP Branching Decisions by Using the Sample Variance of Pseudo Costs*, in *Integration of AI and OR Techniques in Constraint Programming*, Vol. 9075, in press, 2015, pp. 199 – 214.

[20] P. Kilby, J.K. Slaney, S. Thiébaux, and T. Walsh, *Estimating Search Tree Size*, in *AAAI*, AAAI Press, 2006, pp. 1014–1019.

[21] D.E. Knuth, *Estimating the efficiency of backtrack programs.*, Tech. Rep., Stanford University, Stanford, CA, USA, 1974.

[22] T. Koch, A. Martin, and M.E. Pfetsch, *Progress in academic computational integer programming*, in *Facets of Combinatorial Optimization*, M. Jünger and G. Reinelt, eds., Springer, 2013, pp. 483–506.

[23] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter, *MIPLIB 2010*, Mathematical Programming Computation 3 (2011), pp. 103–163, Available at http://dx.doi.org/10.1007/s12532-011-0025-9.

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

[25] L.H.S. Lelis, L. Otten, and R. Dechter, *Predicting the Size of Depth-first Branch and Bound Search Trees*, in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, Beijing, China, Available at http://dl.acm.org/citation.cfm?id=2540128.2540215, AAAI Press, 2013, pp. 594–600.

[26] L.H.S. Lelis, L. Otten, and R. Dechter, *Memory-Efficient Tree Size Prediction for Depth-First Search in Graphical Models*, in *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, Available at http://dx.doi.org/10.1007/978-3-319-10428-7_36, 2014, pp. 481–496.

[27] J.T. Linderoth and M.W.P. Savelsbergh, *A computational study of search strategies for mixed integer programming*, INFORMS Journal on Computing 11 (1999), pp. 173–187.

[28] H. Marchand, A. Martin, R. Weismantel, and L.A. Wolsey, *Cutting planes in integer and mixed integer programming*, Discrete Applied Mathematics 123/124 (2002), pp. 391–440.

[29] A. Sabharwal, H. Samulowitz, and C. Reddy, *Guiding Combinatorial Optimization with UCT.*, in *CPAIOR*, Lecture Notes in Computer Science, Vol. 7298, Available at http://dblp.uni-trier.de/db/conf/cpaior/cpaior2012.html#SabharwalSR12, Springer, 2012, pp. 356–361.

[30] SCIP, *SCIP. Solving Constraint Integer Programs.*, http://scip.zib.de/ (2016).

[31] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch, *ParaSCIP – a parallel extension of SCIP*, in *Competence in High Performance Computing 2010*, Springer, 2012, pp. 135–148.

[32] *SoPlex. An open source LP solver implementing the revised simplex algorithm.*, http://soplex.zib.de/ (2016).

[33] D.T. Wojtaszek and J.W. Chinneck, *Faster mip solutions via new node selection rules*, Computers & Operations Research 37 (2010), pp. 1544–1556.

[34] Xpress, *FICO Xpress-Optimizer* (2016), http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx.

[35] U. Zahavi, A. Felner, N. Burch, and R.C. Holte, *Predicting the performance of ida* using conditional distributions*, J. Artif. Intell. Res. (JAIR) 37 (2010), pp. 41–83, Available at http://dx.doi.org/10.1613/jair.2890.

Appendix

Table 4: Instancewise results from Section 6.2.

	default	estim	oracle	rank-1	default	estim	oracle	rank-1
	t (sec)	*t* (sec)	*t* (sec)	*t* (sec)	*n*	*n*	*n*	*n*
10teams	25.6	35.4	33.3	33.3	1272	1612	1552	1612
30n20b8	183.5	151.7	135.6	152.0	14	6	4	6
a1c1s1	7200.0	7200.0	7200.0	7200.0	718786	626920	640772	1815029
acc-tight5	1436.6	61.7	61.6	62.2	16931	608	608	608
aflow30a	13.9	14.5	13.3	13.5	2118	2506	1852	1976
aflow40b	912.9	941.0	681.0	1328.9	151103	112411	69617	323752
air03	1.0	1.1	1.0	1.1	1	1	1	1
air04	70.4	64.1	65.6	65.5	213	146	156	156
air05	37.5	39.3	39.3	39.4	181	309	299	365
app1-2	1118.2	1270.6	976.2	964.7	427	429	306	246
arki001	7200.0	7200.0	7200.0	7200.0	1166439	915413	960617	997478
atlanta-ip	7200.0	7200.0	7200.0	7200.0	14375	9861	9852	260171
bab5	7200.0	7200.0	7200.0	7200.0	25819	17781	17837	17728
beasleyC3	7200.0	7200.0	7200.0	7200.0	796130	1106432	1106852	2065757
bell3a	6.1	5.7	4.2	4.5	22487	23064	22579	23083
bell5	0.8	0.5	0.6	0.6	1140	1226	1218	1224
biella1	781.4	578.7	575.2	1395.4	2133	2538	2436	14468
bienst2	297.4	297.8	301.6	291.8	93988	93988	92643	106272
binkar10_1	177.2	158.3	147.8	137.2	138787	120843	119374	120533
blend2	0.9	1.4	1.4	1.2	412	932	933	634
bley_xl1	430.6	434.6	429.0	431.4	20	20	20	20
bnatt350	1477.1	819.2	818.9	828.6	21343	14092	14092	14092
cap6000	2.7	3.1	2.7	2.8	3788	4064	4080	4561
core2536-691	318.1	321.2	324.1	319.6	218	218	218	481
cov1075	7200.0	7200.0	6923.5	7200.0	1557428	1559145	1635309	1854530
csched010	7200.0	7200.0	7200.0	7200.0	931270	1049236	997781	1128300
danoint	5078.2	4785.0	3836.3	4451.6	1050040	966643	881640	1089980
dcmulti	1.6	1.2	1.2	1.2	322	316	306	261
dfn-gwin-UUM	139.6	133.4	113.9	115.1	66936	61708	63462	60074
disctom	3.6	3.9	3.9	3.8	1	1	1	1
ds	7200.0	7200.0	7200.0	7200.0	523	542	546	546
dsbmip	1.2	1.1	1.4	1.3	15	11	11	11
egout	0.5	0.5	0.5	0.5	1	1	1	1
eil33-2	52.8	57.3	78.9	96.7	735	851	1235	1339
eilB101	436.2	477.7	474.1	323.3	8028	8283	8357	6776
enigma	0.5	0.6	0.6	0.7	954	2759	2759	2759
enlight13	8.6	16.8	119.5	10.7	13479	30211	270890	23151
ex9	38.2	37.2	39.5	37.4	1	1	1	1
fast0507	576.8	574.8	574.8	615.3	1376	1376	1348	1862
fiber	1.6	1.5	3.4	1.7	8	8	5	8
fixnet6	3.0	3.3	7.5	3.3	9	9	8	9
flugpl	0.5	0.5	0.5	0.5	251	115	115	174
gen	0.5	0.5	0.5	0.5	1	1	1	1
gesa2	0.9	1.2	1.1	1.0	2	2	2	2
gesa2-o	1.2	1.3	1.2	1.6	5	5	5	5
gesa3	1.6	1.6	2.4	2.0	7	7	6	7
gesa3_o	1.6	1.8	1.9	1.7	8	8	8	8

Table 5: Continued instancewise results from Section 6.2.

	`default`	`estim`	`oracle`	`rank-1`	`default`	`estim`	`oracle`	`rank-1`
	t (sec)	*t* (sec)	*t* (sec)	*t* (sec)	*n*	*n*	*n*	*n*
glass4	7200.0	7200.0	7200.0	7200.0	16199130	14938613	14873922	19924545
gmu-35-40	7200.0	7200.0	7200.0	7200.0	13065327	14149261	14168881	22385757
gt2	0.5	0.5	0.5	0.5	1	1	1	1
harp2	3700.6	4479.6	4430.6	7200.0	12630591	11703033	11722399	22409557
iis-100-0-cov	1663.9	1722.3	1305.2	1358.8	102734	105711	85533	89706
iis-bupa-cov	6142.9	6512.3	5434.2	5552.2	182534	179812	172416	182329
iis-pima-cov	1383.2	1388.3	1388.6	610.8	20364	20278	20296	7935
khb05250	0.5	0.6	1.0	0.6	3	3	2	3
l152lav	2.5	3.0	3.2	3.3	49	92	92	92
lectsched-4-obj	399.0	109.6	200.4	200.1	24222	8296	9683	9683
lseu	0.6	0.8	0.9	0.8	336	606	602	379
m100n500k4r1	7200.0	7200.0	7200.0	7200.0	7184542	6987522	6993419	2046170
macrophage	7200.0	7200.0	7200.0	7200.0	1251604	1233931	1236445	1239753
manna81	0.8	0.5	0.8	0.6	1	1	1	1
map18	424.3	433.5	382.6	438.6	393	315	333	305
map20	335.0	333.3	333.9	327.6	299	299	319	315
markshare1	7200.0	7200.0	7200.0	7200.0	73327325	76824489	75830406	43086938
markshare2	7200.0	7200.0	7200.0	7200.0	60920471	60781960	60892446	28720432
mas74	565.4	583.8	516.2	440.1	2834519	2767121	2760117	2594501
mas76	66.7	79.2	118.7	56.2	404939	471714	848147	436756
mcsched	211.9	172.9	173.4	159.9	19507	15565	15471	14275
mik-250-1-100-1	365.1	362.7	236.8	278.7	943440	943440	595707	683166
mine-166-5	31.0	30.7	30.7	31.0	2045	2045	1997	2045
mine-90-10	256.3	228.4	232.2	199.4	77784	68094	67313	57851
misc03	1.2	1.1	1.0	1.2	139	123	137	170
misc06	0.7	0.5	0.8	0.5	4	4	6	4
misc07	14.4	13.1	11.2	12.3	21721	20003	15439	17292
mitre	6.0	5.9	6.7	6.0	1	1	1	1
mkc	7200.0	7200.0	7200.0	7200.0	2524672	2989875	2985313	3871184
mod008	0.9	0.7	1.2	1.1	7	7	4	7
mod010	0.8	0.7	0.7	0.7	2	7	7	7
mod011	176.8	180.3	152.0	145.8	1229	1229	1021	1068
modglob	1.4	1.5	1.5	1.3	905	905	739	820
momentum1	7200.0	7200.0	7200.0	7200.0	44070	15148	15305	15331
momentum2	7200.0	7200.0	7200.0	7200.0	90508	99580	86526	74211
msc98-ip	7200.0	7200.0	7200.0	7200.0	3391	18438	10164	10162
mspp16	2579.3	2841.5	5437.7	2838.0	51	57	47	57
mzzv11	267.9	266.0	258.1	262.7	1999	1999	1908	1975
mzzv42z	340.3	337.7	338.4	316.6	534	534	536	1012
n3div36	7200.0	7200.0	7200.0	7200.0	250934	260655	372168	345004
n3seq24	7200.0	7200.0	7200.0	7200.0	393	392	393	393
n4-3	542.3	564.0	472.5	633.3	32231	33154	29104	44646
neos-1109824	156.1	154.0	103.4	123.3	21927	22678	10652	14781
neos-1337307	7200.0	7200.0	7200.0	7200.0	370421	553458	519383	391710
neos-1396125	766.1	925.5	856.8	778.5	61200	69115	59721	70372
neos-1601936	7200.0	7200.0	7200.0	7200.0	6755	1615	1606	1615
neos-476283	275.9	282.0	279.6	279.8	685	685	667	855
neos-686190	93.8	118.6	110.5	109.8	7264	10378	9405	9445
neos-849702	174.7	558.8	559.6	559.9	6115	48917	48917	48917
neos-916792	406.2	454.3	399.2	593.9	106472	123066	124088	210792
neos-934278	7200.0	7200.0	7200.0	7200.0	889	1133	992	1095
neos13	1514.8	1727.5	1725.1	1738.6	4422	3230	3230	3209
neos18	32.1	29.8	31.9	31.6	6778	5601	5179	6249
net12	2532.6	2746.2	3018.6	3473.0	3864	4985	5016	4605
netdiversion	7200.0	6630.7	6581.5	6580.6	72	119	113	113
newdano	3557.7	3573.5	3704.2	3242.0	2083404	2083404	1993781	2002198
noswot	177.5	180.3	175.1	173.1	829543	829543	436956	455271
ns1208400	1870.2	1279.0	1519.8	1062.9	3118	2777	2785	2772
ns1688347	738.9	275.2	388.7	380.6	6667	2609	3905	4330
ns1758913	7200.0	7200.0	7200.0	7200.0	2	2	2	2
ns1830653	440.3	371.5	382.9	394.2	41218	46638	36114	46887
nsrand-ipx	7200.0	7200.0	7200.0	7200.0	1599798	1763180	1758600	1730377
nw04	24.5	24.4	25.3	24.9	11	11	6	11
opm2-z7-s2	794.6	789.9	794.6	1220.6	2092	2092	2094	15231
opt1217	0.5	0.5	0.5	0.5	1	1	1	1

Table 6: Continued instancewise results from Section 6.2.

	`default`	`estim`	`oracle`	`rank-1`	`default`	`estim`	`oracle`	`rank-1`
	t (sec)	*t* (sec)	*t* (sec)	*t* (sec)	*n*	*n*	*n*	*n*
p0033	0.5	0.5	0.5	0.5	1	1	1	1
p0201	1.8	1.7	1.8	1.7	67	67	59	65
p0282	0.5	0.6	0.8	0.6	3	3	1	3
p0548	0.5	0.5	0.5	0.5	1	1	1	1
p2756	1.6	1.2	1.4	1.2	137	9	9	9
pg5_34	1287.1	1338.1	1297.1	1400.7	291242	291323	273355	305210
pigeon-10	7200.0	7200.0	7200.0	7200.0	17116573	17457654	10565366	17502182
pk1	64.2	65.2	53.8	55.0	284323	284323	281331	281341
pp08a	1.3	1.1	1.4	1.3	221	225	231	253
pp08aCUTS	1.1	1.4	1.4	1.3	194	165	149	153
protfold	7200.0	7200.0	7200.0	7200.0	10226	11588	11587	11595
pw-myciel4	3542.8	3550.0	2199.1	2629.2	712713	712713	368355	433819
qiu	79.9	81.6	77.8	77.2	12604	12618	12616	12629
qnet1	8.3	4.5	5.3	4.5	36	7	7	7
qnet1_o	6.6	5.1	5.3	5.4	16	6	6	6
rail507	242.4	239.9	235.7	239.6	799	799	855	865
ran16x16	291.3	283.9	231.0	276.4	368022	346094	265832	373581
rd-rplusc-21	7200.0	7200.0	7200.0	7200.0	77078	61764	60360	70998
reblock67	253.3	251.3	172.6	246.5	109664	109664	57072	105846
rentacar	2.7	2.6	3.5	2.8	4	4	4	4
rgn	0.5	0.5	0.5	0.5	1	1	1	1
rmatr100-p10	135.1	135.8	131.0	130.7	851	851	909	909
rmatr100-p5	302.9	304.2	267.6	280.0	420	451	483	439
rmine6	6096.9	2287.9	2246.4	2263.0	2004491	742664	736822	738018
rocII-4-11	463.1	204.7	433.9	299.9	40477	11718	30330	17308
rococoC10-001000	1217.3	1011.1	876.6	1274.5	203201	174936	135810	224776
roll3000	7200.0	3082.7	1387.3	5522.8	2781398	1063691	423972	2331855
rout	40.1	32.2	27.9	28.2	26664	20646	18547	18646
satellites1-25	660.2	995.5	765.3	913.5	3064	2648	1588	2212
set1ch	0.7	0.8	0.9	0.7	9	9	9	9
seymour	7200.0	7200.0	7200.0	7200.0	150798	146737	146590	146047
sp97ar	7200.0	7200.0	7200.0	7200.0	6289	8065	8101	7619
sp98ic	7200.0	7200.0	7200.0	7200.0	131473	137628	137322	185344
sp98ir	106.6	86.3	81.3	69.2	8210	5379	5375	4630
stein27	1.0	1.1	1.2	0.7	3905	3905	3973	3607
stein45	12.2	13.1	11.3	10.7	47352	47352	50336	46693
stp3d	7200.0	7200.0	7200.0	7200.0	1	1	1	1
swath	7200.0	7200.0	7200.0	7200.0	1231280	1214072	1220123	1672194
tanglegram1	1161.2	1138.7	1138.6	1142.8	37	37	37	37
tanglegram2	14.9	15.1	14.7	14.7	5	5	5	5
timtab1	390.1	388.1	376.2	377.6	870361	868207	896679	965573
timtab2	7200.0	7200.0	7200.0	7200.0	9144342	14814841	9027482	14855771
tr12-30	1814.6	1521.3	1505.3	1454.0	1471731	1186814	1162158	1306275
triptim1	2791.3	981.7	997.6	981.7	47	4	4	4
unitcal_7	1304.6	1469.5	1230.3	1271.4	23265	27125	19216	19861
vpm1	0.5	0.5	0.5	0.5	1	1	1	1
vpm2	1.3	1.2	1.7	1.4	294	218	224	206
vpphard	7200.0	7200.0	7200.0	7200.0	7476	6321	6292	6321
zib54-UUE	3829.7	5375.5	2703.8	2625.2	539744	706521	296047	294878