# Exploiting Solving Phases for Mixed-Integer Programs

**Gregor Hendel**

**Abstract** Modern MIP solving software incorporates dozens of auxiliary algorithmic components for supporting the branch-and-bound search in finding and improving solutions and in strengthening the relaxation. Intuitively, a dynamic solving strategy with an appropriate emphasis on different solving components and strategies is desirable during the search process. We propose an adaptive solver behavior that dynamically reacts on transitions between the three typical phases of a MIP solving process: The first phase objective is to find a feasible solution. During the second phase, a sequence of incumbent solutions gets constructed until the incumbent is eventually optimal. Proving optimality is the central objective of the remaining third phase. Based on the MIP-solver SCIP, we demonstrate the usefulness of the phase concept both with an exact recognition of the optimality of a solution, and provide heuristic alternatives to make use of the concept in practice.

## 1 Introduction

The availability of sophisticated solving software technology based on the branch-and-bound approach [8] has made Mixed integer programming (MIP) the modeling tool of choice for many practical optimization problems. One of its main advantages is that after termination, branch-and-bound provides a proof of optimality for the best found solution. In many situations, however, practical limits on the run time and memory consumption prevent the search from completing the proof, although the solution found at termination might already be optimal. During the search process, we typically observe three phases: The first phase until a feasible solution is found, a second phase during which a sequence of improving solutions gets constructed, and a third phase during which the remaining search tree must be fully explored to prove

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).

G. Hendel (✉)
Konrad Zuse Zentrum für Informationstechnologie, Takustraße 7, 14195 Berlin, Germany
e-mail: hendel@zib.de



K.F. Dœrner et al. (eds.), *Operations Research Proceedings 2015*,
Operations Research Proceedings, DOI 10.1007/978-3-319-42902-1_1

optimality. In [6] we empirically demonstrated that the MIP solver SCIP [1] spends more than 40 % of its average solving time during the third phase.

Since every phase emphasizes a different goal of the solving process, it seems natural to pursue these goals with different search strategies to achieve the phase objective as fast as possible. Research on adaptive solver behavior that reacts on solving phases naturally poses the question how the solver should guess that the current incumbent is optimal prior to termination.

There has been little work on such heuristic criteria for deciding whether a solution can be assumed to be optimal. Such criteria cannot be expected to be exact because the decision problem of proving whether a given solution is optimal is still $\mathcal{NP}$-complete in general, hence the term "heuristic".

A bipartion of the solving process has already been suggested in the literature, see [9] for an overview and further references, where the proposed strategies solely involve the node selection in use. Our suggested three-phase approach gives a more refined control of the solver behaviour.

The remainder of the paper is organized as follows: We formally introduce Mixed-Integer Programs and and the concept of solving phases in Sect. 2. The main novelty of this paper are *heuristic transitions* for deciding when the solver should stop searching for better solutions and concentrate on proving optimality. We present two heuristic transitions that take into account global information of the list of open subproblems in Sect. 3. We conclude with a computational study of the proposed adaptive solvers in Sect. 4.

## 2 Solving Phases in Mixed Integer Programming

Let $A \in \mathbb{R}^{m \times n}$ a real matrix, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, let $l, u \in \mathbb{R}^n_\infty$ and $\mathcal{I} \subseteq \{1, \ldots, n\}$, where $n, m \in \mathbb{N}$. A *mixed-integer program (MIP)* is a minimization problem $P$ of the form

$$c^{\text{opt}} := \inf\{c^t x \,:\, x \in \mathbb{R}^n,\ Ax \leq b,\ l \leq x \leq u,\ x_j \in \mathbb{Z} \quad \forall j \in \mathcal{I}\}.$$

A vector $y \in \mathbb{R}^n$ is called a *solution* for $P$, if it satisfies all linear constraints, bound requirements, and integrality restrictions of $P$. We call $\mathcal{I}$ the set of *integer variables* of $P$. A solution $y^{\text{opt}}$ that satisfies $c^t y^{\text{opt}} = c^{\text{opt}}$ is called *optimal*. The *LP-relaxation* of $P$ is defined by dropping the integrality restrictions. By solving the LP-relaxation to optimality, we obtain a *lower bound* $\delta$ (also called dual bound) on the optimal objective of $P$. All commercial and noncommercial general purpose MIP solvers are based on the *branch-and-bound* procedure [8], which they extend by various auxiliary components such as *primal heuristics* [5], *cutting plane routines*, and *node presolving techniques* for improving the primal or dual convergence of the method.

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

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

A primal gap of 0 % 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. Similarly, we use a *dual gap* $\gamma^*$ to measure the relative distance between $c^{\text{opt}}$ and the proven dual bound $\delta$.

In the context of solving phases, elapsed time since the solving process was started plays an important role. All definitions such as the incumbent solution $\hat{y}$ and its objective (the primal bound) $c^t \hat{y}$ or its dual counter parts $\delta$ and the corresponding gaps $\gamma$ and $\gamma^*$ can be translated into functions of the elapsed time. Let $t_1^* > 0$ denote the point in time when the first solution is found or the *first phase transition*. The *primal gap function* $\gamma : [t_1^*, \infty] \mapsto [0, 100]$ measures the primal gap at every point in time $t \geq t_1^*$ during solving by calculating the primal gap for the best incumbent $\hat{y}(t)$ found until $t$.

For the solving time $T > 0$ for $P$, we partition the solving time interval $[0, T]$ into three disjoint *solving phases*:

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

Every solving phase is named after its main primal objective of finding a first and optimal solution in $\mathscr{P}_1$ and $\mathscr{P}_2$, respectively, and proving optimality during $\mathscr{P}_3$. We presented promising strategies for each phase in [6]; During the Feasibility phase, we search for feasible solutions with a two-stage node selection strategy combining a uct [10] and depth-first strategy with restarts together with an inference branching rule. The Improvement phase is conducted with the default search strategy of SCIP except for the use of uct inside Large Neighborhood Search heuristics. For the Proof phase, we deactivate primal heuristics, and apply cutting planes periodically during a depth-first search traversal of the remaining search tree. Note that a phase-based solver that uses different settings after a heuristic phase transition remains exact; the use of different settings based on the heuristic phase transition might only influence the performance of the solver to finish the solving process.

The desired moment in time when a phase-based solver should switch from an improvement strategy to a proof strategy is given by the *second phase transition*

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

Because of the practical impossibility to detect $t_2^*$ exactly before the solving process finishes, we dedicate the next section to introduce *heuristic phase transitions* for our phase-based solver.

## 3 Heuristic Phase Transitions

We propose to use properties of the frontier of open subproblems during the solving process as heuristic phase transitions. Let $\mathscr{Q}$ denote the set of open subproblems. We call $Q \in \mathscr{Q}$ an *active node* and denote by $d_Q$ the *depth* of $Q$ in the search tree. If the solving process has not found an optimal solution yet, there exists an active node $Q \in \mathscr{Q}$ that contains it. We use the *best-estimate* [3] to circumvent the absence of true knowledge about best solutions in the unexplored subtrees. After solving the LP-relaxation of a node $P$ with solution $\tilde{y}_P$, the *best-estimate* defined as

$$\hat{c}_P = c^t \tilde{y}_P + \sum_{j:(\tilde{y}_P)_j \notin \mathbb{Z}} \min\{\Psi_j^- \cdot \left((\tilde{y}_P)_j - \lfloor (\tilde{y}_P)_j \rfloor\right), \Psi_j^+ \cdot \left(\lceil (\tilde{y}_P)_j \rceil - (\tilde{y}_P)_j\right)\}$$

is an estimate of the best solution objective attainable from $P$ by adding the minimum *pseudo-costs* [3] to make all variables $j \in \mathscr{I}$ with fractional LP-solution values $(\tilde{y}_P)_j \notin \mathbb{Z}$ integral, where we use average unit gains $\Psi_j^-, \Psi_j^+$ over all previous branching decisions. For active nodes $Q \in \mathscr{Q}$, an initial estimate can be calculated from the parent estimate and the branching decision to create $Q$.

**Definition 1** (*active-estimate transition*) We define the *active-estimate transition* as the first moment in time $t_2^{\text{estim}}$ when the incumbent objective is smaller than the minimum best-estimate amongst all active nodes, i.e.

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

In practice, the best-estimate may be very inaccurate and over- or underestimate the true objective value obtainable from a node, which may lead to an undesirably early or late active-estimate transition. In order to drop the use of the actual incumbent objective, we introduce another transition that compares all active and already processed nodes only at their individual depths. Let the *rank-1 nodes* be defined as

$$\mathscr{Q}^{\text{rank-1}}(t) := \{Q \in \mathscr{Q}(t) : \; \hat{c}_Q \leq \inf\{\hat{c}_{Q'} : Q' \text{ processed before } t, \; d_{Q'} = d_Q\}\}.$$

$\mathscr{Q}^{\text{rank-1}}(t)$ contains all active nodes with very small lower bounds or near-integral solutions with small pseudo-cost contributions compared to already processed nodes at the same depth.

**Definition 2** (*rank-1 transition*) The *rank-1 transition* is the moment in time when $\mathscr{Q}^{\text{rank-1}}(t)$ becomes empty for the first time:

$$t_2^{\text{rank-1}} := \min\{t \geq t_1^* : \mathscr{Q}^{\text{rank-1}}(t) = \emptyset\}. \tag{2}$$

The main difference between the rank-1 and the active-estimate transitions is that the former does not compare an incumbent objective with the node estimates. Note that the rank-1 criterion $\mathscr{Q}^{\text{rank-1}} = \emptyset$ is never satisfied as long as there exist active

nodes which are deeper in the tree than any previously explored node. The name of this transition is inspired by a node rank definition that requires full knowledge about the entire search tree at completion, see [6] for details.

## 4 Computational Results

We conducted a computational study to investigate the performance benefits of a phase-based solver that reacts on phase transitions with a change of its search strategy. Apart from the default settings of SCIP we tested an `oracle` that detects the second phase transition exactly, `estim` uses the active-estimate transition (1), and `rank-1` the rank-1 transition (2). For the latter two, we also required that at least 50 branch-and-bound nodes were explored. At the time a criterion is met, we assume that the current incumbent is optimal and let the solver react on this assumption by switching to settings for the Proof phase. We tested with a time limit of 2 h on the 168 instances from three publicly available MIPLIB libraries [2, 4, 7]. We excluded four instances for which no optimal solution value was known by the time of this writing.

In Table 1, we present the shifted geometric means of the measured running times of the different settings with a shift of 10 s. We also show the percentage time compared to `default`, the number of solved instances for every setting, and $p$-values obtained from a two-sided Wilcoxon signed rank test that takes into account logarithmic shifted quotients, see [6] for details. The `oracle` setting could solve three instances more than the default setting. Over the entire test set, we observe improvements in the shifted geometric mean solving time for every new setting, where the highest improvement of 5.6 % was obtained with the `oracle`-setting. With the `rank-1` setting, we obtain a similar speed-up of 5.4 %. Both are accompanied by small $p$-values of 0.013 and 0.008. The table also shows the results for two instance groups based on the performance of the slowest of the four tested algorithms. On the 73 easy instances, `oracle` is slower than `default` by almost 5 %, whereas `rank-1` is the fastest amongst the tested settings. The computational overhead of the reactivated separation during the Proof phase seems to outweigh its benefits on this easy group. The $p$-values, however do not reveal any of the settings to be significantly different from `default`.

**Table 1** Shifted geometric mean results for $t$ (s) and number of solved instances

| | All instances | | | | Easy (max $t \leq 200$) | | | Hard (max $t > 200$) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | # solv. | $t$ (s) | % | $p$ | $t$ (s) | % | $p$ | $t$ (s) | % | $p$ |
| `default` | 127 | 257.0 | 100.0 | | 11.7 | 100.0 | | 1992.8 | 100.0 | |
| `estim` | 129 | 245.0 | 95.3 | 0.905 | 11.7 | 100.7 | 0.521 | 1827.1 | 91.7 | 0.488 |
| `oracle` | 130 | 242.7 | 94.4 | 0.013 | 12.2 | 104.9 | 0.410 | 1766.3 | 88.6 | 0.000 |
| `rank-1` | 128 | 243.1 | 94.6 | 0.008 | 11.3 | 97.0 | 0.226 | 1832.9 | 92.0 | 0.026 |

The results on the hard instances show more pronounced improvements with all new settings by up to 11.4 % obtained with the `oracle` setting. The setting `estim` improves the time by 8.2 % but the corresponding $p$-value of 0.488 does not identify this improvement as significant. A smaller time improvement of 8 % with the `rank-1` setting is indicated as significant by a $p$-value of less than 5 %. This result indicates a more consistent improvement over the entire test set for `rank-1`, whereas the active-estimate transition could rather improve the performance on a few outliers.

## 5 Conclusions

In our experiment, the use of a phase-specific solver adaptation could significantly improve the running time, especially on harder instances. Furthermore, we introduced two heuristic phase transitions that yielded performance improvements similar to what can be obtained in principle if we could determine the phase transitions exactly, which is an important first step to make use of such adaptive solver behavior in practice. We attribute the significant improvements with the exact and rank-1 transitions in particular to the judicious reactivation of cutting plane separation locally in the tree at the cost of deactivating primal heuristics. Future work on solving phases could comprise experiments with different heuristic phase transitions, or base the work distribution between primal heuristics and separation on more local properties that are specific to the subtree.

## References

1. Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007)
2. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. **34**(4), 1–12 (2006)
3. Bénichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribière, G., Vincent, O.: Experiments in mixed-integer programming. Math. Program. **1**, 76–94 (1971)
4. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Optima **58**, 12–15 (1998)
5. Fischetti, M., Lodi, A.: Heuristics in mixed integer programming. In: Cochran, J.J., Cox, L.A., Keskinocak, P., Kharoufeh, J.P., Smith, J.C. (eds.) Wiley Encyclopedia of Operations Research and Management Science. Wiley, New York (2010). (Online publication)
6. Hendel, G.: Empirical analysis of solving phases in mixed integer programming. Master thesis, Technische Universität Berlin (2014)
7. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Math. Program. Comput. **3**(2), 103–163 (2011)
8. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960)

9. Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of search strategies for mixed integer programming. INFORMS J. Comput. **11**(2), 173–187 (1999)
10. Sabharwal, A., Samulowitz, H., Reddy, C.: Guiding combinatorial optimization with UCT. In: Beldiceanu, N., Jussien, N., Pinson, E. (eds.) CPAIOR. Lecture Notes in Computer Science, vol. 7298, pp. 356–361. Springer, New York (2012)