Heuristics of the Branch-Cut-and-Price-Framework SCIP

Timo Berthold*

Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany
berthold@zib.de

Summary. In this paper we give an overview of the heuristics which are integrated into the open source branch-cut-and-price-framework SCIP. We briefly describe the fundamental ideas of different categories of heuristics and present some computational results which demonstrate the impact of heuristics on the overall solving process of SCIP.

1 Introduction

A lot of problems arising in various areas of Operations Research can be formulated as *Mixed Integer Programs (MIP)*. Although MIP-solving is an $\mathcal{NP}$-hard optimization problem, many practically relevant instances can be solved in reasonable time. The standard exact method for solving MIPs is *branch-and-cut*, a combination of LP-based branch-and-bound and cutting plane techniques. Besides that, heuristics (Greek εὑρίσκειν – to find) are incomplete methods which quickly try to construct feasible solutions of high quality, but without any guarantee to find one.

In state-of-the-art MIP-solvers like the branch-cut-and-price-framework SCIP (Solving Constraint Integer Programs) [1, 3] heuristics play a major role in finding and improving feasible solutions at early stages of the solution process. This helps to reduce the overall computational effort, guides the remaining search process, and proves the feasibility of the MIP model. Furthermore, a heuristic solution with a small gap to optimality often is sufficient for practical applications.

Overall, there are 23 heuristics integrated into SCIP version 1.00. They can be roughly subclassified into four categories: rounding, diving, objective diving, and large neighborhood search heuristics. In the

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

remainder, we will give a short introduction into these strategies and afterwards we will present some computational results. For more detail, we refer to Achterberg [1] and Berthold [6].

2 Rounding Heuristics

All rounding heuristics in SCIP work in the following way: they take an LP-feasible but fractional point – normally the optimum of some LP-relaxation – and iteratively round the fractional variables. Thereby, the number of fractional variables is reduced one by one in each iteration step (except if a shift is performed, see below). Regarding rounding heuristics, the most important issue is, not to loose the LP-feasibility during the iteration process, or if so, try to immediately recover LP-feasibility.
There are four rounding heuristics in SCIP:

- *Simple Rounding* only performs roundings, which assure to keep feasibility;
- *Rounding* conducts roundings, which potentially violate some constraints and reduces existent violations by further roundings;
- *Shifting* is allowed to change (shift) the values of integral or continuous variables in order to recover feasibility;
- *Integer Shifting* proceeds like Shifting, but does not consider continuous variables. If it succeeds, it solves an LP in order to set the continuous variables to their optimal value.

Each of these procedures is an extension of the ones which are listed before it. The latter are more powerful, but also more expensive in terms of running time and therefrom they are applied less frequently.

3 Diving Heuristics

The principal idea of diving heuristics comes from the branch-and-bound procedure. They iteratively round some fractional variable and resolve the LP-relaxation, simulating a depth-first-search in the tree. In doing so, diving heuristics use a special branching rule which tends towards feasibility and not primary towards a good subdivision of the problem, as common branching rules do.

The six diving heuristics implemented in SCIP mainly differ in the applied branching rule. It chooses a variable with:

- *Fractional Diving:* smallest fractionality;

- *Coefficient Diving:* smallest number of potentially violated rows;
- *Linesearch Diving:* greatest difference of root solution and current LP solution;
- *Guided Diving:* smallest difference to the best known integral solution;
- *Pseudocost Diving:* smallest ratio of estimated objective increase if rounding to either direction;
- *Vectorlength Diving:* smallest ratio of potential objective change and number of affected constraints.

In [6], it is shown that none of them dominates the others in terms of performance.

4 Objective Diving Heuristics

Heuristics of this category iteratively manipulate the objective function and resolve the LP-relaxation in order to reach an integral vertex of the LP-polyhedron. They perform "soft roundings" by adding punishment terms to the objective instead of performing "hard roundings", i.e., fixing variables like the heuristics of Sections 2 and 3.

There are actually three objective diving heuristics in SCIP: Objective Pseudocost Diving, Rootsolution Diving and the Objective Feasibility Pump. In our computational studies, the latter one proved to be superior to the others.

The Feasibility Pump was first described by Fischetti et al. [10, 5], the version which is implemented in SCIP was introduced by Achterberg and Berthold [2]. By taking the original objective of the MIP into account, the Objective Feasibility Pump is able to produce solutions of a much better objective value in a comparable running time.

5 LNS Heuristics

Large neighborhood search (LNS) heuristics solve a sub-MIP of the original MIP in order to investigate a neighborhood of a special point, e.g., the best known integral solution (*incumbent*). This sub-MIP is created by fixing a sufficient number of variables or adding very restrictive constraints. The hope is that the sub-MIP is much easier to solve, but still contains solutions of high quality.

Four of the five LNS heuristics available in SCIP are improvement heuristics, i.e., they take some feasible solution as a starting point:

- *Local Branching* [11] adds a distance constraint which allows only a certain number of variables to differ from their value in the incumbent;
- *RINS* [9] fixes variables which take identic values in the current node's LP-relaxation and the incumbent;
- *Crossover* [6] fixes variables which take identic values in a certain number of feasible solutions;
- *Mutation* [6] randomly fixes variables to their incumbent value.

In contrast to these four, *RENS* [6, 7] is an LNS rounding heuristic. It fixes all variables which take integral values in the optimum of the LP-relaxation (often more than 90%) and changes the bounds to the nearest integers for fractional variables. This implies that all integer variables of the sub-MIP are binary.

By completely solving the RENS sub-MIP, one is able to determine whether a point can be rounded to an integral solution and which one is the best possible rounding. Furthermore, a slightly restricted version of RENS proves to be a reasonable start heuristic.

6 Computational Results

The computational experiments reported here were obtained with SCIP version 0.82b running on a 3.80 GHz Intel Pentium 4 with 2 GB RAM, using CPLEX 10.0 as underlying LP-solver. We chose a test set of 129 instances taken from the Miplib 3.0 [8], the Miplib2003 [4] and the MIP collection of Mittelmann [12].

First, we evaluated the individual impact of the 15 heuristics which are used by default. For each heuristic, we investigated the change of performance caused by deactivating it. We compared the geometric means of the running time and the number of branch-and-bound-nodes taken over the 97 instances which could be solved to optimality within an hour, using SCIP with default settings. For the other instances we compared the primal-dual gap after running SCIP for an hour.

We observed that deactivating a single heuristic only has a small impact; the geometric means of the running time and the number of branch-and-bound-nodes always changed by less than 5%, except for the Objective Feasibility Pump (12% and 30%, respectively).

On the other hand, deactivation of all available heuristics leads to a significant deterioration: the geometric mean of the running time and the number of branch-and-bound-nodes raises by a factor of two, the remaining gap by about 50%. There are considerably less instances which are solved to optimality within an hour, or for which at least one feasible solution is found, respectively.

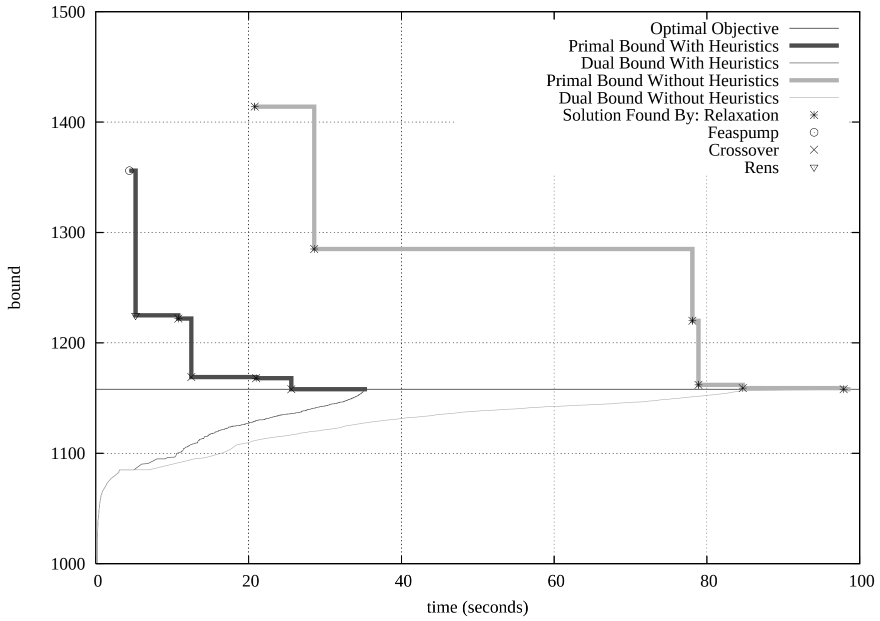

Fig. 1. Instance aflow30a: developing of primal and dual bound if SCIP runs with (dark) and without any heuristics (light)

Figure 1 exemplarily shows the developing of the primal and dual bound for two runs of SCIP 0.82b with an instance taken from the Miplib2003 [4]: one with the default heuristics and one without any heuristics activated.

As expected, SCIP with heuristics is faster in finding the first feasible solution, an optimal solution and proving the optimality. We also observe that the dual bound raises faster immediately after feasible solutions were found and that even the first improvement by an integral node LP-relaxation occurs at an earlier step in time. This is due to the fact that with the knowledge of a good primal bound, one is able to prune suboptimal nodes, fix additional variables, which itself leads to stronger cuts and so forth.

All these results emphasize that heuristics are an important part of a branch-cut-and-price-framework and point out the importance of the interaction between different heuristics.

References

1. T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.
2. T. Achterberg and T. Berthold. Improving the Feasibility Pump. *Discrete Optimization*, Special Issue 4(1):77–86, 2007.
3. T. Achterberg, T. Berthold, M. Pfetsch, and K. Wolter. SCIP (Solving Constraint Integer Programs). `http://scip.zib.de`.
4. T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. `http://miplib.zib.de`.
5. L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, Special Issue 4(1):77–86, 2007.
6. T. Berthold. Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006.
7. T. Berthold. RENS - Relaxation Enforced Neighborhood Search. ZIB-Report 07-28, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2007. `http://opus.kobv.de/zib/volltexte/2007/1053/`.
8. R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, (58):12–15, 1998.
9. E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming A*, 102(1):71–90, 2004.
10. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming A*, 104(1):91–104, 2005.
11. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming B*, 98(1-3):23–47, 2003.
12. H. Mittelmann. Decision tree for optimization software: Benchmarks for optimization software. `http://plato.asu.edu/bench.html`.