Hybrid Branching

Tobias Achterberg[1] and Timo Berthold[2⋆]

[1] ILOG, an IBM company, Ober-Eschbacher Str. 109, 61352 Bad Homburg, Germany
tachterberg@ilog.de
[2] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany berthold@zib.de

State-of-the-art solvers for Constraint Satisfaction Problems (CSP), Mixed Integer Programs (MIP), and satisfiability problems (SAT) are usually based on a branch-and-bound algorithm. The question how to split a problem into subproblems (*branching*) is in the core of any branch-and-bound algorithm. Branching on individual variables is very common in CSP, MIP, and SAT. The rules, however, which variable to choose for branching, differ significantly. In this paper, we present hybrid branching, which combines selection rules from all three fields.

Branching Rules. In MIP, *reliability pseudocost branching* [2] is the current state-of-the-art. This rule estimates the objective change in the LP relaxation when branching downwards and upwards. It uses the average objective gains per unit change, taken over all nodes, where this variable has been chosen for branching. The resulting two values are called the *pseudocosts* of a variable.

In CSP and SAT, where no objective function is available, one may better estimate the impact of a branching by taking the number of implied reductions of other variable domains into account [4]. In analogy to the pseudocosts, we call the estimated numbers of implied reductions the *inference values* of a variable.

In pure SAT solvers, learning short, valid conflict clauses from the analysis of infeasible subproblems is one of the key ingredients [5]. The *variable state independent decaying sum (VSIDS)* [6] branching strategy, which is a common rule in SAT solving, prefers variables that have been used to create recent conflict clauses. We call the VSIDS the *conflict values* of a variable.

The idea to use the average lengths of the conflict clauses a variable appears in for branching was recently suggested by Kilinc et. al. [3]. We call this the *conflict lengths* of a variable.

As noted above, all the described measures exist twice for each variable: for upwards and downwards branching. Therefore, one has to combine them into a single score value, which we do by multiplication [1].

Hybrid Branching. *Hybrid branching* combines all four selection criteria into a single one and additionally includes a score which is based on the number of subproblems that could be pruned due to branching on this variable, called the *cutoff values*. We first normalize all the five individual values by mapping them into the interval $[0, 1)$. Afterwards, we take a weighted sum of them that puts a high weight on the pseudocosts, a medium weight on the conflict values and lengths, and a low weight on the inference and cutoff values.

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

Table 1. Geometric means of time (in seconds) and branch-and-bound nodes over four test sets

test set	MIPLIB2003		Cor@l		Cor@l-BP		Infeasible	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
reliability	450.4	5091	803.6	4110	672.4	2145	290.7	5612
hybrid	445.6	5051	735.0	3575	577.2	1681	166.0	1998
ratio	1.01	1.01	1.09	1.15	1.16	1.28	1.75	2.81

Computational Results. We tested our new approach on a set of infeasible binary programs (BPs) and two libraries of general MIP instances which are publicly available: the MIPLIB2003 and the Cor@l collection. All computations were performed on a PowerEdge$^{\text{TM}}$ 1955 Xeon 5150 with 4 MB cache and 8 GB RAM. A time limit of one hour was imposed.

We incorporated hybrid branching into the constraint integer programming framework SCIP [1], version 1.1.0.5, using SoPlex 1.4.1 as underlying LP solver. We compared hybrid branching against reliability pseudocost branching, which is state-of-the-art in MIP solving. The shifted geometric means of the running times and the branch-and-bound nodes were used as performance measures.

It turns out that for the MIPLIB2003, both branching rules perform equally good, the difference is 1% in mean. For the Cor@l library, hybrid branching outperforms reliability branching, the running time increases by 9%, the number of nodes by 15%, when using reliability branching instead of hybrid branching. If we only regard binary programs, which are roughly a third of the Cor@l test set, the difference in performance is even larger. This meets our expectations, since the inference and conflict values are especially meaningful for 0-1 variables.

The results for BPs gave rise to the last experiment, which showed that for a set of infeasible binary programs, reliability branching is 75% slower and nearly triplicates the number of branch-and-bound nodes. The conflict values and conflict lengths arose from the analysis of infeasible subproblems, which explains that taking them into account is crucial for handling infeasible MIPs.

Overall, hybrid branching is a successful integration of CSP, SAT, and MIP technologies, which enables to solve standard MIP problems faster. By now, hybrid branching is used as default branching rule in SCIP.

References

1. T. Achterberg, *SCIP: solving constraint integer programs*, Mathematical Programming Computation, issue 1 (2008).
2. T. Achterberg, T. Koch, and A. Martin, *Branching rules revisited*, Operations Research Letters, 33 (2005), pp. 42–54.
3. F. Kilinc Karzan, G. Nemhauser, and M. Savelsbergh, *Information based branching rules in integer programming*. presentation at INFORMS Annual Meeting 2008.
4. C. M. Li and Anbulagan, *Look-ahead versus look-back for satisfiability problems*, in Proc. of CP, Autriche, 1997, Springer, pp. 342–356.
5. J. P. Marques-Silva and K. A. Sakallah, *GRASP: A search algorithm for propositional satisfiability*, IEEE Trans. of Comp., 48 (1999), pp. 506–521.
6. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *Chaff: Engineering an efficient SAT solver*, in Proc. of the DAC, July 2001.