

Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

GERALD GAMRATH

Improving strong branching by domain propagation

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

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

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

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

Improving strong branching by domain propagation

Gerald Gamrath*

Abstract

One of the essential components of a branch-and-bound based mixed-integer linear programming (MIP) solver is the branching rule. *Strong branching* is a method used by many state-of-the-art branching rules to select the variable to branch on. It precomputes the dual bounds of potential child nodes by solving auxiliary linear programs (LPs) and thereby helps to take good branching decisions that lead to a small search tree. In this paper, we describe how these dual bound predictions can be improved by including *domain propagation* into strong branching. Domain propagation is a technique usually used at every node of the branch-and-bound tree to tighten the local domains of variables. Computational experiments on standard MIP instances indicate that our improved strong branching method significantly improves the quality of the predictions and causes almost no additional effort. For a full strong branching rule, we are able to obtain substantial reductions of the branch-and-bound tree size as well as the solving time. Moreover, also the state-of-the-art hybrid branching rule can be improved this way.

This paper extends previous work [18] by the author published in the proceedings of the CPAIOR 2013.

Keywords: mixed-integer programming, branch-and-bound, branching rule, variable selection, strong branching, domain propagation

Mathematics Subject Classification: 90C10, 90C11, 90B40, 9008, 90C57

1 Introduction

Since the invention of the linear programming (LP) based branch-and-bound method for solving mixed-integer linear programs (MIPs) in the 1960s [24, 14], branching rules have been an important field of research in that context, being one of the core parts of the method (for surveys, see [28, 25, 5]). Their task is to split the current problem into two or more disjoint subproblems if the solution to the LP relaxation of the current problem does not fulfill the integrality restrictions. Thereby, it should exclude the LP solution from all subproblems while keeping at least one optimal solution.

The most common way to split the problem is to branch on trivial inequalities, which split the domain of a single variable into two parts (called *variable branching*). Alternatively, branching can be performed on general linear constraints (see [31, 29, 27, 21, 13]) or can create more than two subproblems, cf. [12, 26]. In case of variable branching, the variable to actually branch on is typically chosen with the goal of improving the local dual bound of both created subproblems (also called *child nodes*). This helps to tighten the global dual bound and prune nodes early. For recent research on alternative criteria, see, e.g., [30, 22, 15, 20].

A very popular branching rule called *pseudo-cost branching* [9] uses history information about the change of the dual bound caused by previous branchings. More accurate, but also more

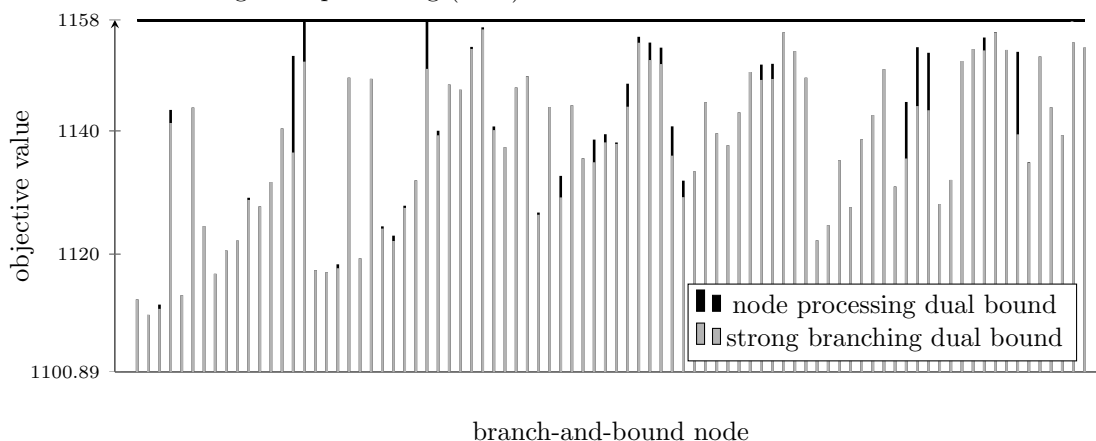
*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, gamrath@zib.de

expensive, is *strong branching* [19, 7, 25], which explicitly computes dual bounds of potential child nodes by solving auxiliary LPs with the branching bound change temporarily added. The *full strong branching rule* does this at every node for each integer variable with fractional LP value which empirically leads to very small branch-and-bound trees [1, 5]. Modern branching rules typically combine these two approaches and use strong branching in the case of uninitialized or unreliable pseudo-cost values (see [5, 3]). Often, they also impose an iteration limit to restrict the effort for solving the strong branching LPs, see, e.g., [19]. Although the result is then not necessarily a valid dual bound for the potential child node anymore, it can still serve as a prediction for it. An important example for a modern branching rule is *reliability pseudo-cost branching* [5], which considers the history information for a variable unreliable if the number of pseudo-cost updates for this variable is below a given threshold. What this all amounts to is that strong branching is an important component of state-of-the-art branching rules, predicting the bounds of potential children when no other information is available yet.

In practice, however, one can often observe a difference between the dual bound that strong branching computes for a node and the actual dual bound obtained later when this particular node is processed. An example is shown in Figure 1, which depicts the differences for instance *aflow30a* from MIPLIB 2003. The y-axis ranges from the root node dual bound up to the optimum objective value which was provided at the beginning. The instance was solved with full strong branching after 87 nodes, each of which is represented by one bar. All strong branching LPs were solved to optimality, but still for almost half of the nodes, the bound obtained during node processing is better than the predicted bound, on average, 10% of the gap between predicted and primal bound is closed by the node LP. This restrains the effectiveness of strong branching, since with more accurate predictions, better branching variables could have been selected. There are various reasons for the difference, most prominently *domain propagation* and global domain changes found in the meantime. The task of domain propagation (or *node preprocessing*) is to tighten the local domains of variables by inspecting the constraints and current domains of other variables at the local subproblem. It is the essential part of each constraint programming solver [8] and has also proven to improve MIP solvers significantly by tightening the LP relaxation, resulting in better dual bounds and detecting infeasibilities earlier [1, 32, 17].

While strong branching cannot do anything about the difference in the dual bounds caused by

Figure 1: Comparison of the dual bounds computed during strong branching (light) and those obtained later during node processing (dark) for instance *aflow30a* from MIPLIB 2003.



global domain changes, it should react upon the continuous improvement in domain propagation techniques. In this paper, we examine how strong branching can be improved by combining it with domain propagation in order to compute better dual bound predictions. This means that we perform the same domain propagation steps that are already performed at each node of the branch-and-bound tree also during strong branching, prior to solving the strong branching LP of a potential child node. The general idea and an evaluation of the direct effects on single strong branching calls are presented in the next section. We implemented the modified strong strong branching method within the MIP solver itself instead of using external methods of the LP solver. This allows us to apply additional tricks to further improve the strong branching performance which are discussed in Section 3. After that, in Section 4, we provide detailed computational results on a collection of MIPLIB [11, 6, 23] instances. In a setting which focuses on the branching performance of a full strong branching rule, we achieve significant reductions of both number of nodes and solving time, but also the default algorithm, using a hybrid branching method combining strong branching and pseudo-costs, can be improved.

2 Strong branching with domain propagation

In the following, we regard mixed-integer linear programs of the form:

$$\min\{c^T x \mid Ax \geq b, x \geq 0, x_i \in \mathbb{Z} \forall i \in I\}. \quad (1)$$

A basic implementation of strong branching works as follows: Given the current problem P of form (1) and an integer variable x_i , $i \in I$, with fractional LP solution value \hat{x}_i , it computes dual bounds of the two potential child nodes that would be created by branching on x_i . Therefore, it creates two temporary subproblems P_d (the *down child*) and P_u (the *up child*) by adding to P the bound changes $x_i \leq \lfloor \hat{x}_i \rfloor$ and $x_i \geq \lceil \hat{x}_i \rceil$, respectively, and solves their LP relaxations. Strong branching with domain propagation (*SBDP*) improves this by applying domain propagation to tighten the variable domains of P_d and P_u after adding the bound changes. The strong branching dual bound for each of the children is then either set to $+\infty$, if propagation detected infeasibility, or obtained by solving the (tightened) LP relaxation of the respective child node. Computing these two dual bound predictions for a given variable with fractional value will be referred to as a strong branching *call* in the following.

Since the additional domain propagation step can only tighten the LP relaxation, the dual bounds obtained by SBDP are always greater than or equal to the ones computed by standard strong branching as long as we solve the LPs to optimality. This is the case for the full strong branching rule we focus on in this section, later, in Section 4, we will also regard a hybrid branching rule which imposes an iteration limit for the strong branching LPs. The questions we want to consider in the remainder of this paper are the following:

- a) How does the strong branching effort change?
- b) By how much can SBDP improve the strong branching predictions?
- c) What is the effect of SBDP on the overall performance of a MIP solver?

Question a) addresses the time spent for domain propagation as well as the change in the number of LP iterations (and therewith the LP solving time). The simplex warmstart normally allows to solve the strong branching LPs with just a few iterations as there is only one bound tightened, but additional changes performed by domain propagation might change this. Question b) addresses both the detection of infeasibilities as well as the computed dual bounds of potential child nodes. While answering the first two questions will give us an indication concerning the profitability of

Table 1: Impact of domain propagation on the strong branching calls.

category	calls	strong branching			strong branching with propagation				
		inf	iters	time	domchgs	inf	cl. gap	iters	time
better bound	376.0	–	44.0	19.5	38.9	–	20.7%	57.2	23.1
same bound	23802.0	–	82.3	39.7	23.7	–	–	79.0	40.6
cutoff	3342.6	0.92	56.8	27.3	35.7	1.11	8.5%	46.7	25.6
all	30469.4	0.14	81.0	40.2	26.3	0.17	2.7%	77.5	40.5

SBDP, the final question c) should assess it by taking into account the changes of tree size and solving time caused by SBDP.

For answering these questions, we performed computational experiments using an implementation of SBDP based on the MIP solver SCIP 3.0 [1, 2] with underlying LP solver Soplex 1.7 [35]. They were performed on a cluster of Intel Xeon E5420 2.5 GHz computers, with 6 MB cache and 16 GB RAM, running Linux (in 64 bit mode). A time limit of two hours per instance was imposed. We use full strong branching to measure the impact of our changes for each candidate variable at each node and concentrate on the branch-and-bound performance by providing the optimal objective value as objective cutoff and disabling primal heuristics and cutting plane separation as well as the components presolver¹ of SCIP. As test set, we used the MMM test set consisting of all instances from MIPLIB 3.0 [11], MIPLIB 2003 [6], and the benchmark set of MIPLIB 2010 [23]. We excluded all instances for which no significant amount of strong branching was performed—either because the instance was solved in presolving or at the root node prior to branching or because the time limit of two hours was hit before strong branching was performed on at least ten variables. Additionally, we excluded the three infeasible instances from MIPLIB 2010 in order to be able to compute the additional gap closed by SBDP, which left us with a total number of 147 instances. With respect to the other measures, however, the results on the infeasible instances were consistent to those presented in the following.

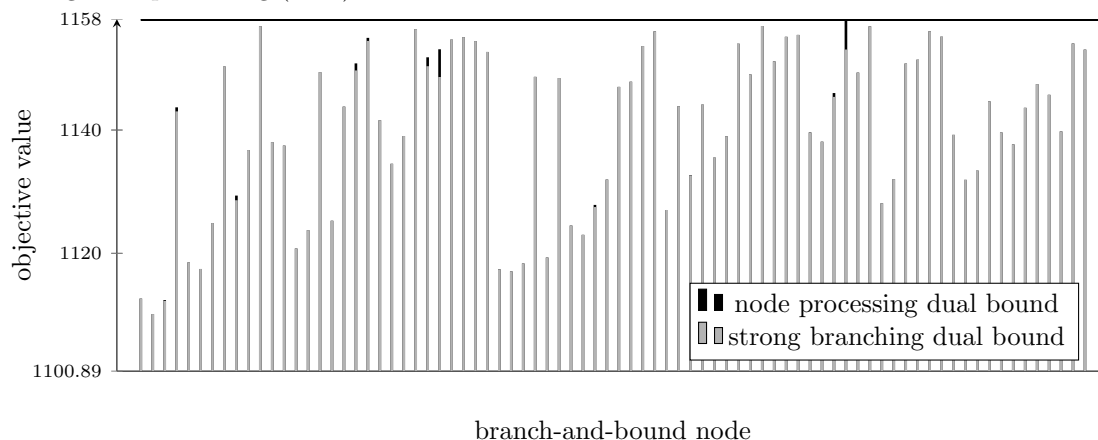
The experiments were conducted as follows: After each standard strong branching call, we additionally performed a call of SBDP on the same variable, running the default domain propagation techniques of SCIP (see [1]). We collected statistics about the differences, but did not use any of the information produced by SBDP within the branch-and-bound search. We chose this approach instead of running twice, one time with each variant, to isolate the impact of the new method on each single strong branching call and avoid noise introduced by differences in the branch-and-bound tree created by different branching choices.

For analyzing the impact of SBDP, we divide the strong branching calls into three categories: *Cutoff* if at least one of the two potential child nodes was detected to be infeasible by any of the methods, *better bound* if no infeasibility was detected and SBDP computed a better dual bound for at least one of the potential child nodes, and *same bound* if both strong branching variants computed the same (finite) bounds for both potential child nodes.

The results for each of these categories are presented in one line in Table 1, with an additional line that summarizes these results for all strong branching calls. Besides the number of strong branching calls (column *calls*), we show for both strong branching variants the number of potential subproblems detected to be infeasible (column *inf*), the number of LP iterations for solving the LPs of the two subproblems (column *iters*), and the strong branching time in milliseconds (column *time*). Furthermore, we present the number of additional domain changes performed by SBDP (column *domchgs*) and the percentage of the gap between primal bound and strong branching dual bound closed by using SBDP instead of standard strong branching (column *cl. gap*). Except

¹The components presolver solves small independent subproblems in advance, excluding them from the main branch-and-bound search.

Figure 2: Comparison of the dual bounds computed by SBDP (light) and those obtained later during node processing (dark) for instance aflow30a from MIPLIB 2003.



for the entries in column `cl. gap`, each of the numbers listed is computed by taking the arithmetic mean over all strong branching calls for a single instance and average over the instances by taking the *shifted geometric mean*.² We use a shift of 100 for the number of strong branching calls, 10 for time, iteration number and domain changes, and 1 for the number of child nodes declared infeasible per call. For the closed gap, having only values between 0 and 100, the average over the instances is determined by arithmetic mean.

As expected, the *better bound* case—which happens only rarely—is typically caused by a high number of domain changes during propagation and leads to an increase in both the average sum of LP iterations and time per strong branching call, thereby closing the gap by more than 20% on average. In the most common case, the *same bound* category, a smaller, but still relevant number of domains are changed by propagation. But instead of slowing down the simplex warm start, these bound changes even reduce the average number of LP iterations, e.g., by fixing variables that would otherwise need to be rendered feasible by some simplex pivots. Last, in the *cutoff* case, SBDP detects infeasibility of more potential child nodes—on average 1.11 of the two children regarded per call are declared infeasible compared to 0.92 otherwise. In about 15% of the cases, infeasibility is detected already during propagation, leading to a reduction of the average number of LP iterations and strong branching time. On average over all strong branching calls, SBDP can declare every twelfth instead of nearly every fourteenth strong branching child node infeasible and closes the gap by 2.66%. The average number of LP iterations is slightly decreased, while the time per strong branching call increases marginally. This demonstrates that the domain propagation time is relatively small compared to the total strong branching time; on average, it was less than 5%.

We also repeated our motivating example from Section 1 using SBDP, the results are illustrated in Figure 2. The differences between the predicted bounds and the bounds obtained when processing the nodes are much smaller: the prediction was inaccurate for only 17% rather than 47% of the nodes, the average gap closed by the node LP reduces from 10% to 2.6%. This also leads to solving the instance after 81 instead of 87 branch-and-bound nodes.

Summing up, the increase in the strong branching effort is negligible—which answers question a)—while also question b) can be answered positively: The strong branching predictions

²See Achterberg [1, Appendix A3] for a definition and discussion of the shifted geometric mean.

are improved both by detecting infeasibilities more often as well as computing more accurate LP bounds. For the `aflow30a` instance which served as our motivating example, this helps to reduce the difference between predicted and LP bound during node processing by 74%. In a branch-and-bound search, this should lead to more variable fixings and help us taking better branching decisions. This assumption will be examined further in Section 4, based on computational results showing the impact of SBDP on the performance of SCIP, thereby also answering the remaining question c).

3 Additional strong branching improvements

Our enhanced strong branching method uses the domain propagation methods of the MIP solver which also exploit integrality information. Thus, we implemented it within SCIP itself and do not use the highly optimized strong branching interface methods provided by many LP solvers such as CPLEX or XPRESS. On the one hand, this might slightly deteriorate the performance of the LP solver for the strong branching LPs; on the other hand, it allows us to optimize the strong branching method with respect to the MIP solver and exploit integrality information not only for domain propagation. In this section, we present three such improvements. Some of them may be considered common sense, but at least they have not been implemented in the state-of-the-art non-commercial MIP solver SCIP so far. The first two improvements were already discussed in the proceedings paper [18] extended by this article while the third one was newly added.

The first improvement treats the case of an *infeasible strong branching subproblem*, which traditionally leads to simply tightening the domain of the candidate variable at the current node (or cutting off the current node if both subproblems are infeasible). While, e.g., the strong branching interface method of CPLEX always regard both subproblems,³ we interrupt a strong branching call when the first potential child is found infeasible, saving the effort we would spend for the second child node. As usual, the domain change of the other subproblem is then applied at the current node either directly or after a certain number of domain changes has been collected this way. This causes a reoptimization of the node LP, after which branching is started again, if needed. Any information we would get by solving the second strong branching child, e.g., an improved bound for the current node or a proof for its infeasibility, we get when reoptimizing the node LP, anyway. A simple trick helps to exploit this even further: In our computational experiments presented in Section 2, about 69% of the infeasible subproblems were up children. This is not surprising since problems are often modeled in a way such that changing a variable’s lower bound—in particular, fixing a binary variable to one—has more impact than changing its upper bound (fixing a binary variable to zero). Thus, we investigate the potential *up child first* in order to profit from infeasible child nodes more often.

Secondly, we can often identify *valid local bounds* for some variables even if neither of the two potential child nodes is infeasible. If any variable’s domain in the two potential child nodes was tightened to $[lb_d, ub_d]$ and $[lb_u, ub_u]$, respectively, we can change the domain of the variable in the local problem to $[\min\{lb_d, lb_u\}, \max\{ub_d, ub_u\}]$. This means that we essentially perform *probing preprocessing* [32] locally as a side product of SBDP with negligible additional cost. For 94 of the 147 instances regarded in Section 2, this technique was able to identify tighter bounds, identifying on average 3.15 bounds that could have been tightened per strong branching call with both subproblems feasible.

Finally, we exploit the fact that we get *LP solutions during strong branching* instead of just

³Also with other LP solvers that do not provide a strong branching interface method, in particular SOPLEX, which we use in our computational experiments, the LP interface implemented in SCIP behaves the same way.

the LP objective value as would be the case when using the external strong branching methods of CPLEX or XPRESS.⁴ Because the changed bound of the investigated variable typically forces this variable to an integer value in the strong branching LP solution, we are optimistic that this solution is “more integral” than the node LP solution. Therefore, we check this solution for integrality, hoping to find a new feasible solution. Moreover, we even run the fast *simple rounding heuristic* [4] implemented in SCIP on the optimal strong branching LP solution. In the next section, we will answer the question of how often solutions are found this way and how good they are.

4 Computational results

In this section, we present computational experiments illustrating the effect of SBDP on the overall performance of SCIP and will finally answer question c) posed in Section 2. The results significantly extend those presented in [18] and use new features implemented in the meantime, e.g., the third improvement discussed in Section 3, so that we used a new development version of SCIP 3.0.1 [1, 2] (git hash 9e5a0ff) with underlying LP solver Soplex 1.7.1 [35] (git hash 82b4e30). The experiments were performed on a cluster of Intel Xeon E5672 3.2 GHz computers, with 12 MB cache and 48 GB RAM, running Linux (in 64 bit mode). As test set, we used the MMM set as in Section 2, this time without excluding any instances, and imposed a time limit of two hours per instance. We use a shifted geometric mean with shift 10 and 100 to average over the solving times and node numbers, respectively. Detailed instance-wise results of all computational experiments presented in this section are provided in Table 4 in the Appendix.

The setting of our first experiment is similar to that used in Section 2: We provide the optimum as cutoff bound and disable primal heuristics, cutting plane separation, and the components presolver in order to focus on the branch-and-bound search and to reduce random performance changes (see [23]). Moreover, we use full strong branching as a branching rule which completely relies on strong branching and uses no history information. Within the full strong branching rule, we exchanged the strong branching calls for SBDP using all improvements discussed in Section 3. Note that the last improvement has no impact in this setting because we provide the optimum right at the beginning and disable heuristics.

The results are summarized in Table 2. They are promising: With SBDP, SCIP is able to solve 107 out of the 168 instances of the MMM test set within the time limit of two hours, seven instances more than with standard strong branching. The shifted geometric mean of the solving time is reduced by 14%. For the subset of instances that both versions solved to optimality (row MMM: solved by both), the average number of nodes and the solution time are reduced by 25% and 13%, respectively. Since we are testing a branching rule, we are mainly interested in instances of this subset for which of least one of the variants built a branch-and-bound tree and did not solve the instance at the root node already. The results for these instances (row MMM: solved with tree), are slightly better: The number of branch-and-bound nodes and the solving time are

⁴Also with other LP solvers like Soplex, SCIP did not use the strong branching LP solution before.

Table 2: Comparison of full strong branching with and without SBDP.

test set	size	full strong branching			full strong with SBDP		
		solved	nodes	time	solved	nodes	time
MMM: complete	168	100	1836.9	534.0	107	1373.8	459.7
MMM: solved by both	100	100	753.1	83.8	100	568.3	73.1
MMM: solved with tree	85	85	1143.2	113.4	85	832.8	97.6

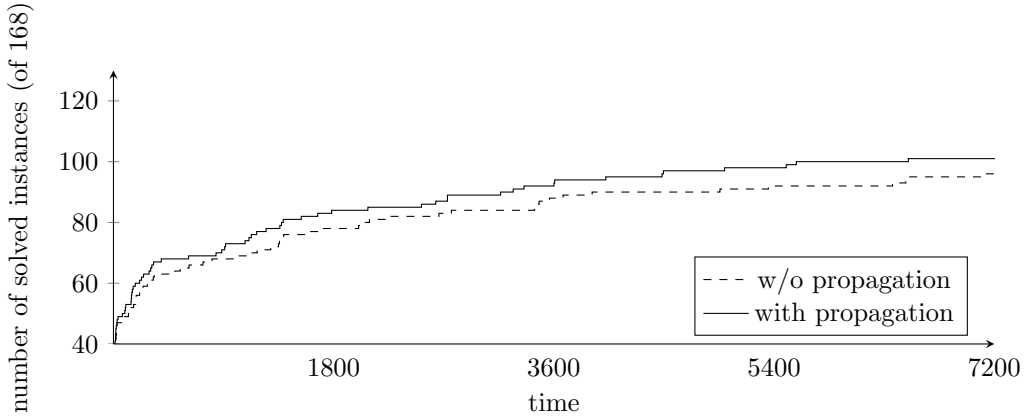


Figure 3: Performance diagram for full strong branching on the MMM test set. The graph indicates the number of instances solved within a certain time.

reduced by 27% and 15% in the shifted geometric mean, respectively. The running times are also illustrated by the performance diagram shown in Figure 3. We see that in this experiment, full strong branching with domain propagation clearly dominated the version without propagation: For every possible time limit between 10 seconds and two hours, it solved more instances to optimality within this time. Finally, let us mention that we also did an experiment with SBDP without the additional improvements to single out their effects. It turned out that they only have a small impact in this setting, causing only a change of one percent in running time and number of nodes; the main improvement comes from applying domain propagation.

Our previous experiments give an answer to question c) posed in Section 2: They indicate that SBDP is quite beneficial in an environment focusing on the strong branching performance. But does this also hold for a state-of-the-art branching rule? In order to answer this question, we take the next step and include SBDP into *hybrid branching*, the default branching rule of SCIP. Hybrid branching is an extension of reliability pseudo-cost branching which takes into account not only the predicted dual bounds, but also other history information, e.g., the number of implied reductions of other variable domains and the number of (recent) conflicts a variable appears in. For more detail on hybrid branching, we refer to [3].

The main criterion for selecting the branching variable, however, is still the predicted child node dual bound, which in the beginning of the search is computed by strong branching. Thus, we expect SBDP to have a positive impact also on hybrid branching. Moreover, also some of the other regarded history information is automatically initialized when running SBDP on a variable. On the other hand, hybrid branching uses a reliability mechanism, i.e., strong branching will be performed only a limited number of times on each variable; later on, only history information is used. This is why we expect the effect of SBDP to be smaller in this setting.

Another important difference to our previous experiments is that hybrid branching imposes a limit on the number of iterations per strong branching LP. Therefore, tightening the LP relaxation might not pay off if this results in hitting the iteration limit where the untightened LP would be solved to optimality. Thus, we did an experiment similar to that presented in Section 2: Running hybrid branching on each instance of the MMM test set with a time limit of two hours, we additionally performed SBDP after each normal strong branching call and compared how often the LP could be solved to optimality (or infeasibility was detected) by the two variants. It turns out that for most of the instances, the iteration limit is reached more

Table 3: Comparison of hybrid branching with and without SBDP.

test set	size	hybrid branching			hybrid branching with SBDP		
		solved	nodes	time	solved	nodes	time
MMM: complete	168	127	8084.0	303.9	128	7538.5	295.2
MMM: solved by both	126	126	3284.9	101.5	126	3083.9	99.6
MMM: solved with tree	114	114	4798.8	126.7	114	4478.3	124.2

often by SBDP, only for four instances SBDP performs slightly better than standard strong branching. On average over all the instance where branching is performed, standard strong branching hits the iteration limit for 4.9% of the regarded children while SBDP hits it for 9.8% of them. Although the majority of strong branching LPs is still solved to optimality, this clearly restricts the possible improvement by SBDP.

The overall effects of SBDP on the performance of the hybrid branching rule are the subject of our final computational experiment. This time, we use the default settings of SCIP, i.e., we do not disable cutting plane separation, heuristics, or the components presolver, neither do we provide the optimum objective value as the cutoff bound. Since we are using heuristics now, let us first evaluate the impact of running the simple rounding heuristic on the strong branching LP solutions: For 19 instances of the MMM test set, we can improve the incumbent this way at least once during search; in total, 53 solutions are found. Moreover, the first primal solution is found this way for eight instances and five times, even the optimum is obtained during strong branching. Table 3 illustrates the same type of information as Table 2 now for the experiments running hybrid branching. The performance is slightly improved: The shifted geometric mean of the solving time over all instances is reduced by 2.9% and one more instance is solved within the time limit when using SBDP. Also the reduction of the number of nodes is smaller compared to the full strong branching rule, but still, the tree size is reduced by 6.1% on average for those instances solved to optimality and slightly more for those not already solved at the root node. Because the one instance that is only solved with SBDP is excluded, the performance improvement reduces to about 2% for both subsets. Taking into account the less important role of strong branching within hybrid branching, these results are consistent to our previous results for full strong branching.

Finally, we can conclusively answer question c) posed in Section 2: Our experiments indicate that SBDP also improves the performance of the default hybrid branching rule used within the state-of-the-art non-commercial MIP solver SCIP.

5 Conclusions and outlook

This paper examined improvements of strong branching, one of the main components of most state-of-the-art branching rules for mixed-integer linear programming. The primary improvement is the incorporation of domain propagation into the strong branching method in order to compute more accurate dual bound predictions. Our computational experiments on general MIP instances show that this comes with negligible cost and reduces the branch-and-bound tree size as well as the solving time. In addition, we presented three further enhancements which improve the strong branching performance.

The effect is most distinctive in a full strong branching rule, where our method might prove particularly useful when the branch-and-bound tree should be kept small, e.g., under tight memory restrictions or for massive parallel MIP solvers (see, e.g., [33, 34]), where reducing the tree size has the added advantage of reducing the message passing overhead. But also in a state-of-the-art branching rule like hybrid branching, the average tree size was reduced and the average

solving time was slightly decreased. Therefore, the new method will be included in the next SCIP release and used by default.

For “structured” or more general problem classes like MINLP or CIP [1] for which typically the LP misses more information that can be exploited by domain propagation, we expect an even larger improvement by SBDP. A field for future research is the combination with other recent strong branching improvements such as cloud branching [10] or nonchimerical branching [16].

6 Acknowledgements

The author would like to thank Tobias Achterberg and Michael Winkler for fruitful discussions, Ambros Gleixner and four anonymous reviewers for helpful suggestions on an earlier version of the paper, and Timo Berthold for constructive comments on this paper.

References

- [1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.
- [2] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
- [3] T. Achterberg and T. Berthold. Hybrid branching. In W. J. van Hoes and J. N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, volume 5547 of *Lecture Notes in Computer Science*, pages 309–311. Springer, May 2009.
- [4] T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. In D. Klatte, H.-J. Lthi, and K. Schmedders, editors, *Operations Research Proceedings 2011*, Operations Research Proceedings, pages 71–76. Springer Berlin Heidelberg, 2012.
- [5] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.
- [6] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006.
- [7] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. On the solution of traveling salesman problems. *Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung*, pages 645–656, 1998.
- [8] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
- [9] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.
- [10] T. Berthold and D. Salvagnin. Cloud branching. In C. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 28–43. Springer Berlin Heidelberg, 2013.

- [11] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, (58):12–15, June 1998.
- [12] R. Borndörfer, C. E. Ferreira, and A. Martin. Decomposing matrices into blocks. *SIAM J. Optim.*, 9(1):236 – 269, 1998.
- [13] G. Cornuéjols, L. Liberti, and G. Nannicini. Improved strategies for branching on general disjunctions. *Mathematical Programming*, 130:225–247, 2011.
- [14] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.
- [15] M. Fischetti and M. Monaci. Backdoor Branching. In O. Günlück and G. J. Woeginger, editors, *Integer Programming and Combinatorial Optimization*, volume 6655 of *Lecture Notes in Computer Science*, pages 183–191. Springer Berlin / Heidelberg, 2011.
- [16] M. Fischetti and M. Monaci. Branching on nonchimerical fractionalities. *OR Letters*, 40(3):159–164, 2012.
- [17] A. Fügenschuh and A. Martin. Computational integer programming and cutting planes. In K. Aardal, G. L. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 69–122. Elsevier, 2005.
- [18] G. Gamrath. Improving strong branching by propagation. In C. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 347–354. Springer Berlin Heidelberg, 2013.
- [19] J.-M. Gauthier and G. Ribière. Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming*, 12(1):26–47, 1977.
- [20] A. Gilpin and T. Sandholm. Information-theoretic approaches to branching in search. *Discrete Optimization*, 8(2):147–159, 2011.
- [21] M. Karamanov and G. Cornuéjols. Branching on general disjunctions. *Mathematical Programming*, 128:403–436, 2011.
- [22] F. Kilinç Karzan, G. L. Nemhauser, and M. W. Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1:249–293, 2009.
- [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. Mittelman, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.
- [24] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.
- [25] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies in mixed-integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.
- [26] A. Lodi, T. Ralphs, F. Rossi, and S. Smriglio. Interdiction branching. Technical Report OR/09/10, DEIS, Università di Bologna, 2009.

- [27] A. Mahajan and T. K. Ralphs. Experiments with branching using general disjunctions. In J. W. Chinneck, B. Kristjansson, and M. J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, volume 47 of *Operations Research/Computer Science Interfaces Series*, pages 101–118. Springer US, 2009.
- [28] G. Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155–170, 1973.
- [29] J. H. Owen and S. Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. *Computational Optimization and Applications*, 20:159–170, 2001.
- [30] J. Patel and J. Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming*, 110:445–474, 2007.
- [31] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North Holland, Amsterdam, 1981.
- [32] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
- [33] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch. ParaSCIP – a parallel extension of SCIP. In C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, editors, *Competence in High Performance Computing 2010*, pages 135 – 148, 2012.
- [34] Y. Shinano, T. Berthold, S. Heinz, T. Koch, M. Winkler, and T. Achterberg. ParaSCIP – a parallel extension of SCIP. Technical Report ZR 11-10, Zuse Institute Berlin, 2011.
- [35] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.

A Detailed Computational Results

Table 4 lists detailed results for the computational experiments described in Section 4. For each of the instances of the MMM test set, we present the results for full strong branching and hybrid branching both with and without propagation. We list the number of processed branch-and-bound nodes and the solving time, if an instance hit the time limit, we write the gap, followed by “%”. According to [23], the gap is defined as $\text{gap} = \frac{pb-db}{\inf\{|z|, z \in [db, pb]\}}$, where pb and db are primal bound and dual bound, respectively. If the gap is infinite, we print “inf%”. Since the test set contains only minization problems, $pb \geq db$ holds for all instances. If the result with and without strong branching differ by more than five percent, we print the dominating value in boldface. Large numbers are abbreviated, “k” and “M” stand for a missing factor of 1 000 and 1 000 000, respectively. At the end of the table, we present the number of solved instances as well as the shifted geometric mean of the number of nodes and the solving time. Additionally, we show these means over two subsets of the instances: The instances that were solved to optimality with and without domain propagation and the part of the former subset where at least one of the two variants did not solve the problem in the root node already. These subsets are defined individually for full strong branching and hybrid branching, so that for each of the two branching rules, the impact of SBDP can be evaluated separately.

Table 4. Detailed computational results for the comparison strong branching with and without propagation, both for full strong branching and hybrid branching, as described and summarized in Section 4.

instance	full strong branching				hybrid branching			
	w/o propagation		with propagation		w/o propagation		with propagation	
	nodes	time	nodes	time	nodes	time	nodes	time
10teams	2	7.0	2	6.4	151	12.6	439	15.2
30n20b8	178	100.0%	29	1365.0	141	383.0	466	605.0
a1c1s1	19.8k	190.9%	15.4k	183.0%	442.1k	16.8%	490.3k	17.8%
acc-tight5	1	0.0	1	0.0	619	124.8	3.2k	399.4
aflow30a	4.1k	70.5	2.4k	50.6	2.1k	10.3	1.4k	10.8
aflow40b	52.0k	5.7%	49.5k	7028.9	211.1k	1255.4	114.0k	758.2
air03	1	0.7	2	0.8	1	1.6	1	1.4
air04	9	39.6	9	36.3	122	61.3	111	68.7
air05	9	121.1	9	114.8	212	36.9	286	39.3
app1-2	39	2478.8	13	1678.0	39	1169.7	112	1627.1
arki001	108.1k	0.0%	135.4k	0.0%	383.4k	0.0%	456.2k	0.0%
ash608gpia-3col	1	inf%	1	inf%	9	48.1	7	34.3
atlanta-ip	4	9.5%	3	10.3%	10.1k	5.6%	7.4k	13.6%
beasleyC3	27.7k	178.2%	19.4k	129.0%	1.0M	16.0%	1.2M	15.2%
bell3a	12.3k	3.3	14.4k	3.0	24.0k	5.2	23.6k	5.2
bell5	930.2k	273.1	18.4k	9.8	1.2k	0.7	1.1k	0.7
bab5	216	7.5%	276	7.8%	21.5k	1.3%	28.3k	1.4%
biella1	9	402.4	11	419.2	5.1k	1141.4	4.4k	1594.7
bienst2	21.3k	851.4	20.9k	767.5	92.3k	294.7	91.7k	288.7
binkar10.1	426.7k	0.4%	310.9k	0.4%	137.7k	187.6	217.9k	311.2
blend2	107	0.3	103	0.4	199	0.8	183	0.7
bley_xl1	17	3563.0	2	686.9	1	245.4	1	244.9
bnatt350	1	0.0	1	0.0	6.1k	534.4	4.6k	415.7
cap6000	616	0.9	627	0.9	3.4k	2.5	2.7k	2.3
core2536-691	1	9.8	1	9.7	97	177.1	206	414.8
cov1075	6.0k	12.3%	5.7k	12.4%	1.6M	7.5%	1.6M	7.3%

cont'd next page

instance	full strong branching				hybrid branching			
	w/o propagation nodes	with propagation time	w/o propagation nodes	with propagation time	w/o propagation nodes	with propagation time	w/o propagation nodes	with propagation time
csched010	27.2k	10.2%	25.5k	10.4%	749.4k	5025.5	785.2k	5296.3
dano3mip	31	19.3%	30	19.3%	2.1k	20.1%	2.2k	20.1%
danoimt	32.0k	3.3%	35.6k	3.2%	865.2k	4026.2	954.2k	4462.0
dcmulti	290	0.9	294	1.2	146	1.8	258	2.5
dfn-gwin-UUM	71.2k	674.5	67.6k	834.8	68.8k	152.1	48.1k	99.6
disctom	1	0.0	1	0.0	1	3.9	1	4.0
ds	1	63.2%	1	63.2%	446	387.4%	509	433.5%
dsbmip	1	0.4	1	0.4	15	1.4	14	1.4
egout	35	0.0	17	0.0	1	0.0	1	0.0
eil33-2	314	166.7	318	75.1	8.9k	62.0	751	57.3
eilB101	163	1006.0	161	680.0	17.2k	394.5	9.9k	343.9
enigma	1	0.0	1	0.0	831	0.6	1.1k	0.8
enlight13	942.8k	42.0%	5.2k	72.8	17.8M	43.8%	1.2M	475.5
enlight14	914.1k	inf%	8.5k	152.2	17.7M	inf%	5.1M	2348.2
ex9	1	0.3	1	0.3	1	31.4	1	31.1
fast0507	26	1837.4	26	2247.7	890	291.2	743	222.4
fiber	1.2k	15.2	1.1k	18.6	21	1.5	19	1.3
fixnet6	81	0.9	79	0.8	12	2.3	14	2.1
flugpl	340	0.1	61	0.0	424	0.1	113	0.1
gen	22	0.1	27	0.2	1	0.1	1	0.1
gesa2-o	8.1k	139.1	5.4k	145.6	8	1.3	6	1.2
gesa2	7.7k	91.0	4.8k	89.3	6	1.3	4	1.2
gesa3	58	1.6	36	1.6	8	1.7	12	1.6
gesa3_o	56	2.5	36	2.4	11	1.6	13	1.7
glass4	95.8k	643.1	623	14.4	5.2M	2336.2	11.5M	5254.4
gmu-35-40	1.3M	4355.7	1.2M	5430.5	11.8M	0.0%	11.1M	0.0%
gt2	2	0.0	2	0.0	1	0.1	1	0.1
harp2	160.3k	695.2	95.4k	563.3	15.9M	5685.9	13.9M	4753.8
iis-100-0-cov	3.0k	22.8%	3.3k	26.1%	111.5k	1975.3	109.4k	1917.4
iis-bupa-cov	741	20.6%	789	21.9%	192.4k	6.2%	185.3k	7.1%
iis-pima-cov	663	12.0%	645	11.8%	13.8k	1009.6	12.2k	1021.8
khb05250	421	2.1	413	2.7	6	0.6	6	0.6
lectsched-4-obj	1	0.1	1	0.2	145.5k	1178.5	15.1k	301.6
liu	18.4k	102.1%	12.8k	102.1%	1.5M	157.1%	1.8M	132.9%
l152lav	17	1.2	17	1.4	26	2.0	23	2.2
lseu	772	0.7	479	0.7	183	0.4	249	0.5
m100n500k4r1	1	0.0	1	0.0	8.3M	4.2%	7.7M	4.2%
macrophage	3.4k	353.3%	3.1k	356.1%	938.7k	31.3%	898.8k	31.3%
manna81	4.4k	1.0%	3.5k	1.0%	1	0.8	1	0.7
map18	149	3239.0	141	3248.6	367	417.5	327	431.9
map20	151	2676.8	141	2749.7	545	352.6	380	335.8
markshare1	13.9M	inf%	8.7M	inf%	81.4M	inf%	85.9M	inf%
markshare2	10.6M	inf%	6.6M	inf%	69.0M	inf%	68.0M	inf%
mas74	544.8k	1002.1	523.0k	1317.3	3.5M	634.9	3.3M	582.8
mas76	75.6k	91.7	72.3k	123.0	468.7k	68.0	424.5k	60.9
mcsched	498	7.1%	484	7.0%	12.3k	148.9	12.8k	180.7

cont'd next page

instance	full strong branching				hybrid branching			
	w/o propagation		with propagation		w/o propagation		with propagation	
	nodes	time	nodes	time	nodes	time	nodes	time
mik-250-1-100-1	1.8M	3.3%	1.2M	4.1%	1.7M	421.4	1.7M	440.4
mine-166-5	193	12.0	81	11.4	2.1k	26.2	3.1k	26.2
mine-90-10	59.8k	1381.0	15.2k	623.8	44.2k	222.7	476.2k	2273.8
misc03	103	1.2	103	1.1	308	1.4	122	1.1
misc06	12	0.3	12	0.4	9	0.8	9	0.7
misc07	2.0k	45.1	2.1k	46.4	17.5k	12.2	22.8k	15.2
mitre	1	5.0	1	5.1	1	5.2	1	5.3
mkc	292.0k	1.4%	154.8k	1.4%	2.5M	0.6%	1.4M	2.1%
mod008	363	0.5	347	0.6	272	1.0	268	0.9
mod010	5	0.5	5	0.6	3	0.9	2	0.8
mod011	2.9k	587.2	2.7k	537.9	919	154.1	738	121.9
modglob	1.1M	3917.9	1.1M	5284.7	436	1.2	561	1.4
momentum1	351	15.9%	333	18.6%	28.9k	25.0%	23.8k	62.6%
momentum2	196	14.0%	96	3627.3	87.1k	5451.4	77.4k	6228.2
momentum3	1	150.8%	1	150.8%	156	258.1%	154	258.1%
msc98-ip	53	0.7%	32	0.7%	3.1k	35.7%	4.6k	35.3%
mspp16	29	3184.2	27	2790.7	75	4260.2	47	2878.3
mzzv11	41	5265.9	35	4952.4	3.8k	250.5	2.8k	232.9
mzzv42z	5	563.9	3	458.0	1.4k	181.8	1.9k	225.6
n3div36	2.7k	13.6%	2.2k	13.8%	269.1k	9.9%	248.5k	11.0%
n3seq24	5	0.4%	4	0.4%	1.2k	7.3%	3.8k	0.4%
n4-3	49.4k	41.5%	38.9k	43.3%	58.3k	907.3	59.9k	956.8
neos-1109824	7.3k	1125.2	6.9k	1259.0	25.2k	197.1	26.3k	210.5
neos-1337307	10.4k	0.5%	2.4k	0.4%	319.4k	0.0%	407.7k	0.0%
neos-1396125	2.8k	3238.4	1.3k	1582.3	69.8k	2067.3	98.0k	2969.9
neos13	6	496.3	6	585.4	23.2k	33.3%	23.3k	19.5%
neos-1601936	1	309.0	1	307.1	14.9k	33.3%	32.0k	66.7%
neos18	11.4k	1727.9	2.0k	553.9	6.3k	32.0	5.0k	29.6
neos-476283	22	92.9	22	106.9	469	244.2	585	277.2
neos-686190	266	152.1	258	156.7	5.3k	67.3	5.8k	78.8
neos-849702	1	0.0	1	0.0	141.7k	2479.5	21.3k	411.3
neos-916792	83.6k	11.7%	61.4k	11.9%	117.6k	461.5	84.5k	341.9
neos-934278	2	0.2%	2	0.2%	6.0k	2.5%	3.3k	2.9%
net12	53	102.6%	46	154.7%	6.1k	3798.0	5.8k	4009.8
netdiversion	1	4.9%	1	4.9%	111	2096.7k%	125	2100.8k%
newdano	192.6k	12.4%	198.7k	11.3%	2.0M	3745.4	2.0M	3497.9
noswot	372.5k	790.6	89.6k	454.5	625.6k	127.9	760.8k	156.9
ns1208400	1	250.5	1	164.0	2.6k	704.6	5.4k	455.3
ns1688347	164	111.7	79	48.6	6.6k	940.4	4.0k	496.9
ns1758913	1	2.2%	1	2.2%	17	514.3%	49	514.3%
ns1766074	248.3k	365.8	218.5k	891.8	911.2k	788.3	936.9k	672.8
ns1830653	1.8k	3951.9	1.3k	2497.7	56.6k	787.1	56.6k	637.5
nsrand-ipx	38.7k	1.5%	27.3k	1.5%	1.7M	2.8%	1.7M	2.1%
nw04	12	9.9	12	9.8	5	31.2	5	26.9
opm2-z7-s2	58	1046.3	56	1039.8	6.5k	1378.0	4.8k	927.9
opt1217	1.5M	18.8%	940.8k	18.8%	1	0.6	1	0.7

cont'd next page

instance	full strong branching				hybrid branching			
	w/o propagation		with propagation		w/o propagation		with propagation	
	nodes	time	nodes	time	nodes	time	nodes	time
p0033	32	0.0	7	0.0	1	0.0	1	0.0
p0201	61	1.4	59	1.7	63	1.5	45	1.6
p0282	14	0.1	10	0.1	3	0.5	3	0.5
p0548	143	0.4	18	0.2	5	0.3	5	0.3
p2756	592	5.2	436	5.9	79	1.2	42	1.3
pg5_34	57.2k	12.2%	53.7k	12.3%	299.5k	1283.6	210.6k	1075.0
pigeon-10	1.9M	11.1%	1.1M	11.1%	15.6M	11.1%	15.9M	11.1%
pk1	55.8k	162.8	51.6k	195.5	342.7k	71.2	359.0k	74.2
pp08a	2.0M	24.5%	1.3M	27.4%	721	1.7	541	1.6
pp08aCUTS	249.3k	1721.3	238.1k	2012.1	489	1.6	600	1.9
protfold	12	34.2%	14	34.2%	14.7k	inf%	3.7k	52.1%
pw-myciel4	2.7k	150.0%	2.4k	100.0%	856.0k	6134.7	1.1M	6194.7
qiu	15.7k	1739.8	16.2k	1803.3	12.1k	74.8	11.6k	77.2
qnet1	6	1.1	8	1.0	46	6.1	38	4.8
qnet1.o	25	0.9	23	1.1	17	3.6	10	3.2
rail507	26	1956.2	24	2241.9	882	266.4	849	233.3
ran16x16	1.1M	5.6%	909.8k	5.2%	392.8k	332.2	380.7k	317.2
reblock67	19.4k	695.7	7.9k	442.6	265.0k	557.2	71.1k	186.1
rd-rplusc-21	130	165.3k%	202	165.3k%	62.4k	181.3k%	47.5k	167.0k%
rentacar	2	1.2	2	1.4	2	2.3	2	2.4
rgn	252	0.7	248	0.8	2	0.3	2	0.3
rmatr100-p10	93	678.6	87	694.1	907	143.2	870	151.2
rmatr100-p5	33	1330.1	33	1364.9	381	293.3	421	307.6
rmine6	37.9k	1781.7	35.1k	1800.5	581.9k	2061.3	2.2M	0.0%
rocII-4-11	3.3k	16.7%	445	1633.8	25.8k	311.3	15.1k	275.3
rococoC10-001000	9.5k	7.7%	14.2k	6.3%	1.0M	6072.3	1.1M	7082.2
roll3000	10.4k	8.7%	9.3k	9.0%	2.0M	5766.3	1.6M	4591.8
rout	2.8k	107.0	1.6k	72.0	27.5k	35.7	12.7k	22.0
satellites1-25	2	300.0%	54	4674.7	17.8k	2283.6	2.9k	707.1
set1ch	181.5k	33.8%	131.9k	32.2%	19	0.9	18	1.0
seymour	200	3.9%	217	4.2%	147.6k	2.1%	108.1k	2.4%
sp97ar	5.0k	1.2%	5.1k	1.2%	12.7k	6.6%	5.8k	7.1%
sp98ic	1.5k	0.6%	1.3k	0.5%	188.7k	2.5%	184.6k	2.5%
sp98ir	110	72.2	98	77.5	6.4k	93.2	9.9k	142.8
stein27	881	1.6	879	2.2	4.5k	1.2	3.9k	1.1
stein45	7.8k	61.3	7.8k	74.0	52.3k	13.9	49.1k	13.6
stp3d	1	2.5%	1	2.5%	1	inf%	1	inf%
swath	47.6k	34.5%	39.8k	33.9%	1.2M	18.0%	1.1M	16.1%
t1717	7	25.3%	8	25.3%	2.9k	39.2%	2.5k	43.5%
tanglegram1	2	77.5%	2	77.5%	29	691.0	29	711.5
tanglegram2	1	1.9	1	1.7	3	6.8	3	7.1
timtab1	750.7k	38.2%	477.8k	34.9%	1.2M	489.1	922.7k	393.8
timtab2	163.4k	122.5%	122.1k	111.9%	10.1M	46.4%	9.8M	49.8%
tr12-30	45.7k	332.6%	31.9k	333.2%	1.2M	1625.2	1.1M	1505.3
triptim1	1	82.4	1	80.7	16	1303.8	39	1647.6
unitcal.7	361	0.7%	211	0.4%	28.0k	1703.8	36.3k	2194.8

cont'd next page

instance	full strong branching				hybrid branching			
	w/o propagation		with propagation		w/o propagation		with propagation	
	nodes	time	nodes	time	nodes	time	nodes	time
vpm1	11.8k	13.0	4.2k	8.6	1	0.0	1	0.0
vpm2	7.6k	15.2	4.6k	15.0	256	1.0	90	0.9
vpphard	9	inf%	12	inf%	19.0k	inf%	16.3k	inf%
zib54-UUE	31.9k	39.4%	38.4k	31.1%	175.8k	1789.5	194.3k	1762.5
solved (of 168)		100		107		127		128
sh. geom. mean	1836.9	534.0	1373.8	459.7	8084.0	303.9	7538.5	295.2
solved by both		100		100		126		126
sh. geom. mean	753.1	83.8	568.3	73.1	3284.9	101.5	3083.9	99.6
solved with tree		85		85		114		114
sh. geom. mean	1143.2	113.4	832.8	97.6	4798.8	126.7	4478.3	124.2