Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

GERALD GAMRATH

Improving strong branching by propagation

This paper is to appear in the Proceedings of the 10th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR 2013) held May 18-22, 2013, in Yorktown Heights, NY, USA. The final publication is available at http://www.springerlink.com/.

ZIB-Report 12-46 (December 2012)

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 propagation *

Gerald Gamrath†

Abstract

Strong branching is an important component of most variable selection rules in branch-and-bound based mixed-integer linear programming solvers. It predicts the dual bounds of potential child nodes by solving auxiliary LPs and thereby helps to keep the branch-and-bound tree small. In this paper, we describe how these dual bound predictions can be improved by including *domain propagation* into strong branching. Computational experiments on standard MIP instances indicate that this is beneficial in three aspects: It helps to reduce the average number of LP iterations per strong branching call, the number of branch-and-bound nodes, and the overall solving time.

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

Mathematics Subject Classification: 90C10, 90C11, 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 [1, 2], branching rules have been an important field of research in that context, being one of the core parts of the method (for surveys, see [3, 4, 5]). Their task is to split the current node's problem into two or more disjoint subproblems if the solution to the current LP relaxation does not fulfill the integrality restrictions, thereby excluding 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 [6, 7, 8, 9, 10]) or can create more than two subproblems, cf. [11, 12]. 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 child nodes. This helps to tighten the global dual bound and prune nodes early (for recent research on alternative criteria, see, e.g., [13, 14, 15, 16]). A very popular branching rule called *pseudo-cost branching* [17] uses history information about the change of the dual bound caused by previous branchings. More accurate, but also more expensive, is *strong branching* [18, 19, 4], which explicitly computes dual bounds of potential child nodes by solving an auxiliary LP 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 [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, 20]).

*This article is to appear in the Proceedings of the 10th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR 2013) held May 18-22, 2013, in Yorktown Heights, NY, USA.

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

In practice, one can often observe a difference between the dual bound that strong branching computes for a node and the actual dual bound obtained later during node processing. This restrains the effectiveness of strong branching, which should predict the actual dual bound of the node and not just compute some valid dual bound. 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 integral part of each constraint programming solver [21] and has also proven to improve MIP solvers significantly by tightening the LP relaxation, resulting in better dual bounds and detecting infeasibilities earlier [22, 23, 24].

While strong branching cannot do anything about the difference in the dual bounds caused by 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 are presented in the next section. Based on that, we discuss additional improvements in Section 3 and provide benchmark results on a collection of MIPLIB [25, 26, 27] instances showing a reduction of both number of nodes and solving time when propagation is applied within a full strong branching rule.

2 Strong branching with domain propagation

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

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

The basic implementation of strong branching with domain propagation (*SBDP*) 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. After that, the variable domains of P_d are tightened by domain propagation. If propagation detects infeasibility, a dual bound of $+\infty$ is returned for P_d, otherwise the LP relaxation of P_d is solved and its optimal value provides the strong branching dual bound. The dual bound of P_u is computed analoguously.

The only difference to "standard" strong branching is that domain propagation is performed before solving the LP. Since this tightens the LP relaxation, the dual bounds obtained by solving the strong branching LP are always greater than or equal to the ones computed by standard strong branching.[1] The questions to be considered in this paper are: Is this worth the additional effort? In particular, how big is the propagation time and how does the number of LP iterations change? The simplex warmstart normally allows to solve the strong branching LPs with just a few iterations as there is only one bound changed, but additional changes performed by domain propagation might change this.

[1] In this paper, we assume that the strong branching LPs are solved to optimality and no iteration limit is applied. This is also the case for the implementation of the full strong branching rule used in our computational experiments.

For answering these questions, we performed computational experiments using an implementation of SBDP based on the MIP solver SCIP 3.0 [22, 28] with underlying LP solver SoPlex 1.7 [29]. They were performed on 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 presolver[2] of SCIP. As test set, we used the MMM test set consisting of all instances from MIPLIB 3 [25], MIPLIB 2003 [26], and the benchmark set of MIPLIB 2010 [27]. We excluded all instances for which no significant amout of strong branching was performed (less than ten strong branching calls on single variables)—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. 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.

The experiments were then conducted as follows: After each standard strong branching call, we additionally performed a call of SBDP on the same variable, running the same domain propagation techniques as SCIP does on any node of the branch and bound tree (cf. [22]). 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 exclude the difference in the branch-and-bound tree created by different branching methods and isolate the impact of the new method on each single strong branching call.

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, *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 infeasible (column cutoffs), the number of LP iterations for solving the LPs of the two subproblems (column LP iters), and the strong branching time in milliseconds (column time). Furthermore, we present the number of domain changes performed by SBDP (column dom. chgs.) and the percentage of the gap between primal bound and strong branching dual bound closed by using SBDP instead of standard strong branching (column gap closed). For each of the numbers listed, we compute the arithmetic mean over all strong branching calls for the single instances and average over the instances by taking a *shifted geometric mean*[3]. 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. Only for the gap closed, having only values between 0 and 100, do we average over the instances 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 number 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

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

[3]For a definition and discussion of the shifted geometric mean, see Achterberg [22, Appendix A3].

Table 1: Impact of SBDP on the strong branching calls.

		standard strong branching			strong branching with domain propagation				
category	calls	cutoffs	LP iters	time	dom. chgs.	cutoffs	gap closed	LP iters	time
better bound	376.02	–	44.00	19.5	38.85	–	20.73%	57.19	23.1
same bound	23801.96	–	82.33	39.7	23.74	–	–	78.99	40.6
cutoff	3342.63	0.92	56.81	27.3	35.70	1.11	8.50%	46.74	25.6
all	30469.42	0.14	81.03	40.2	26.26	0.17	2.66%	77.52	40.5

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%.

To summarize, SBDP exhibits benefits in all three categories. In the majority of strong branchings, where it yields no bound improvement, it reduces the number of LP iterations. In the remaining cases, significantly more child nodes can be cut off and about 20% additional gap is closed.

3 Further improvements and computational results

In this section we describe further improvements motivated by the results of our first computational experiments and present the effect of SBDP on the overall performance when it is used within the full strong branching rule.

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 normally, strong branching methods always regard both subproblems, 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, causing a reoptimization of its LP, after which branching is started again, if needed.

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). In order to profit from infeasible child nodes more often, we decided to investigate the potential up child first.

As in probing preprocessing (see [23]), 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 $[\min\{lb_d, lb_u\}, \max\{ub_d, ub_u\}]$. 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

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

		full strong branching			full strong with SBDP		
test set	size	solved	nodes	time	solved	nodes	time
MMM: complete	168	97	814	633.1	100	645	582.2
MMM: all optimal	94	94	321	86.3	94	253	78.9

both subproblems feasible. With this improvement, probing preprocessing is performed as a side product of SBDP.

Using these improvements, we performed computational experiments to compare the performance of SBDP against standard strong branching. We used the same computing environment as described in Section 2 and also the MMM testset described there, this time without excluding any instances. Within SCIP, we exchanged the strong branching calls in the full strong branching rule for SBDP and again provided the optimum as cutoff bound, disabled 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 [27]).

The results are summarized in Table 2. We regard both the complete MMM test set (row MMM: complete) as well as the subset of instances that both variants solved to optimality (row MMM: all optimal), and present—besides the size of the sets—aggregated results for both sets. More specifically, we list the number of solved instances and the shifted geometric mean (with a shift of 10) of the number of processed branch-and-bound nodes and the solving time. The results are promising: with the improved strong branching method, SCIP is able to solve 100 out of the 168 instances of the MMM test set within the time limit of two hours, three instances more than with standard strong branching. For the subset of instances that both versions solved to optimality, the average number of nodes and the solution time are reduced by 21% and 9%, respectively. Detailed instance-wise results of the computational experiments are provided in Table 3 in the Appendix.

4 Conclusions and outlook

In this paper, we improved strong branching by applying domain propagation to compute more accurate dual bound predictions. First computational experiments on general MIP instances show that this comes with relatively small cost and, used in a full strong branching rule, can speed up the solution process while reducing the branch-and-bound tree size. For "structured" or more general problems classes like MINLP or CIP [22] where typically the LP misses more information which can be exploited by domain propagation, we expect an even larger improvement by the new method.

Our preliminary results show the potential of the approach. An integration into state-of-the-art branching rules like reliability branching [5] and a possible combination with other recent strong branching improvements like cloud branching [31] or nonchimerical branching [32] are fields for future research.

Already the improved full strong branching might prove 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.

5 Acknowledgements

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

References

[1] Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3) (1960) 497–520

[2] Dakin, R.J.: A tree-search algorithm for mixed integer programming problems. The Computer Journal **8**(3) (1965) 250–255

[3] Mitra, G.: Investigation of some branch and bound strategies for the solution of mixed integer linear programs. Mathematical Programming **4** (1973) 155–170

[4] Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of search strategies in mixed-integer programming. INFORMS Journal on Computing **11**(2) (1999) 173–187

[5] Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33** (2005) 42–54

[6] Ryan, D.M., A.Foster, B.: An integer programming approach to scheduling. In Wren, A., ed.: Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling. North Holland, Amsterdam (1981) 269–280

[7] Owen, J.H., Mehrotra, S.: Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. Computational Optimization and Applications **20** (2001) 159–170

[8] Mahajan, A., Ralphs, T.K.: Experiments with branching using general disjunctions. In Chinneck, J.W., Kristjansson, B., Saltzman, M.J., eds.: Operations Research and Cyber-Infrastructure. Volume 47 of Operations Research/Computer Science Interfaces Series. Springer US (2009) 101–118

[9] Karamanov, M., Cornuéjols, G.: Branching on general disjunctions. Mathematical Programming **128** (2011) 403–436

[10] Cornuéjols, G., Liberti, L., Nannicini, G.: Improved strategies for branching on general disjunctions. Mathematical Programming **130** (2011) 225–247

[11] Borndörfer, R., Ferreira, C.E., Martin, A.: Decomposing matrices into blocks. SIAM J. Optim. **9**(1) (1998) 236 – 269

[12] Lodi, A., Ralphs, T., Rossi, F., Smriglio, S.: Interdiction branching. Technical Report OR/09/10, DEIS, Università di Bologna (2009)

[13] Patel, J., Chinneck, J.: Active-constraint variable ordering for faster feasibility of mixed integer linear programs. Mathematical Programming **110** (2007) 445–474

[14] Kılınç Karzan, F., Nemhauser, G.L., Savelsbergh, M.W.: Information-based branching schemes for binary linear mixed integer problems. Mathematical Programming Computation **1** (2009) 249–293

[15] Fischetti, M., Monaci, M.: Backdoor Branching. In Günlück, O., Woeginger, G.J., eds.: Integer Programming and Combinatorial Optimization. Volume 6655 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2011) 183–191

[16] Gilpin, A., Sandholm, T.: Information-theoretic approaches to branching in search. Discrete Optimization **8**(2) (2011) 147–159

[17] Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G., Vincent, O.: Experiments in mixed-integer linear programming. Mathematical Programming **1** (1971) 76–94

[18] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: On the solution of traveling salesman problems. Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung (1998) 645–656

[19] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics). Princeton University Press, Princeton, NJ, USA (2007)

[20] Achterberg, T., Berthold, T.: Hybrid branching. In van Hoeve, W.J., Hooker, J.N., eds.: 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., Springer (May 2009) 309–311

[21] Apt, K.R.: Principles of Constraint Programming. Cambridge University Press (2003)

[22] Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007)

[23] Savelsbergh, M.W.P.: Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing **6** (1994) 445–454

[24] Fügenschuh, A., Martin, A.: Computational integer programming and cutting planes. In Aardal, K., Nemhauser, G.L., Weismantel, R., eds.: Discrete Optimization. Volume 12 of Handbooks in Operations Research and Management Science. Elsevier (2005) 69–122

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

[26] Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters **34**(4) (2006) 1–12

[27] 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. Mathematical Programming Computation **3**(2) (2011) 103–163

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

[29] Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. PhD thesis, Technische Universität Berlin (1996)

[30] Gamrath, G.: Improving strong branching by propagation. Technical Report 12-46, ZIB, Takustr. 7, 14195 Berlin (2012)

[31] Berthold, T., Salvagnin, D.: Cloud branching. Technical Report 13-01, ZIB, Takustr. 7, 14195 Berlin (2013)

[32] Fischetti, M., Monaci, M.: Branching on nonchimerical fractionalities. OR Letters **40**(3) (2012) 159–164

[33] Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T.: ParaSCIP – a parallel extension of SCIP. In Bischof, C., Hegering, H.G., Nagel, W.E., Wittum, G., eds.: Competence in High Performance Computing 2010. (2012) 135 – 148

[34] Shinano, Y., Berthold, T., Heinz, S., Koch, T., Winkler, M., Achterberg, T.: ParaSCIP – a parallel extension of SCIP. Technical Report ZR 11-10, Zuse Institute Berlin (2011)

A Detailed Computational Results

Table 3 lists detailed results for the computational experiments described in Section 3. For each if the instances from MIPLIB 3, MIPLIB 2003, and MIPLIB 2010, we list the solving time (where 7200 means that the time limit was reached), the number of processed branch-and-bound nodes, and the optimality gap at termination. According to [27], 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. Since the test set contains only minization problems, $pb \geq db$ holds for all instances. Additionally, we present the number of solved instances as well as the shifted geometric mean of the number of nodes and the solving time both for all instances and for only the subset of instances that both variants solved to optimality.

	full strong branching			full strong with SBDP		
instance	nodes	gap	time	nodes	gap	time
10teams	2	0.0	10.2	2	0.0	8.9
30n20b8	55	100.0	7200.0	117	100.0	7200.0
a1c1s1	13.2k	193.9	7200.0	14.0k	171.4	7200.0
acc-tight5	1	—	0.1	1	—	0.1
aflow30a	4.2k	0.0	107.9	2.4k	0.0	83.9
aflow40b	33.8k	6.4	7200.0	24.6k	6.3	7200.0
air03	1	0.0	12.9	1	0.4	13.0
air04	11	0.0	61.4	7	0.0	54.3
air05	9	0.0	188.7	9	0.0	171.2
app1-2	35	0.0	4210.1	13	0.0	1554.4
arki001	190.8k	0.0	7200.0	163.1k	0.0	7200.0
ash608gpia-3col	1	—	7200.0	1	—	7200.0
atlanta-ip	2	10.1	7200.0	2	10.1	7200.0
beasleyC3	18.5k	180.1	7200.0	12.9k	136.5	7200.0
bell3a	14.3k	0.0	3.8	14.3k	0.0	4.3
bell5	921.0k	0.0	423.2	239.8k	0.0	150.0
bab5	194	6.1	7200.0	238	6.6	7200.0
biella1	11	0.0	690.1	14	0.0	635.5
bienst2	21.1k	0.0	1264.2	21.1k	0.0	1152.3
binkar10_1	281.8k	0.5	7200.0	221.3k	0.5	7200.0
blend2	111	0.0	0.2	136	0.0	0.3
bley_xl1	12	13.6	7200.0	11	0.0	7175.8
bnatt350	1	—	0.2	1	—	0.2
cap6000	609	0.0	1.0	624	0.0	1.2
core2536-691	1	0.0	18.1	1	0.0	18.0
cov1075	3.9k	12.7	7200.0	4.4k	13.2	7200.0
csched010	14.7k	11.5	7200.0	15.5k	12.4	7200.0
dano3mip	20	19.3	7200.0	20	19.3	7200.0
danoint	21.3k	3.6	7200.0	24.3k	3.6	7200.0
dcmulti	296	0.0	1.0	288	0.0	1.2
dfn-gwin-UUM	71.2k	0.0	1002.0	66.4k	0.0	1292.7
disctom	1	—	0.1	1	—	0.1
ds	1	63.3	7200.0	1	63.2	7200.0
dsbmip	1	0.0	0.2	1	0.0	0.2
egout	27	0.0	0.0	15	0.0	0.0

cont'd next page

instance	full strong branching			full strong with SBDP		
	nodes	gap	time	nodes	gap	time
eil33-2	242	0.0	225.2	250	0.0	133.8
eilB101	164	0.0	1483.5	160	0.0	1007.5
enigma	1	—	0.0	1	—	0.0
enlight13	656.6k	82.1	7200.0	6.0k	0.0	123.0
enlight14	656.3k	—	7200.0	24.9k	—	1031.7
ex9	1	—	1.5	1	—	1.4
fast0507	13	1.0	7200.0	12	1.0	7200.0
fiber	1.2k	0.0	22.6	1.0k	0.0	26.8
fixnet6	81	0.0	0.6	79	0.0	0.8
flugpl	341	0.0	0.1	74	0.0	0.0
gen	5	0.0	0.1	5	0.0	0.1
gesa2-o	8.1k	0.0	210.0	6.8k	0.0	241.5
gesa2	7.7k	0.0	131.9	4.9k	0.0	137.7
gesa3	58	0.0	2.1	36	0.0	2.2
gesa3_o	56	0.0	3.2	42	0.0	3.7
glass4	165.2k	0.0	2484.9	5.1k	0.0	182.4
gmu-35-40	1.3M	0.0	6548.8	1.1M	0.0	7200.0
gt2	2	0.0	0.0	2	0.0	0.0
harp2	71.7k	0.0	486.1	42.6k	0.0	373.0
iis-100-0-cov	1.9k	24.5	7200.0	2.0k	26.1	7200.0
iis-bupa-cov	484	24.0	7200.0	603	21.7	7200.0
iis-pima-cov	437	13.8	7200.0	451	12.9	7200.0
khb05250	421	0.0	2.6	403	0.0	3.4
lectsched-4-obj	1	—	0.2	1	—	0.2
liu	9.2k	102.1	7200.0	7.7k	102.1	7200.0
l152lav	17	0.0	1.6	17	0.0	1.7
lseu	699	0.0	0.4	466	0.0	0.5
m100n500k4r1	1	0.0	0.0	1	0.0	0.0
macrophage	2.0k	356.1	7200.0	1.4k	475.4	7200.0
manna81	2.9k	1.0	7200.0	1.7k	1.0	7200.0
map18	153	0.0	5713.0	147	0.0	5716.7
map20	143	0.0	4708.4	153	0.0	4791.0
markshare1	9.5M	—	7200.0	6.5M	—	7200.0
markshare2	7.1M	—	7200.0	4.8M	—	7200.0
mas74	544.9k	0.0	1545.4	523.2k	0.0	1888.2
mas76	75.6k	0.0	139.8	72.4k	0.0	174.8
mcsched	246	7.1	7200.0	282	7.2	7200.0
mik-250-1-100-1	1.3M	3.5	7200.0	946.9k	4.2	7200.0
mine-166-5	153	0.0	26.4	135	0.0	25.0
mine-90-10	52.6k	0.0	1664.8	16.9k	0.0	901.9
misc03	103	0.0	1.0	103	0.0	1.1
misc06	12	0.0	0.5	16	0.0	0.5
misc07	2.0k	0.0	67.8	2.1k	0.0	68.7
mitre	1	0.0	8.3	1	0.0	8.2
mkc	191.3k	1.4	7200.0	113.8k	1.4	7200.0
mod008	504	0.0	0.5	490	0.0	0.5

cont'd next page

instance	full strong branching			full strong with SBDP		
	nodes	gap	time	nodes	gap	time
mod010	6	0.0	0.4	5	0.0	0.3
mod011	3.0k	0.0	905.1	2.7k	0.0	878.3
modglob	1.1M	0.0	5748.5	1.0M	0.2	7200.0
momentum1	247	24.4	7200.0	228	18.6	7200.0
momentum2	95	13.8	7200.0	118	14.3	7200.0
momentum3	1	151.0	7200.0	1	151.0	7200.0
msc98-ip	33	0.7	7200.0	8	0.9	7200.0
mspp16	29	5.4	7200.0	27	0.0	6714.8
mzzv11	43	0.8	7200.0	36	1.0	7200.0
mzzv42z	3	0.0	4626.2	3	0.0	4531.6
n3div36	1.5k	13.6	7200.0	1.4k	13.8	7200.0
n3seq24	3	0.4	7200.0	3	0.4	7200.0
n4-3	42.6k	32.0	7200.0	32.4k	36.4	7200.0
neos-1109824	7.4k	0.0	1887.9	7.0k	0.0	2628.8
neos-1337307	6.6k	0.6	7200.0	1.2k	0.6	7200.0
neos-1396125	2.8k	0.0	4457.7	1.4k	0.0	2272.1
neos13	6	0.0	762.8	6	0.0	950.9
neos-1601936	1	0.0	523.5	1	0.0	528.1
neos18	10.6k	0.0	2183.2	1.9k	0.0	846.8
neos-476283	19	0.0	189.3	18	0.0	215.0
neos-686190	262	0.0	226.3	250	0.0	228.4
neos-849702	1	—	0.1	1	—	0.1
neos-916792	49.6k	12.2	7200.0	28.7k	12.7	7200.0
neos-934278	1	0.0	27.9	1	0.0	27.8
net12	33	154.7	7200.0	17	154.7	7200.0
netdiversion	1	4.9	7200.0	1	4.9	7200.0
newdano	113.6k	20.3	7200.0	125.4k	23.9	7200.0
noswot	555.5k	0.0	1797.2	105.1k	0.0	811.8
ns1208400	1	0.0	511.4	1	0.0	447.7
ns1688347	64	0.0	46.3	63	0.0	65.9
ns1758913	1	2.2	7200.0	1	0.0	4777.1
ns1766074	248.7k	—	529.1	220.4k	—	790.4
ns1830653	1.8k	0.0	6253.3	1.4k	0.0	4170.4
nsrand-ipx	25.0k	1.6	7200.0	18.1k	1.6	7200.0
nw04	15	0.0	22.5	14	0.1	22.4
opm2-z7-s2	70	0.0	1807.6	60	0.0	1784.1
opt1217	941.4k	18.8	7200.0	522.6k	21.9	7200.0
p0033	35	0.0	0.0	7	0.0	0.0
p0201	19	0.0	0.5	17	0.0	0.6
p0282	14	0.0	0.1	10	0.0	0.1
p0548	302	0.0	1.0	157	0.0	0.8
p2756	16.4k	0.0	279.2	12.6k	0.0	293.4
pg5_34	39.6k	12.3	7200.0	39.8k	12.4	7200.0
pigeon-10	1.2M	11.1	7200.0	599.5k	11.1	7200.0
pk1	55.7k	0.0	262.0	50.4k	0.0	292.1
pp08a	1.3M	26.8	7200.0	1.1M	31.7	7200.0

cont'd next page

instance	full strong branching			full strong with SBDP		
	nodes	gap	time	nodes	gap	time
pp08aCUTS	249.3k	0.0	2588.3	238.1k	0.0	2989.9
protfold	9	34.2	7200.0	11	34.2	7200.0
pw-myciel4	1.8k	150.0	7200.0	1.9k	150.0	7200.0
qiu	15.7k	0.0	2591.4	15.9k	0.0	2696.0
qnet1	8	0.0	1.4	6	0.0	1.6
qnet1_o	25	0.0	0.9	23	0.0	0.9
rail507	18	0.9	7200.0	12	1.0	7200.0
ran16x16	726.2k	6.3	7200.0	585.4k	7.2	7200.0
reblock67	20.9k	0.0	1066.4	9.2k	0.0	702.0
rd-rplusc-21	88	$>$ 10000	7200.0	87	$>$ 10000	7200.0
rentacar	2	0.0	1.6	2	0.0	1.6
rgn	250	0.0	0.4	252	0.0	0.5
rmatr100-p10	93	0.0	960.2	89	0.0	988.2
rmatr100-p5	33	0.0	1906.5	33	0.0	1998.1
rmine6	37.6k	0.0	2825.2	34.9k	0.0	2901.5
rocII-4-11	1.7k	47.2	7200.0	1.4k	0.0	5408.1
rococoC10-001000	29.6k	4.3	7200.0	29.6k	3.7	7200.0
roll3000	6.2k	9.2	7200.0	7.9k	10.6	7200.0
rout	2.8k	0.0	161.2	1.7k	0.0	114.0
satellites1-25	1	300.0	7200.0	1	290.0	7200.0
set1ch	120.7k	34.6	7200.0	270.2k	33.8	7200.0
seymour	125	4.3	7200.0	130	3.9	7200.0
sp97ar	2.4k	1.2	7200.0	1.7k	1086.0	7200.0
sp98ic	3.0k	0.0	6669.7	2.4k	0.5	7200.0
sp98ir	108	0.0	112.5	102	0.0	164.8
stein27	881	0.0	2.0	863	0.0	2.6
stein45	7.8k	0.0	92.5	7.8k	0.0	108.0
stp3d	1	2.5	7200.0	1	2.5	7200.0
swath	34.2k	35.0	7200.0	29.4k	35.8	7200.0
t1717	4	25.3	7200.0	5	25.3	7200.0
tanglegram1	2	77.5	7200.0	2	77.5	7200.0
tanglegram2	1	0.0	2.3	1	0.0	2.3
timtab1	445.4k	43.6	7200.0	331.4k	39.3	7200.0
timtab2	103.1k	130.9	7200.0	113.9k	109.0	7200.0
tr12-30	30.3k	339.0	7200.0	30.0k	349.4	7200.0
triptim1	1	0.0	265.2	1	0.0	263.0
unitcal_7	171	0.6	7200.0	80	0.6	7200.0
vpm1	11.8k	0.0	18.6	3.7k	0.0	11.1
vpm2	7.6k	0.0	22.4	4.6k	0.0	20.6
vpphard	6	—	7200.0	8	—	7200.0
zib54-UUE	19.8k	42.6	7200.0	25.9k	33.7	7200.0
solved			97/168			100/168
sh. geom. mean	814		633.1	645		582.2
solved by both			94/94			94/94
sh. geom. mean	321		86.3	253		78.9

Table 3. Detailed computational results for the comparision of full strong branching with and without domain propagation as described and summarized in Section 3.