Available online at www.sciencedirect.com
SCIENCE DIRECT®
Operations Research Letters 33 (2005) 42–54
Operations Research Letters
www.elsevier.com/locate/dsw

Branching rules revisited

Tobias Achterberg[a,*], Thorsten Koch[a], Alexander Martin[b]

[a] *Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany*
[b] *Technische Universität Darmstadt, Schloßgartenstr. 7, D-64289 Darmstadt, Germany*

Received 4 September 2002; accepted 14 April 2004

Available online 26 June 2004

Abstract

We present a new generalization called *reliability branching* of today's state-of-the-art *strong branching* and *pseudocost branching* strategies for linear programming based branch-and-bound algorithms. After reviewing commonly used branching strategies and performing extensive computational studies we compare different parameter settings and show the superiority of our proposed new strategy.

Keywords: Mixed-integer-programming; Branch-and-bound; Variable selection; Pseudocost-branching; Strong-branching; Reliability-branching

1. Introduction

In this paper we are dealing with *mixed integer programs* (MIPs), which are optimization problems of the following form:

$$c^\star = \min c^{\mathrm{T}}x, \quad Ax \leqslant b, \quad x \in \mathbb{Z}^I \times \mathbb{R}^{N\setminus I}, \tag{1}$$

with $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $I \subseteq N = \{1,\ldots,n\}$.

Among the most successful methods are currently linear programming based branch-and-bound (B&B) algorithms where the underlying linear programs (LPs) are possibly strengthened by cutting planes. Most commercial integer programming solvers, see [6], are based on this method. As we will see below, B&B algorithms leave two choices: how to split a problem (branching) and which (sub)problem to select next. In this paper we focus on the branching step and introduce a new generalization that contains many of the known branching rules as special cases. We show that specific choices of parameters for this new class of branching rules perform in most cases better than the current rules when tested on real-world instances.

In Section 2 we review current branching strategies from the literature and present our new generalization. In Section 3 the results of extensive numerical tests on specific parameter choices are presented.

We use the following notation. X_{MIP} denotes the set of feasible solutions of (1), and we set $c^\star = \infty$ if $X_{\mathrm{MIP}} = \emptyset$. The *linear programming relaxation* of (1) is obtained by removing the integrality constraints

$$\bar{c}_{P_{\mathrm{LP}}} = \min\{c^{\mathrm{T}}x \mid x \in P_{\mathrm{LP}}\}, \tag{2}$$

* Corresponding author.
E-mail addresses: Achterberg@zib.de (T. Achterberg), koch@zib.de (T. Koch), martin@mathematik.tu-darmstadt.de (A. Martin).

0167-6377/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.orl.2004.04.002

where $P_{\mathrm{LP}} = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$. We also denote $\bar{c}_{P_{\mathrm{LP}}} = \infty$ if $P_{\mathrm{LP}} = \emptyset$. Obviously, $\bar{c}_{P_{\mathrm{LP}}} \leqslant c^\star$, since $P_{\mathrm{LP}} \supseteq X_{\mathrm{MIP}}$. A typical LP based B&B algorithm for solving (1) looks as follows:

Algorithm 1 (LP-based branch-and-bound).
Input: A MIP in the form (1).
Output: An optimal solution $x^\star \in X_{\mathrm{MIP}}$ and its value $c^\star = c^{\mathrm{T}}x^\star$ or the conclusion that $X_{\mathrm{MIP}} = \emptyset$, denoted by $c^\star := \infty$.

1. Initialize the problem set $S := \{P_{\mathrm{LP}}\}$ with the LP relaxation of the MIP. Set $c^\star := \infty$.
2. If $S = \emptyset$, exit by returning the value $c^\star$ (with an optimal solution $x^\star$).
3. Choose a problem $Q \in S$ and delete it from S.
4. Solve the linear program $\bar{c}_Q = \min\{c^{\mathrm{T}}x \mid x \in Q\}$ with optimal solution $\bar{x}_Q$, where Q might have been strengthened by cutting planes.
5. If $\bar{c}_Q \geqslant c^\star$, goto 2.
6. If $\bar{x}_Q \in X_{\mathrm{MIP}}$, set $c^\star := \bar{c}_Q$ and $x^\star := \bar{x}_Q$, goto 2.
7. Branching: Split Q into subproblems, add them to S and goto 3.

If it is clear from the context we omit Q from all parameters and write $\bar{c}$, $\bar{x}$, etc. instead of $\bar{c}_Q$, $\bar{x}_Q$, etc.

2. Branching rules

Since branching is in the core of any B&B algorithm, finding good strategies was important to practical MIP solving right from the beginning [3,13]. We refrain from giving details of all existing strategies, but concentrate on the most popular rules used in todays MIP solvers. For a comprehensive study of B&B strategies we refer to [8,10] and the references therein.

The only way to split a problem Q within an LP based B&B algorithm is to branch on linear inequalities in order to keep the property of having an LP relaxation at hand. The easiest and most common inequalities are *trivial inequalities*, i.e., inequalities that split the feasible interval of a singleton variable. To be more precise, if i is some variable with a fractional value $\bar{x}_i$ in the current optimal LP solution, we set $f_i^+ = \lceil \bar{x}_i \rceil - \bar{x}_i$ and $f_i^- = \bar{x}_i - \lfloor \bar{x}_i \rfloor$. We obtain two subproblems, one by adding the trivial inequality $x_i \leqslant \lfloor \bar{x}_i \rfloor$ (called the *left subproblem* or *left son*, denoted by Q_i^-) and one by adding the trivial inequality $x_i \geqslant \lceil \bar{x}_i \rceil$ (called the *right subproblem* or *right son*, denoted by Q_i^+). This rule of branching on trivial inequalities is also called *branching on variables*, because it only requires to change the bounds of variable i. Branching on more complicated inequalities or even splitting the problem into more than two subproblems are rarely incorporated into general MIP solvers, even though it can be effective in special cases, see for instance [4,5,15].

The basic algorithm for variable selection may be stated as follows.

Algorithm 2 (generic variable selection).
Input: Current subproblem Q with an optimal LP solution $\bar{x} \notin X_{\mathrm{MIP}}$.
Output: An index $i \in I$ of a fractional variable $\bar{x}_i \notin \mathbb{Z}$.

1. Let $C = \{i \in I \mid \bar{x}_i \notin \mathbb{Z}\}$ be the set of branching candidates.
2. For all candidates $i \in C$, calculate a score value $s_i \in \mathbb{R}$.
3. Return an index $i \in C$ with $s_i = \max_{j \in C}\{s_j\}$.

In the following we focus on the most common variable selection rules, which are all variants of Algorithm 2. The difference is how the score in Step 2 is computed.

The ultimate goal is to find a fast branching strategy that minimizes the number of B&B nodes that need to be evaluated. Since a global approach is unlikely, one tries to find a branching variable that is at least a good choice for the current branching. The quality of a branching is measured by the change in the objective function of the LP relaxations of the two children Q_i^- and Q_i^+ compared to the relaxation of the parent node Q.

In order to compare branching candidates, for each candidate the two objective function changes $\Delta_i^- := \bar{c}_{Q_i^-} - \bar{c}_Q$ and $\Delta_i^+ := \bar{c}_{Q_i^+} - \bar{c}_Q$ are mapped on a single score value. This is typically done by using a function of the form (cf. [10])

$$\mathrm{score}(q^-, q^+) = (1-\mu)\cdot \min\{q^-, q^+\} + \mu \cdot \max\{q^-, q^+\}. \tag{3}$$

The *score factor* μ is some number between 0 and 1. It is usually an empirically determined constant, which is sometimes adjusted dynamically through the course of the algorithm (we use $\mu = 1/6$). Note that special treatment is necessary, if one of the subproblems Q_i^- or Q_i^+ is infeasible.

In the forthcoming explanations all cases are symmetric for the left and right subproblem. Therefore we will only consider one direction, the other will be analogous.

2.1. Most infeasible branching

This still very common rule chooses the variable with fractional part closest to 0.5, i.e., $s_i = 0.5 - |\bar{x}_i - \lfloor \bar{x}_i \rfloor - 0.5|$. The heuristic reason behind this choice is that this selects a variable where the least tendency can be recognized to which "side" (up or down) the variable should be rounded. As the numerical results in Section 3 indicate, the performance of this rule is in general not better than selecting the variable randomly.

2.2. Pseudocost branching

This is a sophisticated rule in the sense that it keeps a history of the success of the variables on which already has been branched. This rule goes back to [3]. In the meantime various variations of the original rule have been proposed. In the following we present the one used in SIP [12]. For alternatives see [10].

Let ς_i^+ be the objective gain per unit change in variable i at node Q, that is

$$\varsigma_i^+ = \Delta_i^+ / f_i^+. \tag{4}$$

Let σ_i^+ denote the sum of ς_i^+ over all problems Q, where i has been selected as branching variable and Q_i^+ has already been solved and was feasible. Let η_i^+ be the number of these problems. Then the pseudocosts for the upward branching of variable i are

$$\Psi_i^+ = \sigma_i^+ / \eta_i^+. \tag{5}$$

Using $s_i = \text{score}(f_i^- \Psi_i^-, f_i^+ \Psi_i^+)$ in Algorithm 2 yields what is called *pseudocost branching*.

Observe that at the beginning of the algorithm $\sigma_i^+ = \eta_i^+ = 0$ for all $i \in I$. We call the pseudocosts of a variable $i \in I$ *uninitialized for the upward direction*, if $\eta_i^+ = 0$. Uninitialized upward pseudocosts are set to $\Psi_i^+ = \Psi_{\text{avg}}^+$, where Ψ_{avg}^+ is the average of the initialized upward pseudocosts over all variables. This average number is set to 1 in the case that all upward pseudocosts are uninitialized. The pseudocosts of a variable are called *uninitialized* if they are uninitialized in at least one direction.

2.3. Strong branching

The idea of *strong branching*, introduced in CPLEX 7.5 [7] (see also [2]), is to test which of the fractional candidates gives the best progress before actually branching on any of them. This test is done by temporarily introducing a lower bound $\lfloor \bar{x}_i \rfloor$ and subsequently an upper bound $\lceil \bar{x}_i \rceil$ for variable i with fractional LP value $\bar{x}_i$, and solving the linear relaxations.

If we choose as candidate set the full set $C = \{i \in I \mid \bar{x}_i \notin \mathbb{Z}\}$ and if we solve the resulting LPs to optimality, we call the strategy *full strong branching*. In other words, *full strong branching* can be viewed as finding the locally (with respect to the given score function) best variable to branch on. We will see in Section 3 that selecting this locally best variable usually works very well in practice w.r.t. the number of nodes needed to solve the problem instances.

Unfortunately, the computation times per node of *full strong branching* are high. Accordingly, most branching rules presented in literature may be interpreted as an attempt to find a (fast) estimate of what *full strong branching* actually measures.

One possibility to speedup *full strong branching* is to restrict the candidate set in some way, e.g. by considering only a subset $C' \subseteq C$ of the fractional variables. To estimate the changes in the objective function for a specific branching decision, often only a few simplex iterations are performed, because the change of the objective function in the simplex algorithm usually decreases with the iterations. Thus, the parameters of *strong branching* to be specified are the size of the candidate set C', the maximum number γ of dual simplex iterations to be performed on each candidate variable, and a criterion according to which the candidate set is selected.

In SIP, the size of the candidate set is not fixed in advance to a specific value, but the candidates are evaluated with a "look ahead" strategy: if no new best candidate was found for λ successive candidates, the evaluation process is stopped. By evaluating

variables with largest *pseudocost* scores first, only the most promising candidates are evaluated. The iteration limit for strong branching evaluations is set to $\gamma = 2\bar{\gamma}$, where $\bar{\gamma}$ is the average number of simplex iterations per LP needed so far. Note that this number only protects from unexpected long simplex runs, on average the candidate LPs will be solved to optimality.

2.4. Hybrid strong/pseudocost branching

Even with the speedups indicated at the end of Section 2.3, the computational burden of *strong branching* is high, and the higher the speedup, the less precise the decisions are.

On the other hand, the weakness of *pseudocost branching* is that at the very beginning there is no information available, and s_i is almost identical for all variables $i \in C$. Many of the early nodes are located in the upper part of the search tree where the decisions have the largest impact on the structure of the tree and the subproblems therein. With *pseudocost branching*, these decisions are taken with respect to pseudocost values that are not useful yet.

To circumvent these drawbacks the positive aspects of *pseudocost* and *strong branching* are put together in the combination *hybrid strong/pseudocost branching*, where *strong branching* is applied in the upper part of the tree up to a given depth level d. For nodes with depth larger than d, *pseudocost branching* is used. This branching rule is available for example in LINDO [11].

2.5. Pseudocost branching with strong branching initialization

The decisions of *pseudocost* as well as the ones of *hybrid strong/pseudocost branching* in the lower part of the tree are potentially based on uninitialized pseudocost values, leading to an inferior selection of branching variables.

The idea to avoid this risk, which goes back to [10], is to use strong branching for variables with uninitialized pseudocosts and to use the resulting strong branching estimates to initialize the pseudocosts. In contrast to the fixed depth level of *hybrid strong/pseudocost branching*, this rule uses strong branching in a more dynamic way.

2.6. Reliability branching

We generalize the idea of *pseudocost branching with strong branching initialization* by not only using strong branching on variables with uninitialized pseudocost values, but also on variables with *unreliable* pseudocost values. The pseudocosts of a variable i are called *unreliable*, if $\min\{\eta_i^-, \eta_i^+\} < \eta_{\text{rel}}$, with η_{rel} being the "reliability" parameter. We call this new branching rule *reliability branching*.

An outline of the selection of a branching variable with *reliability branching* is given in the following Algorithm 3 that replaces Step 2 of Algorithm 2.

Algorithm 3 (Reliability branching).

2. For all candidates $i \in C$, calculate the score $s_i = \text{score}(f_i^- \Psi_i^-, f_i^+ \Psi_i^+)$ and sort them in non-increasing order of their pseudocost scores. For all candidates $i \in C$ with $\min\{\eta_i^-, \eta_i^+\} < \eta_{\text{rel}}$, do:
   (a) Perform a number of at most γ dual simplex iterations on each subproblem Q_i^- and Q_i^+. Let $\tilde{\Delta}_i^-$ and $\tilde{\Delta}_i^+$ be the resulting gains in the objective value.
   (b) Update the pseudocosts Ψ_i^- and Ψ_i^+ with the gains $\tilde{\Delta}_i^-$ and $\tilde{\Delta}_i^+$.
   (c) Update the score $s_i = \text{score}(\tilde{\Delta}_i^-, \tilde{\Delta}_i^+)$.
   (d) If the maximum score $s^* = \max_{j \in C}\{s_j\}$ has not changed for λ consecutive score updates, go to 3.

2.7. Branching rule classification

Some of the proposed branching rules can be adjusted with parameter settings. All of the strategies using strong branching include the simplex iteration limit γ and the look ahead value λ. The *hybrid strong/pseudocost branching* exhibits an additional depth parameter d, while the *reliability branching* comes along with the reliability parameter η_{rel}.

It is interesting to note that depending on the parameter settings, the branching rules have interrelations as illustrated in Fig. 1.

Hybrid strong/pseudocost branching with $d = 0$ as well as *reliability branching* with $\eta_{\text{rel}} = 0$ coincide with pure *pseudocost branching*. With $\eta_{\text{rel}} = 1$, *reliability branching* is equal to *pseudocost branching*

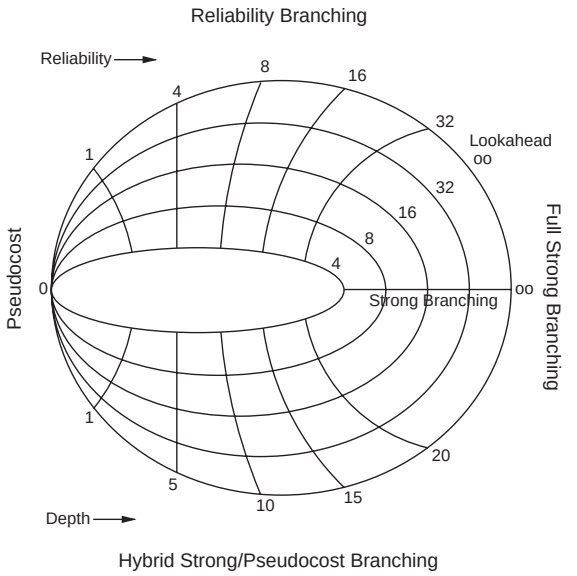

Fig. 1. Interrelations between branching rules and their parameters.

with strong branching initialization. If the depth d and the reliability η_{rel} are increased, the number of strong branching evaluations also increases, and with $d = \eta_{\text{rel}} = \infty$, both strategies converge to pure *strong branching*. Additionally, if the look ahead parameter is set to $\lambda = \infty$, *strong branching* becomes *full strong branching*.

η_{rel}	Classification
0	pseudocost branching
1	pseudocost branching with strong branching initialization
4, 8	best performing branching rules
∞	strong branching

3. Computational results

In this section we compare computational results for different branching rules and parameter settings on several MIP instances. All calculations were performed on a 833 MHz Alpha 21264 workstation with 4 MB Cache and 2 GB RAM.

3.1. Test set

Our test set consists of instances from MIPLIB 2003 [1] and instances used by Mittelmann [14]. We selected all problems where CPLEX [7] 9.0 needs at least 5000 branching nodes and at most one hour CPU time for solving. (CPLEX was run with default settings, except that "absolute mipgap" was set to 10^{-10} and "relative mipgap" to 0.0, which are the corresponding values in SIP.)

In all runs, we used a time limit of 3600 s. Note that the version of SIP used here utilizes CPLEX 9.0 as embedded LP solver. The strong branchings are performed using `CPXstrongbranch()`.

What makes benchmarking branching strategies difficult are the complex interrelations between cutting plane generation, primal heuristics, node selection, and branching variable selection. For example, it is possible that a "worse" branching rule results in less branching nodes and a smaller solution time for a specific instance, because the variable selection leads incidentally to an early discovering of a good or optimal primal solution. However, we dispensed with setting the optimal solution values beforehand, since leading towards feasible solutions fast is a desirable property of branching rules.

Hence, for our comparison of the branching strategies we used the default parameter settings except that cuts are generated in the root node only. For this parameter setting, which is commonly known as cut-and-branch, the influence of the branching strategy is emphasized best.

To verify that the branch-and-bound environment we used with SIP is state-of-the-art, we also ran CPLEX 9.0 on our test set. To better compare the branching decisions we used SIP's preprocessing and cutting plane generation also in CPLEX instead of CPLEX' own routines. In this way both, CPLEX and SIP, operated with pure branch and bound on the same problems. In CPLEX we used the default branching strategy which to the best of our knowledge is some variant of pseudocost branching.

3.2. Description of tables and figures

In Table 4, the summary of all computational results is presented. Columns labeled "total" give the sums of the results over all instances, columns labeled "geom."

give the geometric mean over all instances. The last column lists how many instances where not solved to optimality because the limit of one hour CPU time was reached.

Tables 1 and 2 describe in detail some of our computational results. These tables show for each instance and each branching strategy the number of nodes explored and the time needed to solve all the instances. The number put in parentheses behind the branching rules is the reliability setting for *reliability branching* and the depth setting for *hybrid strong/pseudocost branching* (str/ps).

Numbers in bold face indicate the "winner" for a particular instance between *hybrid strong/pseudocost branching* with $d = 10$, *reliability branching* with $\eta_{\text{rel}} = 1$ which is equal to *pseudocost branching with strong branching initialization*, and *reliability branching* with $\eta_{\text{rel}} = 8$. This is done separately for $\lambda = 4$ and 8.

We did not base our conclusions on performances of single instances and discuss those in detail. We rather rely on average numbers over all instances. It is common sense that the geometric mean is a fair criterion for comparison and we used it as the basis for our conclusions in the next section.

One interesting observation in Table 2 is that *most infeasible branching* is basically as good as *random branching* showing that this rule is of no use. We refrain from considering both rules any further in the following discussion.

Table 3 gives the number of strong branching evaluations performed, i.e., the number of `CPXstrongbranch()` calls, which is the number of times a fractional variable was evaluated with strong branching by solving its two subproblems. The last column shows the average depth of the node tree as encountered while doing *full strong branching*. This gives an indication on how balanced the trees are, the smaller the number the more balanced the trees are. Examples can be seen in Fig. 5 which display the trees of `vpm2` (left) with a low average depth and `neos3` (right) with a very high average depth.

3.3. Results

We have performed a comprehensive computational study of all variants of branching strategies discussed in this paper. For each parameter setting, reflected by an intersection point of two lines in Fig. 1, we ran all instances. The total number as well as the geometric mean in terms of B&B nodes, time and strong branching evaluations is shown in Table 4 for all of these runs.

Fig. 2 illustrates the geometric means of all runs. The circles, squares and stars indicate runs with *reliability branching*, *hybrid strong/pseudocost branching*, and *strong branching*, respectively. The CPLEX run is marked with an '×'. The numbers give, depending on the branching strategy, information about the parameters η_{rel}, d, and λ.

The following conclusions may be drawn from these numbers:

(i) *Strong branching* (stars in Fig. 2) performs remarkably well with respect to the number of evaluated nodes, but, as expected, not with respect to time. Only two instances could not be solved by *full strong branching* within 3600 s. Without time limit, `qiu` is solved in 15 659 nodes and 11 133 s, while `neos7` is solved in 476 601 nodes and 33 418 s.

The only instance where none of the variants of *strong branching* needs the least number of nodes is `neos7`. Especially unexpected is that *full strong branching* needs to evaluate considerably more nodes than *pseudocost branching*. This is due to the fact that *pseudocost branching* is (incidentally) able to find the optimal solution very early (after two minutes and 15 277 nodes). Note that as a consequence *pseudocost branching* is more than twice as fast per node than *most infeasible branching*.

(ii) With respect to time, regardless of the specific parameter setting, *reliability branching* always outperforms *hybrid strong/pseudocost branching*, as can be seen by comparing the circles with the rectangles in Fig. 2.

At least one reason why *hybrid strong/pseudocost branching* performs worse than *reliability branching* can be seen in Fig. 5. What is shown are the branch-and-bound trees generated by *full strong branching* visualized with VBCTOOL [9]. The first tree is from *vpm2* and is reasonably balanced. The second one is from *neos3* and looks like a path. The right branch (fixing a variable to one) is nearly always

Table 1
B&B nodes needed to solve each problem instance

					Lookahead = 4				*Lookahead = 8*				*PLE 9.0*
Example	Random	Most inf.	Pscost	Full str.	Str/ps(10)	Reli(1)	Reli(8)	Strong	Str/ps(10)	Reli(1)	Reli(8)	Strong	(SIP cuts)
aflow30a	> 1 248 221	> 1 025 389	276 002	46 649	**164 112**	203 544	181 397	44 455	210 127	**170 547**	171 909	55 478	105 204
cap6000	7454	6717	6791	3402	4957	5076	**4253**	3452	4957	5076	**4253**	3452	4982
gesa2-o	> 1 023 765	> 1 150 438	126 459	13 743	75 141	57 534	**53 881**	15 831	74 673	79 841	**50 078**	21 988	37 293
mas74	> 4 662 991	> 4 725 208	5 160 828	551 780	**5 163 685**	5 478 830	5 521 061	> 790 014	**4 886 438**	5 296 806	5 425 001	> 645 364	5 617 512
mas76	2 935 689	2 038 945	603 683	114 228	587 813	496 370	**482 122**	125 516	666 398	496 370	**336 826**	188 882	507 536
misc07	47 941	17 955	19 407	2713	49 822	**35 187**	54 932	5228	50 893	**41 902**	55 740	5394	86 825
pk1	1 064 666	850 198	437 758	56 546	**294 469**	367 763	331 339	128 299	**298 771**	367 763	311 611	143 206	366 340
pp08aCUTS	1877	1740	673	124	**355**	673	489	201	**318**	651	464	148	663
qiu	> 163 998	> 165 538	15 471	> 2974	25 180	**16 479**	18 405	6178	23 974	16 148	**14 847**	> 6416	9443
rout	> 714 812	> 702 392	309 779	2 836	45 457	44 232	**13 883**	4179	39 212	39 192	**19 743**	11 305	> 1 467 395
vpm2	92 880	40 258	22 568	1457	11 710	13 780	**9648**	1974	10 826	17 409	**10 054**	1812	4306
ran8 x 32	321 933	674 613	40 069	4902	29 957	26 360	**17 668**	6824	30 526	31 697	**21 092**	4249	45 143
ran10 x 26	> 1 637 058	> 1 500 171	128 327	8520	54 965	64 640	**48 065**	8279	56 448	68 237	**48 626**	11 112	56 548
ran12 x 21	> 1 667 430	> 1 471 340	234 915	13 420	**126 153**	172 161	135 037	12 185	135 165	159 383	**124 455**	18 577	196 478
ran13 x 13	907 443	758 186	149 239	9147	**86 648**	109 241	95 288	16 033	**86 566**	97 195	93 939	20 516	97 325
mas284	162 255	123 569	21 586	3226	**18 700**	24 383	21 179	4564	**17 799**	21 217	20 360	5580	21 472
prod1	> 2 854 213	2 317 393	89 293	10 644	72 890	**63 674**	64 186	14 670	69 992	65 679	**62 689**	15 609	106 369
bc1	33 631	34 127	40 781	2981	42 882	35 132	**25 196**	3666	40 212	35 132	**25 196**	3603	18 463
bienst1	66 099	58 841	19 418	3687	13 594	**10 427**	13 911	5274	13 602	**9248**	9951	5340	12 963
neos2	> 640 712	> 703 794	> 609 109	618	195 730	187 331	**22 742**	4405	244 292	83 146	**30 790**	2685	158 240
neos3	> 403 674	> 393 853	> 446 035	1402	> 505 184	> 737 254	**556 835**	13 215	> 536 938	> 736 396	**626 894**	15 306	606 168
neos7	> 471 073	> 339 479	390 910	> 44 061	> 546 623	535 586	**202 482**	> 55 369	> 598 209	498 573	**252 766**	> 55 799	422 264
swath1	33 021	81 309	19 924	8285	64 320	36 264	**35 161**	11 268	73 512	66 995	**10 615**	16 471	128 017
swath2	83 441	> 240 136	211 976	22 002	71 595	278 413	**28 808**	> 44 510	162 695	258 934	**85 510**	> 36 277	390 480
Total (24)	21 246 277	19 421 589	9 381 001	929 347	8 251 942	9 000 334	7 937 968	1 325 589	8 332 543	8 663 537	7 813 409	1 294 569	10 467 429
Geom. mean	275 789	262 369	88 707	7242	65 966	70 014	48 773	11 640	69 489	69 501	48 377	12 715	79 269

Table 2
Time in seconds needed to solve each problem instance

					Lookahead=4				Lookahead=8				CPLEX 9.0
Example	Random	Most inf.	Pscost	Full str.	Str/ps(10)	Reli(1)	Reli(8)	Strong	Str/ps(10)	Reli(1)	Reli(8)	Strong	(SIP cuts)
aflow30a	>3600.0	>3600.0	582.5	2212.6	**364.6**	403.6	416.4	704.7	457.6	391.7	**368.2**	1522.8	330.6
cap6000	57.1	50.3	52.6	47.4	41.1	38.6	**36.1**	58.9	40.7	37.8	**36.3**	59.5	18.8
gesa2-o	>3600.0	>3600.0	442.7	864.2	275.0	203.2	**199.1**	463.7	272.5	284.3	**195.7**	926.6	93.9
mas74	>2651.9	>2683.5	2734.2	3087.1	**2750.3**	3015.9	2807.4	>3600.0	**2902.2**	3066.4	3104.2	>3600.0	1995.3
mas76	1307.1	916.0	291.7	486.6	274.5	240.6	**225.1**	420.2	315.3	242.2	**161.2**	713.9	136.8
misc07	123.4	54.8	54.6	299.3	141.6	**97.5**	150.2	213.3	143.2	**113.2**	150.2	267.4	225.6
pk1	501.3	483.7	249.9	488.5	**154.3**	199.2	193.3	658.1	**157.2**	200.8	170.3	885.0	161.7
pp08aCUTS	6.8	7.7	3.4	12.4	7.8	**4.5**	5.4	10.3	10.3	**4.4**	5.5	9.9	2.4
qiu	>3600.0	>3600.0	332.3	>3600.0	927.4	**369.5**	419.1	1901.9	1213.9	371.2	**341.9**	>3600.0	152.2
rout	>3600.0	>3600.0	976.9	621.2	150.0	135.7	**58.4**	254.7	165.2	122.2	**81.3**	785.8	>3600.0
vpm2	124.5	62.6	30.6	43.5	18.2	18.6	**15.7**	25.4	19.1	25.0	**16.3**	30.9	4.1
ran8 x 32	592.4	1790.9	78.7	212.1	62.4	54.7	**38.3**	141.8	59.3	63.7	**47.2**	133.0	76.2
ran10 x 26	>3600.0	>3600.0	192.5	463.0	89.0	99.5	**86.5**	183.1	100.3	104.7	**82.0**	294.2	113.0
ran12 x 21	>3600.0	>3600.0	412.4	757.1	**208.9**	272.5	214.5	317.4	222.2	250.8	**199.0**	529.3	419.1
ran13 x 13	1189.5	1168.8	217.7	287.4	**132.8**	159.9	136.9	233.7	**134.4**	144.8	136.1	355.0	126.9
mas284	146.4	116.5	24.6	59.5	28.5	28.8	**25.9**	48.5	27.8	**23.7**	24.1	71.0	15.4
prod1	>3600.0	1741.5	151.0	405.1	112.9	**94.6**	100.8	287.1	101.1	**96.2**	97.8	359.3	185.7
bc1	1243.8	1317.7	1218.5	1180.5	1337.0	1272.9	**1120.0**	1108.2	1282.2	1295.2	**1153.0**	1071.5	1947.4
bienst1	868.2	749.2	230.9	243.0	121.3	**107.9**	115.3	256.8	138.4	**83.3**	91.0	270.3	194.1
neos2	>3600.0	>3600.0	>3600.0	872.8	693.6	675.6	**206.7**	277.2	883.6	385.5	**222.2**	216.9	671.4
neos3	>3600.0	>3600.0	>3600.2	2882.3	>3600.0	>3600.0	**2707.2**	1284.6	>3600.0	>3600.0	**3145.8**	1550.2	2513.8
neos7	>3600.0	>3600.0	2009.4	>3600.0	>3600.0	2661.3	**1039.8**	>3600.0	>3600.0	2377.6	**1350.3**	>3600.0	1831.7
swath1	330.3	894.3	173.4	795.2	566.0	**335.3**	407.3	777.6	668.6	604.6	**166.2**	1166.1	693.1
swath2	1059.4	>3600.0	2285.3	2876.3	842.3	3109.8	**407.2**	>3600.0	1847.6	2864.5	**1034.6**	>3600.0	3108.2
Total (24)	46202.2	48037.5	19945.8	26397.2	16499.4	17199.6	11132.7	20427.0	18362.5	16753.8	12380.2	25618.6	18617.4
Geom. mean	923.6	938.0	283.4	504.4	229.6	216.4	170.5	353.2	253.9	217.2	170.2	468.0	215.8

Table 3
Strong branching evaluations performed and maximal depth of random branching

		Lookahead= 4				*Lookahead= 8*				*Avg. depth*
Example	Full str.	Str/ps(10)	Reli(1)	Reli(8)	Strong	Str/ps(10)	Reli(1)	Reli(8)	Strong	Full str.
aflow30a	972 871	3625	396	2847	239 787	5604	397	2865	495 934	38.7
cap6000	4602	762	92	597	4683	762	92	597	4683	16.7
gesa2-o	401 182	2231	435	2331	114 464	3373	1841	2178	235 608	28.8
mas74	8 773 236	6609	98	649	6 094 848	10 531	88	673	6 617 267	23.3
mas76	1 508 479	6307	69	530	861 005	9080	69	558	1 551 253	21.2
misc07	85 451	3120	379	2471	65 988	2943	266	3262	84 665	29.9
pk1	897 902	7008	54	391	879 768	9334	54	402	1 256 800	24.6
pp08aCUTS	2227	954	61	433	1498	1531	60	438	1530	7.3
qiu	79 685	6879	48	363	47 170	11 732	48	370	87 289	11.6
rout	114 383	3004	573	2703	58 916	5554	426	3473	210 641	22.1
vpm2	21 665	1948	165	881	12 712	2971	141	1089	15 435	18.0
ran8 x 32	129 698	1770	570	2657	63 719	2765	667	2534	58 732	24.8
ran10 x 26	267 261	3038	536	1985	88 240	3771	586	3417	144 009	30.2
ran12 x 21	456 400	5414	593	2675	141 880	6588	940	2591	269 767	25.5
ran13 x 13	210 862	5304	167	1276	144 070	7920	311	1521	234 413	26.3
mas284	54 311	4350	110	462	32 685	5010	85	477	47 686	14.9
prod1	209 540	151	153	1202	118 575	335	158	1173	152 860	32.9
bc1	17 413	804	3189	5092	16 891	787	3189	5092	16810	35.9
bienst1	31 222	4150	27	211	27 841	4380	27	206	30 331	13.8
neos2	115 097	1076	9998	15 893	44 008	267	13 311	13 994	37 362	121.6
neos3	305 325	1264	11 730	45 143	170 679	270	19 218	54 459	223 816	134.6
neos7	475 374	1058	6170	11 256	344 673	171	5944	17 062	417 934	130.2
swath1	111 797	1682	1598	8170	80 944	3235	2088	6474	129 431	28.1
swath2	323 669	2304	1915	7425	310 410	3168	3551	8640	327 248	38.0
Total (24)	15 569 652	74 812	39 126	117 643	9 965 454	102 082	53 557	133 545	12 651 504	—
Geom. mean	146 784	2284	375	1850	86 189	2524	429	1998	119 799	28.2

Table 4
Summary of all considered strategies

Strategy	B&B nodes		Time (s)		Strong branchings		Fails
	Total	Geom.	Total	Geom.	Total	Geom.	
Random	21 246 277	275 789.4	46 202.2	923.6	0	0.0	11
Most infeasible	19 421 589	262 368.9	48 037.5	938.0	0	0.0	11
Pseudocost	9 381 001	88 706.8	19 945.8	283.4	0	0.0	2
Full strong	929 347	7241.7	26 397.2	504.4	15 569 652	146 784.2	2
Lookahead=4							
Strong/pscost (5)	9 698 397	79 535.5	19 487.8	249.3	5792	216.2	2
Strong/pscost (10)	8 251 942	65 966.3	16 499.4	229.6	74 812	2284.2	2
Strong/pscost (15)	7 982 847	57 976.8	17 855.3	258.8	523 377	8137.8	2
Strong/pscost (20)	7 890 374	47 958.5	19 175.6	293.5	2 825 100	17 780.2	2
Reliability (1)	9 000 334	70 013.6	17 199.6	216.4	39 126	374.8	1
Reliability (4)	6 906 698	53 522.9	13 402.9	178.2	110 628	1176.7	0
Reliability (8)	7 937 968	48 772.8	11 132.7	170.5	117 643	1 850.3	0
Reliability (16)	6 022 024	44 649.9	10 782.6	179.0	187 578	3 640.6	0
Reliability (32)	7 940 797	39 655.2	11 103.0	184.2	253 014	5 837.8	0
Strong branching	1 325 589	11 639.5	20 427.2	353.2	9 965 454	86 188.6	3
Lookahead=8							
Strong/pscost (5)	8 653 318	74 730.8	17 389.5	239.6	7397	268.6	1
Strong/pscost (10)	8 332 543	69 489.0	18 362.5	253.9	102 082	2523.8	2
Strong/pscost (15)	7 456 685	59 398.8	20 479.8	295.7	750 577	9 983.1	2
Strong/pscost (20)	7 551 419	48 736.5	22 388.8	343.1	3 695 577	20 837.9	3
Reliability (1)	8 663 537	69 501.0	16 753.8	217.2	53 557	429.0	1
Reliability (4)	8 338 386	54 937.7	12 497.2	179.2	74 906	1 104.6	0
Reliability (8)	7 813 409	48 377.3	12 380.2	170.2	133 545	1 998.0	0
Reliability (16)	7 579 400	43 311.9	11 946.7	171.3	185 136	3589.7	0
Reliability (32)	7 207 836	42 047.5	11 835.7	186.5	259 482	5913.2	0
Strong branching	1 294 569	12 714.7	25 619.4	468.0	12 651 504	119 799.1	4
Lookahead=∞							
Strong/pscost (5)	8 498 292	71 817.4	18 116.9	229.3	14 675	489.9	2
Strong/pscost (10)	9 247 636	70 125.8	20 472.1	276.2	154 458	3870.1	2
Strong/pscost (15)	6 670 440	56 926.6	19 907.4	312.2	890 187	13 127.2	3
Strong/pscost (20)	7 627 640	48 547.0	23 538.5	373.6	3 842 516	26 557.4	3
Reliability (1)	7 747 290	72 159.1	15 825.8	220.0	48 162	408.4	1
Reliability (4)	9 068 723	58 886.4	14 258.1	195.8	78 625	1096.6	2
Reliability (8)	8 551 045	54 118.3	13 563.0	189.6	135 541	2042.9	1
Reliability (16)	6 567 432	49 839.9	12 766.4	191.6	196 220	3601.0	0
Reliability (32)	7 502 942	41 636.1	12 393.6	192.5	281 822	6000.7	0
Strong branching	1 163 822	11 355.7	26 176.4	500.3	12 793 364	127 737.1	4
CPLEX/SIP cuts	10 467 429	79 269.0	18 617.4	215.8	—	—	1

infeasible after two variables are fixed. Since *hybrid strong/pseudocost branching* uses a fixed depth for deciding where to do *strong branching*, only very few strong branching evaluations are performed, as shown in Table 3.

The last column of Table 3 gives the average depth of all nodes generated when using *full strong branching*. If one compares this to the depth setting of *hybrid strong/pseudocost branching* and looks at the number of strong

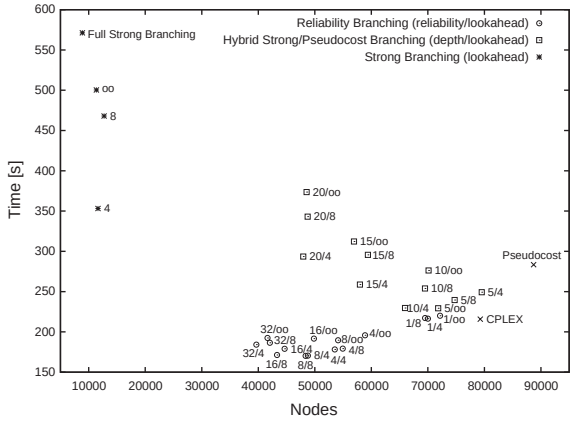

Fig. 2. Nodes vs. time.

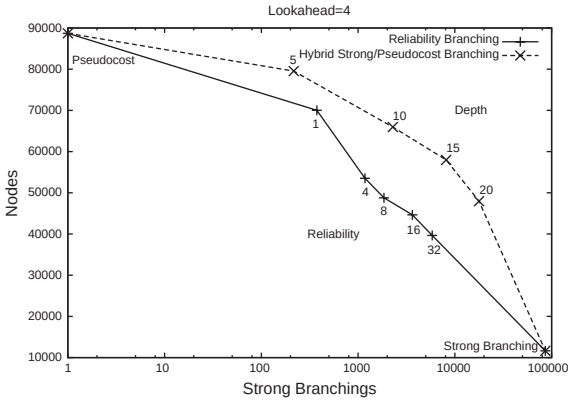

Fig. 3. Nodes vs. strong branching evaluations.

branchings performed, see also Table 3, the difficulties with strategies based on some fixed setting become obvious.

(iii) The lookahead λ from a certain value upwards does not seem to have much influence on the number of nodes.

The higher the settings the more time is spent per node evaluation, which seems not to pay off. This is not really a surprise recalling that a lookahead of four means, *no new best candidate found in four consecutive tries*. Since the candidates are already ordered by pseudocost value and the lookahead counter is reset with every new best candidate found, a setting of four turns out to be good enough to find the overall best candidate in most cases.

(iv) Increasing the reliability η_{rel} in *reliability branching* or the depth d in *hybrid strong/pseudocost branching* decreases the number of evaluated nodes as expected.

See Fig. 3, where the ratio of the number of nodes to strong branching evaluations is shown. With an increasing number of strong branching evaluations we are converging towards *strong branching* with respect to the number of evaluated nodes. The curve of *reliability branching* is always below the curve of *hybrid strong/pseudocost branching* indicating that the information provided by *strong branching* is used in a much better way.

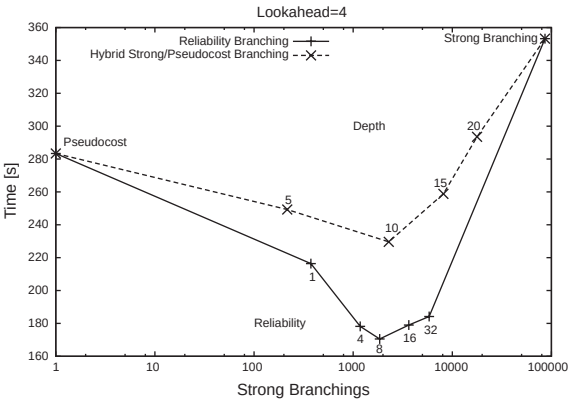

Fig. 4. Time vs. strong branching evaluations.

Fig. 4 on the other hand demonstrates the tradeoff between number of nodes and time per node. We again see that *reliability branching* is always better than *hybrid strong/pseudocost branching*. Fig. 4 also nicely reflects that *pseudocost branching with strong branching initialization* ($\eta_{\text{rel}} = 1$), see Section 2.5, is better than *pseudocost branching* itself ($\eta_{\text{rel}} = 0$), but this is not the best choice. The performance increases up to $\eta_{\text{rel}} = 8$ and decreases with larger values again.

Looking once again at Fig. 2 we see that the setting ($\eta_{\text{rel}} = 8, \lambda = 4$) for *reliability branching* marks a new sweet spot.

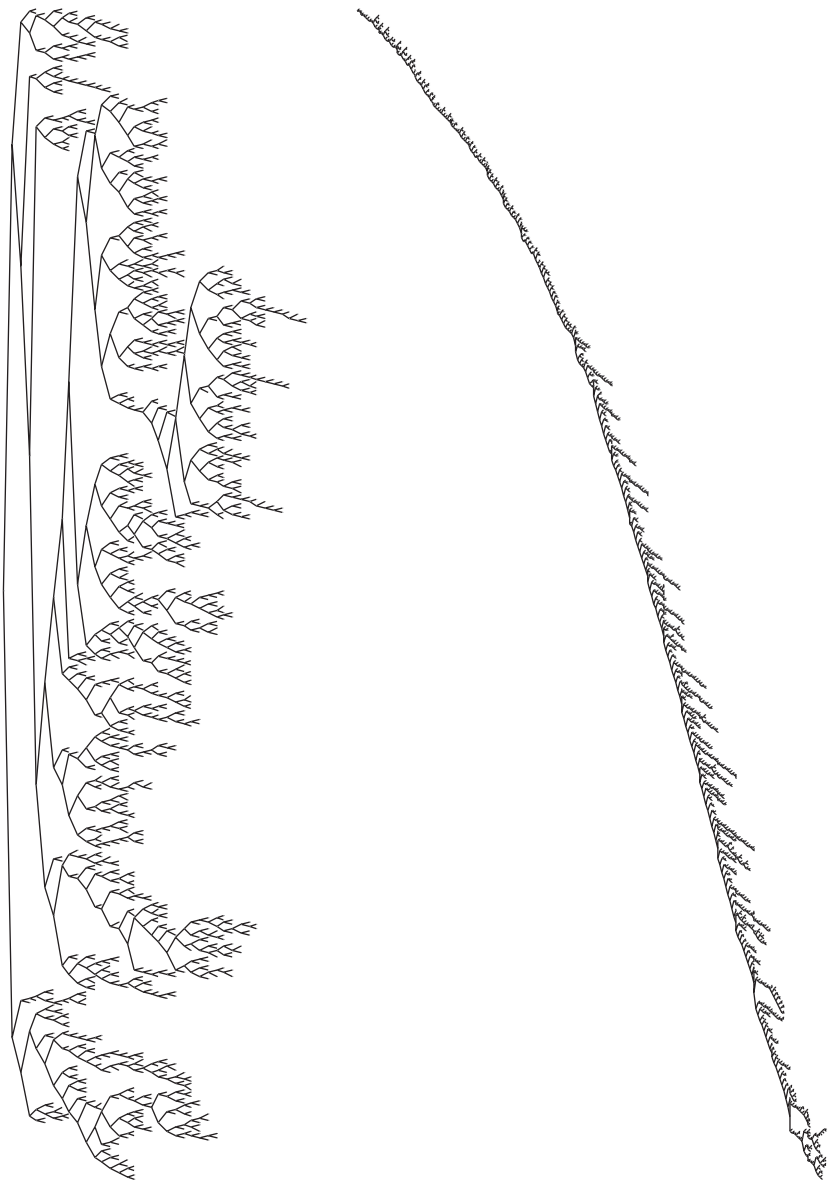

Fig. 5. Comparison of node trees resulting from *full strong branching* for **vpm2** and **neos3**.

4. Conclusion

It was shown that a more intensive dynamic use of *strong branching* leads to significant improvements in both the number of B&B nodes and the time needed to solve the considered problem instances.

It also became evident, that there is still a gap to the number of nodes needed using *full strong branching* Fig. 2. The question is whether it is possible to bridge this gap without increasing the time spent per node too much.

References

[1] T. Achterberg, T. Koch, A. Martin, The mixed integer programming library: MIPLIB 2003, 2003. http://miplib.zib.de.

[2] D. Applegate, R.E. Bixby, V. Chvátal, W. Cook, Finding cuts in the TSP, Technical Report 95-05, DIMACS, March 1995.
[3] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, O. Vincent, Experiments in mixed-integer programming, Math. Programming 1 (1971) 76–94.
[4] R. Borndörfer, C.E. Ferreira, A. Martin, Decomposing matrices into blocks, SIAM J. Optim. 9 (1998) 236–269.
[5] J.M. Clochard, D. Naddef, Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem, in: L.A. Wolsey, G. Rinaldi (Eds.), Proceedings on the Third IPCO Conference, 1993, pp. 291–311.
[6] B. Fourer, 2003 Software survey: linear programming, OR/MS Today 30 (6) (2003) 34–43.
[7] ILOG CPLEX. Reference Manual, 2003. http://www.ilog.com/products/cplex.
[8] A. Land, S. Powell, Computer codes for problems of integer programming, Ann. Discrete Math. 5 (1979) 221–269.
[9] S. Leipert, VBCTOOL—a graphical interface for visualization of branch cut algorithms, 1996. http://www.informatik.uni-koeln.de/ls_juenger/research/vbctool.
[10] J.T. Linderoth, M.W.P. Savelsbergh, A computational study of search strategies for mixed integer programming, INFORMS J. Comput. 11 (1999) 173–187.
[11] LINDO. API Users Manual, 2003. http://www.lindo.com.
[12] A. Martin, Integer programs with block structure, Habilitations-Schrift, Technische Universität, Berlin, 1998.
[13] G. Mitra, Investigations of some branch and bound strategies for the solution of mixed integer linear programs, Math. Programming 4 (1973) 155–170.
[14] H. Mittelmann, Decision tree for optimization software: Benchmarks for optimization software, 2003. http://plato.asu.edu/bench.html.
[15] D. Naddef, Polyhedral theory and branch-and-cut algorithms for the symmetric TSP, in: G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and its Variations, Kluwer, Dordrecht, 2002.