Branching on History Information

Tobias Achterberg [*] Thorsten Koch [†] Alexander Martin [‡]

September 4, 2002

Abstract

Mixed integer programs (*MIPs*) are commonly solved with branch and bound algorithms based on linear programming. The success and the speed of the algorithm strongly depends on the strategy used to select the branching variables. Today's state-of-the-art strategy is called *pseudocost branching* and uses information of previous branchings to determine the current branching.

We propose a modification of *pseudocost branching* which we call *history branching*. This strategy has been implemented in SIP, a state-of-the-art *MIP* solver. We give computational results that show the superiority of the new strategy.

1 Introduction

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

$$\begin{aligned} c^\star = {} & \min c^T x \\ & Ax \le b \\ & x \in \mathbb{Z}^I \times \mathbb{R}^{N\setminus I}, \end{aligned} \tag{1}$$

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

Among the most successful methods are currently linear programming based branch and bound algorithms where the underlying linear programs (*LPs*) are possibly strengthened by cutting planes. For example, most commercial integer programming solvers, see [Sha95], or special purpose codes for problems like the traveling salesman problem are based on this method. As we will see below, branch and bound algorithms leave two choices: how to

[*]Konrad-Zuse-Zentrum für Informationstechnik Berlin, achterberg@zib.de

[†]Konrad-Zuse-Zentrum für Informationstechnik Berlin, koch@zib.de

[‡]Technische Universität Darmstadt, martin@mathematik.tu-darmstadt.de

split a problem (branching) and which (sub)problem to select next. In this paper we focus on the branching step and introduce a new branching rule that performs in most of the cases better than the current rules when tested on real-world instances.

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

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

where $P_{LP} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$. Obviously, $\bar{c}_{P_{LP}} \leq c^\star$, since $P_{LP} \supseteq X_{MIP}$.

A typical *LP* based branch and bound algorithm for solving (1) looks as follows:

Algorithm 1.1 (branch and bound)
Input: A *MIP* in the form (1).
Output: An optimal solution $x^\star \in X_{MIP}$ and its value $c^\star = c^T x^\star$ or the conclusion that $X_{MIP} = \emptyset$, denoted by $c^\star := \infty$.

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

In Section 2 we review current branching strategies from literature and in Section 3 we present our new selection rule. In Section 4 we compare all strategies on some test problems and we indicate, why the new rule performs better than those known from literature on the given problem instances.

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

2 Current Branching Rules

Since branching is in the core of any branch and bound algorithm there is a huge amount of literature on this topic. 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 branch and bound strategies we refer to [LP79] and [LS99] and the references therein.

The only way to split a problem Q within an *LP* based branch and bound 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 $f_i^- \leq 0$ (called the *left subproblem* or *left son*, denoted by Q_i^-) and one by adding the trivial inequality $f_i^+ \leq 0$ (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 actually does not require to add an inequality, but only 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, but turn out to be effective in special cases, see, for instance, [BFM98], [CN93], or [Nad01]. In the following we focus on the most common variable selection rules.

The ultimate goal is to find a fast branching strategy that minimizes the number of branch and bound nodes that need to be evaluated. Since a global approach to this problem is unlikely, one tries to find a branching variable that is at least a good choice for the next branching. The quality of a branching is measured by the change in the objective function of the *LP* relaxations of the two sons 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. [LS99])

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

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.

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 analogously.

2.1 Most Infeasible Branching

This still very common rule chooses the variable with fractional part closest to 0.5. The heuristic reason behind this choice is that this is a variable where the least tendency can be recognized to which "side" (up or down) the variable should be rounded. The hope is that a decision on this variable has the greatest impact on the *LP* relaxation.

2.2 Strong Branching

The idea of *strong branching*, invented by [CPL01] (see also [ABCC95]), is before actually branching on some variable to test which of the fractional candidates gives the best progress. This testing is done by temporarily introducing an upper bound $\lfloor \bar{x}_i \rfloor$ and subsequently a lower 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.

Unfortunately the computation times of *full strong branching* are prohibitive. Accordingly all branching rules we present from now on, try to find a (fast) estimate of what *full strong branching* actually measures.

One possibility to speed strong branching up, is to restrict the candidate set in some way, e.g. by considering only part of the fractional variables. To estimate the changes in the objective function for a specific branching, often only a few simplex iterations are performed, because the change of the objective function in the simplex algorithm is usually decreasing with the iterations. Thus, the parameters of *strong branching* to be specified are the size of the candidate set, the maximum number of dual simplex iterations to be performed on each candidate variable, and a criterion according to which the candidate set is selected.

2.3 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 [BGG$^+$71]. In the meantime various variations of the original have been proposed and implemented. In the following we present the one used in SIP. For alternatives see [LS99].

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 let η_i^+ be the number of these problems.

Then the pseudocosts for the upward branching of variable i are

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

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

A special treatment is necessary, if the subproblems Q_i^+ is infeasible. In SIP, an infeasible Q_i^+ leads to the pseudocost update $\sigma_i^+ \leftarrow \sigma_i^+ + \kappa \cdot \Psi_{\text{avg}}^+$ and $\eta_i^+ \leftarrow \eta_i^+ + 1$, where κ is some constant. Now *pseudocost branching* works as follows:

Algorithm 2.1 (pseudocost branching)
Input: Actual subproblem Q with an optimal *LP* solution $\bar{x} \notin X_{MIP}$.
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 the score $s_i = \text{score}(\Psi_i^-, \Psi_i^+)$.
3. Return an index $i \in C$ with $s_i = \max_{j \in C}\{s_j\}$.

2.4 Hybrid Pseudocost Branching

Even with the limitations indicated at the end of Section 2.2, the computational burden of *strong branching* is high, and the higher the speed up, the less precise the decisions are. On the other hand, the weakness of *pseudocosts* is that at the very beginning there is no information available, and s_i is almost identical for all variables. Thus, at the beginning where the branching decisions are usually the most crucial, the pseudocosts take no effect.

The following rule, which is called *hybrid pseudocost branching* tries to circumvent these drawbacks and to combine the positive aspects of *pseudocost branching* and *strong branching*. The idea, which goes back to [LS99], is to use *strong branching* only for variables where no pseudocost values are at hand and use these *strong branching* values to initialize the pseudocosts. The following algorithm describes the changes to Algorithm 2.1 as implemented in SIP.

Algorithm 2.2 (hybrid pseudocost branching)
Input: Actual subproblem Q with an optimal LP solution $\bar{x} \notin X_{MIP}$.
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 the score $s_i = \text{score}(\Psi_i^-, \Psi_i^+)$.
3. For all candidates $i \in C$ with uninitialized pseudocosts, beginning with
   $\text{argmin}_i |\bar{x}_i - \lfloor \bar{x}_i \rfloor - 0.5|$, recalculate the pseudocosts as follows:
   (a) Perform a small number of dual simplex iterations on each subproblem $Q_i^-$ and $Q_i^+$.
   (b) Update the pseudocosts (5), but instead of $\bar{c}_{Q_i^-}$ and $\bar{c}_{Q_i^+}$ in (4)
       use the suboptimal solution values $\tilde{c}_{Q_i^-}$ and $\tilde{c}_{Q_i^+}$ acquired after
       the limited simplex iterations.
   (c) Recalculate the score $s_i = \text{score}(\Psi_i^-, \Psi_i^+)$.
   (d) If the maximum score $\max_{j \in C}\{s_j\}$ does not change for $\lambda$ consecutive
       initializations, goto 4.
4. Return an index $i \in C$ with $s_i = \max_{j \in C}\{s_j\}$.
```

The parameter λ is some small constant in SIP. *Hybrid pseudocost branching* almost always outperforms the standard version, see for instance [LS99, Mar98, CPL01]. Therefore, we only consider the *hybrid* strategy for comparison from now on.

3 History Branching

As we have seen in the previous section the major tasks for designing a good branching rule, which predicts the gain in the objective function well,i. e., resembles *full strong branching*, are to decide on the quantities for the prediction and how to initialize and update them. In the following we suggest some new ideas for these tasks. For the prediction of the gain in the objective function we define new quantities, which we call *history values.*

Let $\tilde{\eta}_i^+$ denote the number of times strong branching has been applied to variable i for upward branching. Note that we will allow strong branching to be performed more than once for each variable in contrast to hybrid pseudocost branching. Then we calculate similar as in Section 2.3 quantities

$$\tilde{\Psi}_i^+ = \tilde{\sigma}_i^+ / \tilde{\eta}_i^+, \tag{6}$$

where we use in (4) in the numerator the value $\tilde{\Delta}_i^+ = \tilde{c}_{Q_i^+} - \bar{c}_Q$ that we obtain from strong branching instead of Δ_i^+. Now for a given *LP* solution $\bar{x}$ the *history value* for variable i is defined as

$$\omega_i^+ = \begin{cases} f_i^+ \, \alpha_i^+ \, \Psi_i^+, & \text{if } \eta_i^+ > 0, \qquad \text{(pseudocost)} \\ f_i^+ \, \tilde{\alpha}_i^+ \, \tilde{\Psi}_i^+, & \text{if } \eta_i^+ = 0, \, \tilde{\eta}_i^+ > 0, \quad \text{(strong branching)} \\ f_i^+, & \text{if } \eta_i^+ = 0, \, \tilde{\eta}_i^+ = 0. \quad \text{(uninitialized)} \end{cases} \tag{7}$$

The scaling factors $\tilde{\alpha}^+$ and α^+ in (7) are calculated dynamically in time, and represent the average scaling factors needed to achieve the correct gain (which is available after solving a subproblem). This scaling is necessary to compensate for the fact that $\tilde{\Delta}_i^+ \leq \Delta_i^+$ due to the limited number of simplex iterations used.[1]

In case $\eta_i^+ = \tilde{\eta}_i^+ = 0$ the computation of the score for variable i is even more involved than indicated in (7). History values do not only exist for variables, but we also introduce a *global history value* and a history value for so-called *classes*. The *global history value* ω_g^+ is an average of the history values of all variables, similar to the one used in pseudocost branching. A *class* is a group of variables having some common properties.[2] A *class history value* ω_c^+ is

[1]The exact computation of α^+ ($\tilde{\alpha}^+$ is defined accordingly) is done as follows: Whenever we decided to branch on some variable i, we keep track of whether the value of the score function resulted from the strong branching or the pseudocost value. n^+ counts the number of times the score value came from pseudocosts. In addition, we have quantities g^+ and p^+ which reflect the sums of the gains and predictions when the pseudocosts determined the score value. Both are initialized with 0 and updated for a node Q, where i was selected as branching variable, via

$$g^+ \leftarrow \tau^+ \cdot g^+ + (1 - \tau^+) \cdot \Delta_i^+, \quad \text{and} \quad p^+ \leftarrow \tau^+ \cdot p^+ + (1 - \tau^+) \cdot \Psi_i^+.$$

Note that earlier evaluated nodes are weighted by an exponentially decreasing factor influenced by τ^+, defined as

$$\tau^+ = \min\{\frac{n^+}{n^+ + 1}, \tau_{\max}\}, \quad \text{with } \tau_{\max} \in [0, 1].$$

A maximal weight factor $\tau_{\max} = 1$ results in g^+- and p^+-values resembling exactly the average gain and prediction of all nodes. SIP uses a value of $\tau_{\max} = 0.995$, which leads to "forgetting" earlier node evaluations and adapting to the situation of the last nodes.

Now the scaling factor is defined as $\alpha^+ = g^+ / p^+$. The scaling factor for strong branching $\tilde{\alpha}^+$ is determined analogously, where all the above parameters $g^+, p^+, \tau^+, n^+, \Psi^+$ are substituted by their corresponding tilde values $\tilde{g}^+, \tilde{p}^+, \tilde{\tau}^+, \tilde{n}^+, \tilde{\Psi}^+$.

Observe that the scaling factors $\tilde{\alpha}^+$ and α^+ are quotients of two average values. A second, and maybe more natural way of calculating the factors would be taking the average of the individual quotients. We choose the way described above, because it sets priority on nodes with large changes in the dual bound. With the latter approach, a quotient with denominator near zero could easily dominate the whole average quotient.

[2]At the moment, the variables are grouped with respect to their name in the model, or (if the name classification fails) with respect to their objective function coefficient, lower and upper bounds, and constraint matrix entries. In the future, we plan to link SIP to the modeling language ZIMPL [Koc01], from where the variable classes come out naturally.

Feature	Hybrid Pseudocost	History
hierarchy levels	2 levels: variable and global pseudocosts	3 levels: variable, class, and global history
infeasible subproblems	update pseudocosts	don't update history values
nodes, where preprocessing or strengthening cuts were applied	update pseudocosts	don't update history values
use of *strong branching* information	only for uninitialized pseudocost values; no distinction between values from *strong branching* and from solved subproblems	also for unreliable history entries; values are stored separately from solved subproblem values and are dynamically scaled
order of candidate variables for *strong branching* evaluation	ordered by decreasing fractionality $\lvert \bar{x}_i - \lfloor \bar{x}_i \rfloor - 0.5 \rvert$	ordered by decreasing history score values, starting with uninitialized entries
simplex iterations in *strong branching*	fixed	dynamically adjusted

Table 1. *Differences between hybrid pseudocost and history branching.*

the average of all ever calculated history values of the variables belonging to the particular class c. Now the score of variable i is determined from the values in (7) if $\eta_i^+ + \tilde{\eta}_i^+ > 0$, otherwise we look at the entry w_c^+ of the corresponding class c in case $\eta_c^+ + \tilde{\eta}_c^+ > 0$ and finally, if the latter condition does not hold, we use the global history value ω_g^+.

Initialization of history values via *strong branching* is not only performed for uninitialized history entries, but also for "unreliable" variables. A variable i is called *unreliable* if $\eta_i^+ + \tilde{\eta}_i^+ < \eta_{\text{rel}}$, which is a dynamically adjusted parameter depending on the number of integer variables $|I|$. The candidates are initialized in a decreasing order of their history scores, where variables with $\eta_i^+ + \tilde{\eta}_i^+ = 0$, called *uninitialized* variables, are always initialized before the unreliable ones. Note that in each branching step at least for a certain number of uninitialized or unreliable variables strong branching is performed. For these variables the outcome of strong branching is used in the actual branching decision.

In *updating* the history values, we ignore branching results, where one of the subproblems is infeasible or where node preprocessing or strengthening cuts where applied at the child nodes, because this would lead to unfair large gain values in comparison to other nodes.

Nearly all parameters, like the number of simplex iterations used for initializing history values, the maximum number of history entries to be initialized at one node, or the maximal size η_{rel} for entries being called unreliable, are dynamically adjusted depending on measurements like the ratio of total time used for branching and total time used for solving *LP* subproblems. Furthermore, most of the parameters depend on the current node's depth such that the effort of *strong branching* is increased at the upper nodes.

The following Algorithm 3.1 outlines once more the selection of a branching variable by history values and Table 1 summarizes the main differences to hybrid pseudocost branching.

Algorithm 3.1 (history branching)
Input: Actual subproblem Q with an optimal *LP* solution $\bar{x} \notin X_{MIP}$.
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 the score $s_i = \text{score}(\omega_i^-, \omega_i^+)$ and sort them in non-increasing order of their history score.
3. First for uninitialized and then for unreliable variables $i \in C$ do:
   (a) Perform a small number of dual simplex iterations on each subproblem Q_i^- and Q_i^+. Let $\tilde{\Delta}_i^-$ and $\tilde{\Delta}_i^+$ be the outcome.
   (b) Update the strong branching history values $\tilde{\Psi}_i^-$ and $\tilde{\Psi}_i^+$.
   (c) Recalculate the score $s_i = \text{score}(\tilde{\alpha}^- \tilde{\Delta}_i^-, \tilde{\alpha}^+ \tilde{\Delta}_i^+)$.
   (d) If at least a certain number of variables has been tried and the maximum score $\max_{j \in C}\{s_j\}$ has not changed for λ consecutive initializations, goto 4.
4. Return an index $i \in C$ with $s_i = \max_{j \in C}\{s_j\}$.

4 Numerical Results

In this section we present computational results of *hybrid pseudocost branching* and *history branching* on several *MIP* instances. Our test set consists of those MIPLIB 3.0 [BCMS98] problems, where CPLEX 7.5 [CPL01] needs more than 1000 branching nodes[3], and which SIP using *history branching* solves in less than 3600 CPU seconds[4], which was our time limit in all runs. Note that SIP also solves all of the excluded small problems except `modglob` (2007 nodes) in less than 1000 nodes. And CPLEX neither solves one of the excluded big instances in less than an hour.

[3]This excludes `10teams`, `air03/04/05`, `dcmulti`, `dsbmip`, `egout`, `fiber`, `fixnet6`, `flugpl`, `gen`, `gesa2/3/3_o`, `gt2`, `khb05250`, `l152lav`, `lseu`, `misc03/06`, `mitre`, `mod008/010`, `modglob`, `nw04`, `p0033/0201/0282/0548/2756`, `pp08a/CUTS`, `qnet1/_o`, `rentacar`, `set1ch`, and `vpm1`

[4]This excludes `arki001`, `dano3mip`, `danoint`, `fast0507`, `harp2`, `markshare1/2`, `mas74`, `noswot`, `seymour`, and `swath`.

	pseudocost		history		CPlex 7.5	
Example	B & B	Time	B & B	Time	B & B	Time
bell3a	23161	12.3	24153	12.5	24185	19.6
bell5	9498	4.2	35409	14.0	659302	288.3
blend2	4662	9.1	3965	10.2	1884	2.2
cap6000	5201	114.0	3334	70.8	12085	258.8
enigma	2594	0.8	602	0.6	10782	3.2
gesa2_o	193064	2683.6	69336	513.3	1167	5.3
mas76	534909	265.6	451595	235.5	782033	255.0
misc07	118720	452.2	57342	276.0	111859	227.1
mod011	2134	131.5	625	120.1	7877	3039.0
pk1	332534	184.3	316338	185.1	347913	194.5
qiu	39955	713.1	10237	270.9	41558	931.8
rgn	2049	2.1	1632	2.6	2467	0.7
rout	> 478738	> 3600.0	9919	78.6	> 215797	> 3600.0
stein27	3113	2.3	2461	5.2	3661	1.1
stein45	51020	85.2	55252	96.8	74487	46.0
vpm2	21309	37.1	9158	22.9	7911	5.8
aflow_20_90	13662	51.2	4073	35.7	6707	42.7
aflow_22_90a	41615	160.7	15497	93.5	40533	278.1
aflow_22_90b	5930	44.9	2785	43.0	5391	48.9
aflow_30_50	294518	1808.3	60847	352.8	147583	1301.3
aflow_30_90	33627	689.1	9792	111.5	50503	835.9
aflow_40_50	50306	1914.9	34286	982.0	60530	956.9
Total (22)	2262319	12966.7	1178638	3533.5	2616215	12342.1

Table 2. *Branch and bound nodes (B&B) and time in seconds needed to solve each instance.*

In addition, we tested the branching rules on some randomly generated instances of the *arborescence flow problem* (see [Pfe00]).

To compare the results of SIP with CPlex, we used CPlex 7.5 with default options, except that we set the "absolute mipgap" to 10^{-10} and the "relative mipgap" to 0.0, which are the corresponding tolerances in SIP. Note that the version of SIP used here utilizes CPlex as embedded *LP* solver. All calculations were performed on an Alpha 21264 (750 Mhz) with 2 GB memory.

In Table 2, the number of branching nodes and the time needed to solve the problem instances are presented for SIP with *hybrid pseudocost branching*, SIP with *history branching* and CPlex 7.5. For nearly all instances, *history branching* outperforms *hybrid pseudocost branching.*

A problem with the numbers given in Table 2 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 faster solution time for

Example	random	most infeasible	fullstrong	pseudocost	history
bell3a	22881	22739	6513	22003	23655
bell5	496017	> 2975391	288683	*9689*	61097
blend2	6729	6527	121	1553	301
cap6000	6879	5001	2803	4525	3719
enigma	1	1	1	1	1
gesa2_o	> 627982	> 629791	6115	80674	27399
mas76	1913905	1444685	63113	551601	487529
misc07	47359	*15555*	16531	84259	51859
mod011	23861	4555	21	815	91
pk1	919461	654119	45685	321861	294627
qiu	372537	294621	14873	46375	15769
rgn	2137	1891	391	1681	1667
rout	> 1427694	> 1343046	1035	389791	10117
stein27	4481	4635	2187	4425	3219
stein45	62599	74527	24545	55959	53263
vpm2	113003	35073	881	15461	7895
aflow_20_90	861247	> 7401188	957	8949	1443
aflow_22_90a	> 1934913	> 2756897	2147	31611	7981
aflow_22_90b	1532441	> 8633642	573	3079	1347
aflow_30_50	> 1076483	> 1578088	9013	178729	34325
aflow_30_90	> 932820	> 1783439	1305	17767	6435
aflow_40_50	> 1070442	> 1632770	877	14425	1939
Total (22)	> 13455872	> 31298181	488370	1845233	1095678

Table 3. *Branch and bound nodes (B&B) needed to solve each problem instance only using cuts at the root node and supplying the optimal solution in advance.*

a specific problem, because the variable selection leads incidentally to an early discovering of a good or optimal primal solution.

For this reason, we ran a second test on the problem instances, see Table 3, where we provided the optimal solution in advance. Additionally, we generated cutting planes in the root node with SIP once, and used the resulting *MIP* with all cutting plane generations in the solvers disabled.[5] For reasons of comparison, we tried three more branching rules, namely *random branching*, *most infeasible branching*, and *full strong branching. Random branching* picks a random candidate out of the candidate list as the branching variable, *most infeasible* and *full strong branching* are described in Section 2.

As mentioned before, *full strong branching* is the "optimal" branching rule under our local goal to maximize the score at each node. Both, *pseudocost branching* and *history branching*, can be viewed as an approximation of *full strong branching.*

Table 3 shows the number of branching nodes needed to solve the problem

[5]In this setting `enigma` could always be solved in the root node.

	pseudocost (total)		history (total)		pseudocost (depth < 10)		history (depth < 10)	
Example	ξ	ρ	ξ	ρ	ξ	ρ	ξ	ρ
bell3a	0.997	0.997	0.946	0.963	0.983	0.991	1.000	1.000
bell5	0.930	0.927	0.920	0.852	0.706	0.816	0.998	0.973
blend2	0.686	0.758	0.763	0.813	0.597	0.783	0.967	0.986
cap6000	0.726	0.650	0.764	0.699	0.753	0.638	0.853	0.829
enigma	0.537	0.820	0.504	0.741	0.566	0.868	0.960	0.981
gesa2_o	0.589	0.846	0.673	0.862	0.547	0.869	0.856	0.970
mas76	0.439	0.690	0.589	0.779	0.606	0.686	0.726	0.794
misc07	0.260	0.649	0.346	0.673	0.449	0.574	0.525	0.707
mod011	0.526	0.751	0.557	0.723	0.535	0.756	0.771	0.852
pk1	0.416	0.694	0.542	0.764	0.574	0.686	0.799	0.863
qiu	0.316	0.704	0.605	0.732	0.451	0.679	0.736	0.836
rgn	0.647	0.592	0.618	0.604	0.726	0.645	0.959	0.963
rout	0.280	0.704	0.490	0.798	0.350	0.696	0.740	0.931
stein27	0.388	0.750	0.478	0.868	0.549	0.688	0.707	0.917
stein45	0.512	0.760	0.569	0.686	0.638	0.630	0.730	0.764
vpm2	0.432	0.647	0.528	0.711	0.512	0.745	0.679	0.864
aflow_20_90	0.415	0.775	0.506	0.835	0.417	0.691	0.797	0.931
aflow_22_90a	0.364	0.844	0.518	0.865	0.291	0.762	0.779	0.948
aflow_22_90b	0.441	0.848	0.514	0.893	0.383	0.840	0.730	0.951
aflow_30_50	0.346	0.824	0.418	0.873	0.306	0.799	0.575	0.918
aflow_30_90	0.418	0.800	0.424	0.843	0.291	0.717	0.669	0.900
aflow_40_50	0.266	0.796	0.603	0.922	0.335	0.836	0.698	0.942
Total (22)	0.497	0.764	0.585	0.795	0.526	0.746	0.784	0.900

Table 4. *Score quotients (ξ) and position quotients (ρ) of the full trees (left half), and the average quotients of the uppermost* 10 *depth levels (right half).*

instances. With the exception of the instances `bell5` and `misc07`, the local optimal *full strong branching* leads to the by far fewest branching nodes. *History branching* performs better than *hybrid pseudocost branching* on all instances except `bell3a` and `bell5`. On some instances like `gesa2_o`, `qiu`, `rout` and the `aflow` problems the improvement is significant.

In order to find out the reasons for the success of *history branching*, we analyzed the branchings performed in *hybrid pseudocost* and *history branching* with the settings of Table 2. At each node, we solved all subproblems associated to the fractional variables (i. e., the branching candidates), and compared the score s_{selected} of the selected variable with the score $s_{\max}$ of the best possible candidate under the given score function. This gives us two different measurements at each node n: the *score quotient*

$$\xi_n := \frac{s_{\text{selected}}}{s_{\max}}, \tag{8}$$

and the *position quotient*

$$\rho_n := 1.0 - \frac{\text{pos}_{\text{selected}} - 1}{|C| - 1}, \tag{9}$$

where $\text{pos}_{\text{selected}}$ is the position of the selected candidate in the candidate list C ordered by descending score. If there is only one candidate or if all candidates have the same score, we define both quotients as 1.0.

Both quotients have a value of 1.0, if an optimal (with respect to the score function) candidate was chosen, and are strictly less than one, if a sub-optimal candidate was selected. The smaller the number the worse the candidate. Note that *full strong branching* always has score and position quotients of $\xi = \rho = 1.0$.

Table 4 shows that *history branching* yields better score and position quotients than *hybrid pseudocost branching* on nearly all of the problems. This results in a behavior that more closely resembles *full strong branching.* Accordingly the number of branching nodes is smaller in those cases where *full strong branching* performs well.

Especially on the problems `blend2`, `gesa2_o`, `mas76`, `mod011`, `qiu`, `rout` and all `aflow` instances, the differences in the average score quotient are substantial and coincide with substantial differences in the number of nodes needed.

Note that `bell5` on which *full strong branching* performed badly, is not well handled by *history branching* as well. One reason for this might be that all score quotients are very close to one, which indicates a lack of discrimination for the branching decision.

In the upper (and more important) part of the branch and bound tree, the differences in the average score and position quotients are even stronger. This results from the more frequent use of *strong branching* (primarily in the top level nodes) and the more balanced comparisons between values derived from *history* and *strong branching* as described in Section 3. A comparison of Table 2 and Table 4 shows that large differences in the number of branch and bound nodes coincide with large differences in the average score quotients of the top level nodes.

A second indication, why *history branching* performs better than *hybrid pseudocost branching*, is the different ability of the *history* and *pseudocost* values to predict the actual gains Δ^- and Δ^+ in the objective function. To demonstrate this consider Figure 5, which shows the history and pseudocost estimates compared to the actual gains for problem instance `aflow_40_50`. Looking at the average lines the history values give a more monotone and sharper prediction of the actual gains Δ^- and Δ^+. That means, a large history value is on average a more reliable indication of a large gain than a large pseudocost value.

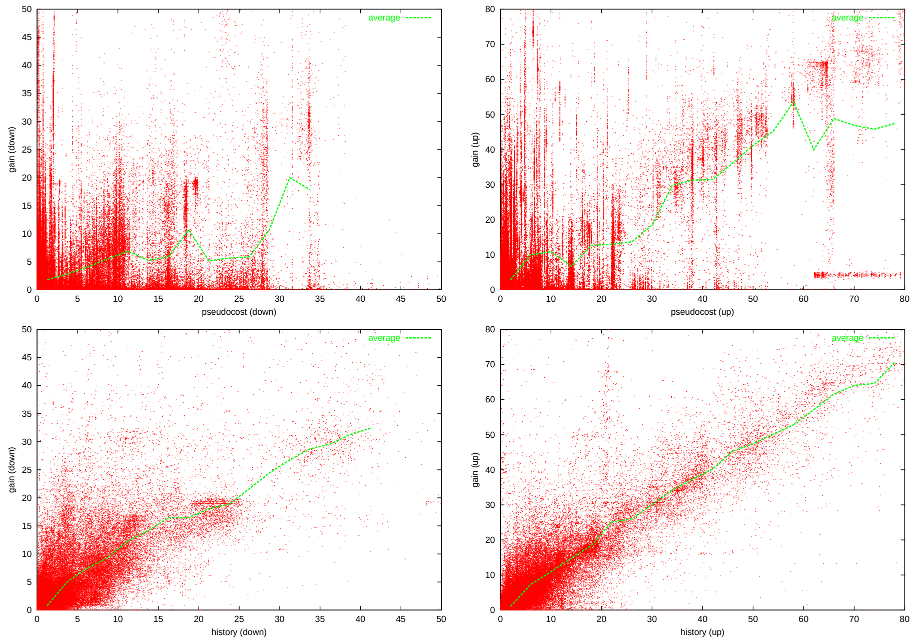

Figure 5. *Pseudocost and history predictions for* `aflow_40_50`. In the two left hand plots, the actual gains Δ^- for downward branching are plotted over the prediction values from pseudocosts Ψ^- in the upper plot, and history ω^- in the lower plot. On the right hand side, respective results are shown for the upward branching.

5 Conclusion

We presented a variant of the today's state of the art *pseudocost branching*, which we call *history branching*. Both branching rules have been implemented in SIP, a *LP* based branch and bound solver for mixed integer programs. It was shown that the superior gain prediction capabilities of the history values together with the dynamic parameter adjustments and the more intensive use of *strong branching* — specifically in the upper part of the branch and bound tree — leads to significant improvements in both, the number of branch and bound nodes and the time needed to solve the considered problem instances.

References

[ABCC95] D. Applegate, R. E. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS, March 1995.

[BCMS98] R.E. Bixby, S. Ceria, C. McZeal, and M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. Paper and Problems are available at WWW Page: http://www.caam.rice.edu/~bixby/miplib/miplib.html, 1998.

[BFM98] R. Borndörfer, C.E. Ferreira, and A. Martin. Decomposing matrices into blocks. *SIAM Journal on Optimization*, 9:236 – 269, 1998.

[BGG$^+$71] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer programming. *Mathematical Programming*, 1:76 – 94, 1971.

[CN93] J.M. Clochard and D. Naddef. Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem. In L.A. Wolsey and G. Rinaldi, editors, *Proceedings on the Third IPCO Conference*, pages 291–311, 1993.

[CPL01] ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. *ILOG CPLEX 7.5 Reference Manual*, 2001. Information available at URL http://www.cplex.com.

[Koc01] T. Koch. Zimpl user guide. Technical Report ZIB-Report 01-20, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2001.

[LP79] A. Land and S. Powell. Computer codes for problems of integer programming. *Annals of Discrete Mathematics*, 5:221 – 269, 1979.

[LS99] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.

[Mar98] A. Martin. Integer programs with block structure. Habilitations-Schrift, Technische Universität Berlin, 1998.

[Nad01] D. Naddef. Polyhedral theory and branch-and-cut algorithms for the symmetric TSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*. Kluwert, 2001. To appear.

[Pfe00] T. Pfender. Arboreszenz-Flüsse in Graphen: polyedrische Untersuchungen. Master's thesis, Technische Universiät Berlin, 2000.

[Sha95] R. Sharda. Linear programming solver software for personal computers: 1995 report. *OR/MS Today*, 22(5):49 – 57, 1995.