Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TIMO BERTHOLD AND DOMENICO SALVAGNIN*

Cloud Branching

* DEI, Via Gradenigo, 6/B, 35131 Padova, Italy, salvagni@dei.unipd.it

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

ZIB-Report 13-01 (January 2013)

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

Cloud Branching

Timo Berthold* and Domenico Salvagnin†

Abstract

Branch-and-bound methods for mixed-integer programming (MIP) are traditionally based on solving a linear programming (LP) relaxation and branching on a variable which takes a fractional value in the (single) computed relaxation optimum. In this paper we study branching strategies for mixed-integer programs that exploit the knowledge of *multiple* alternative optimal solutions (a *cloud*) of the current LP relaxation. These strategies naturally extend state-of-the-art methods like strong branching, pseudocost branching, and their hybrids.

We show that by exploiting dual degeneracy, and thus multiple alternative optimal solutions, it is possible to enhance traditional methods. We present preliminary computational results, applying the newly proposed strategy to full strong branching, which is known to be the MIP branching rule leading to the fewest number of search nodes. It turns out that cloud branching can reduce the mean running time by up to 30% on standard test sets.

Keywords: mixed integer programming, branching rule, search strategy, dual degeneracy

Mathematics Subject Classification: 90C11

1 Introduction

In this paper we address branching strategies for the exact solution of a generic mixed-integer program (MIP) of the form (w.l.o.g.):

$$\min\{cx : Ax \leq b \quad x_j \in \mathbb{Z} \quad \forall j \in J\}$$

where $x \in \mathbb{R}^n$ and $J \subseteq N = \{1, \ldots, n\}$.

Good branching strategies are crucial for any branch-and-bound based MIP solver. Unsurprisingly, the topic has been subject of constant and active research since the very beginning of computational mixed-integer programming, see, e.g., [1]. We refer to [2, 3, 4] for some comprehensive studies on branching strategies.

In mixed-integer programming, the most common methodology for branching is to split the domain of a single variable into two disjoint intervals. In this paper we will address the key problem of how to select such a variable. Let $x^\star$ be an optimal solution of the linear programming (LP) relaxation at the current node of the branch-and-bound tree and let $F = \{j \in J : x^\star_j \notin \mathbb{Z}\}$ denote the set of fractional variables. A general scheme for branching strategies consists in computing a score s_j for each fractional variable $j \in F$, and then picking the variable with maximum score. Different branching rules then correspond to different ways of computing this score.

Several branching criteria have been studied in the literature. The simplest one (*most-fractional branching*) is to branch on the variable whose fractional part is as close as possible to

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, berthold@zib.de
†DEI, Via Gradenigo, 6/B, 35131 Padova, Italy, salvagni@dei.unipd.it

0.5; however, this is well known to perform poorly in practice [5]. A more sophisticated branching strategy is *pseudocost branching* [1], which consists in keeping a history of how much the *dual bound* (the LP relaxation) improved when branching on a given variable in previous nodes, and then using these statistics to estimate how the dual bound will improve when branching on that variable at the current node. Pseudocost branching is computationally cheap since no additional LPs need to be solved and performs reasonably well in practice. Yet at the very beginning, when the most crucial branching decisions are taken, there is no reliable historic information to build upon.

Another effective branching rule is *strong branching* [6, 7]. The basic idea consists in simulating branching on the variables in F and then choosing the actual branching variable as the one that gives the best progress in the dual bound. Interestingly, this greedy local method is currently the best w.r.t. the number of nodes of the resulting branch-and-bound tree, but introduces quite a large overhead in terms of computation time, since $2 \cdot |F|$ auxiliary LPs need to be solved at every node. Many techniques have been studied to speedup the computational burden of strong branching, in particular by heuristically restricting the list of branching candidates and imposing simplex iteration limits on the strong branching LPs [2] or by ruling out inferior candidates during the strong branching process [8]. However, according to computational studies, a pure strong branching rule is still too slow for practical purposes. Branching rules such as *reliability branching* [3] or *hybrid branching* [9], that combine ideas from pseudocost branching and strong branching, are considered today's state of the art.

Other approaches to branching include the *active constraint* method [10], which is based on the impact of variables on the set of active constraints, branching on general disjunctions [11], *inference branching* and *VSIDS* [12, 13, 4] based on SAT-like domain reductions and conflict learning techniques. Finally, information collected through restarts is at the heart of the methods in [14, 15].

All branching strategies described so far are naturally designed to deal with only one optimal fractional solution. History-based rules use the statistics collected in the process to compute the score of a variable starting from the current fractional solution. Even with strong branching, the list of branching candidates is defined according to the current fractional solution $x^\star$.

However, LP relaxations of MIP instances are well-known for often being massively degenerate; multiple equivalent optimal solutions are the rule rather than the exception. Thus branching rules that consider only one optimal solution risk taking arbitrary branching decisions (thus contributing to performance variability, see [16]), or being unnecessarily inefficient. In the present paper we study the extension of some branching strategies to exploit the knowledge of multiple optimal solutions of the current LP relaxations.

The contribution of the present paper is twofold. First, we introduce for the first time, to the best of our knowledge, a branching strategy that makes use of multiple relaxation solutions and show how it can be naturally integrated into existing branching rules. Second, we evaluate one particular implementation of it in the context of full strong branching, the branching rule commonly known to be most efficient w.r.t. the number of branch-and-bound nodes [4, 17]. We show that it leads to significant savings in computation time while not increasing the number of nodes.

The remainder of the paper is organized as follows. In Section 2 we discuss how to generate alternative optimal solutions (a *cloud* of solutions) and how to exploit this information to enhance some standard branching rules such as pseudocost branching and strong branching. In Section 3 we give more details on the technique applied to full strong branching, while in Section 4 we report a preliminary computational evaluation of the proposed method. Some conclusions are finally drawn in Section 5.

2 A cloud of solutions

In order to extend standard branching strategies to deal with multiple LP optima at the same time, we need to solve two problems:

1. How to generate efficiently multiple optimal solutions of the current LP relaxation?
2. How to make use of the additional information provided by these solutions?

The first problem can be effectively solved by restricting the search to the optimal face of the LP relaxation polyhedron. On this face, an auxiliary objective function can be used to move to different bases. From the computational point of view, fixing to the optimal face can be easily and safely implemented by fixing all variables (structural and artificial) whose reduced costs are non-zero, using the reduced costs associated to the starting optimal basis. As far as the choice of the second level objective function(s) is concerned, different strategies can be used. One option is to try to minimize and maximize each variable which is not yet fixed: this is what optimality-based bound tightening techniques do (see, e.g., [18, 19]), with the additional constraint of staying on the optimal face. Another option is to use a feasibility pump [20] like objective function, in which the current LP point is rounded and a Hamming distance function is generated to move to a different point (more details will be given in the next section): this is related to the pump-reduce procedure that Cplex performs to achieve more integral LP optima [21]. Finally, a random objective function might be used.

Suppose now that we have constructed, in one way or another, a *cloud* $C = \{x^1, \ldots, x^k\}$ of alternative optimal solutions to the current LP relaxation. We assume that the initial fractional solution $x^\star \in C$. Given C, we can define our initial set of branching candidates $F(C)$ as

$$F(C) = \{j \in J \mid \exists x^i \in C : \; x^i_j \notin \mathbb{Z}\}$$

i.e., $F(C)$ contains all the variables that are fractional in at least one solution of the cloud. For each variable in $F(C)$ it is then possible to calculate its *cloud interval* $I_j = [l_j, u_j]$, where:

$$l_j = \min\{x^i_j \mid x^i \in C\}$$

$$u_j = \max\{x^i_j \mid x^i \in C\}$$

Given the cloud interval for each branching candidate, we partition the set $F(C)$ into three subsets, depending on the relative intersection between each interval I_j and the *branching interval* $B_j = [\lfloor x^\star_j \rfloor, \lceil x^\star_j \rceil]$. In particular, we define:

$$F_2 = \{j \in F(C) \mid \lfloor x^\star_j \rfloor < l_j \;\wedge\; u_j < \lceil x^\star_j \rceil\}$$

$$F_0 = \{j \in F(C) \mid l_j \leq \lfloor x^\star_j \rfloor \;\wedge\; \lceil x^\star_j \rceil \leq u_j\}$$

$$F_1 = F(C) \setminus (F_2 \cup F_0)$$

In particular for binary variables, F_2 contains exactly those variables which are fractional for all $x^i \in C$, or differently spoken: $F(C)$ is the union (taken over C) of all branching candidates, F_2 is the intersection. If C contained all vertices of the optimal face, then F_2 would be exactly the set of variables that are guaranteed to improve the dual bound in both child nodes. The hope is that also with a limited set of sample point in C, F_2 is still a good approximation to that set.

A variable being contained in the set F_0 is a certificate that branching on it will not improve the dual bound on either side since alternative optima exist which respect the bounds after

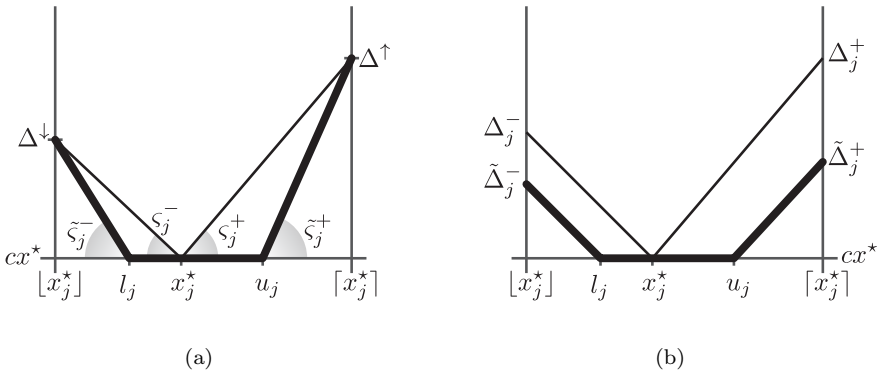

Figure 1: Graphical representation of pseudocosts update and usage.

branching. For the same reasoning, variables in F_1 are those for which the objective function will stay constant for one child, but hopefully not for the other.

The details about how branching rules can be extended to deal with this additional information, namely this three-way partition of the branching candidates (F_2, F_1, F_0) and the set of cloud intervals I_j of course depends on the particular strategy. For example, a rule based on strong branching can safely skip variables in F_0, thus saving some LPs (more details on how to extend a full strong branching policy to the cloud will be given in the next section). In the remaining part of this section, we will describe how pseudocost branching can be modified to exploit cloud information.

Pseudocost branching consists mainly in two operations: (i) updating the pseudocosts after an actual branching has been performed and the LP relaxations of the child nodes have been solved and (ii) computing the score of a variable using the current pseudocosts when deciding for a branching candidate. When updating the pseudocosts, the objective gains ς_j^+ and ς_j^- per unit change in variable x_j are computed, that is:

$$\varsigma_j^+ = \frac{\Delta^{\uparrow}}{\lceil x_j^\star \rceil - x_j^\star} \quad \text{and} \quad \varsigma_j^- = \frac{\Delta^{\downarrow}}{x_j^\star - \lfloor x_j^\star \rfloor} \tag{1}$$

where $\Delta^{\uparrow}$ and $\Delta^{\downarrow}$ are the differences between the optimal LP objectives of the corresponding child nodes and the current LP value. These gains are then used to update the current pseudocosts Ψ_j^+ and Ψ_j^- which are the averages of the objective gains (per unit step length) that have been observed for that particular variable so far. The thin line in Figure 1(a) illustrates the operation. These estimation formulas are based on the assumption that the objective increases linearly in both directions (hence the resulting triangle). This, however, may be a too crude approximation of the real shape of the projection on the split domain of x_j. In the case of dual degeneracy, there might be many optimal LP solutions with different values for x_j. Which of these values $x_j^\star$ takes is more or less arbitrary, but crucial for the current – and by that also for future – branching decisions.

Using interval I_j on the other hand it is possible to replace this approximation with a more precise model (thick line in Figure 1(a)). The corresponding way to compute gains is then:

$$\tilde{\varsigma}_j^+ = \frac{\Delta^{\uparrow}}{\lceil x_j^\star \rceil - u_j} \quad \text{and} \quad \tilde{\varsigma}_j^- = \frac{\Delta^{\downarrow}}{l_j - \lfloor x_j^\star \rfloor} \tag{2}$$

Where the values for ς^+ and ς^- may vary by chance, $\tilde{\varsigma}^+$ and $\tilde{\varsigma}^-$ will be constant, when the set of all corners of the optimal face is used as a cloud.

As far as the computation of the score s_j is concerned, the standard formulas to predict the objective gains when branching on variable x_j are

$$\Delta_j^+ = \Psi_j^+(\lceil x_j^\star \rceil - x_j^\star) \quad \text{and} \quad \Delta_j^- = \Psi_j^-(x_j^\star - \lfloor x_j^\star \rfloor) \tag{3}$$

Again, the underlying linear model may give a too optimistic estimate on the dual bound improvements. A more accurate estimate exploiting interval I_j can be obtained as:

$$\tilde{\Delta}_j^+ = \Psi_j^+(\lceil x_j^\star \rceil - l_j) \quad \text{and} \quad \tilde{\Delta}_j^- = \Psi_j^-(u_j - \lfloor x_j^\star \rfloor) \tag{4}$$

A graphical representation is depicted in Figure 1(b). Furthermore, the following observation holds:

Lemma 2.1. *Let $x^\star$ be an optimal solution of the LP relaxation at a given branch-and-bound node and $\lfloor x_j^\star \rfloor \leq l_j \leq x_j^\star \leq u_j \leq \lceil x_j^\star \rceil$. Then*

1. *for fixed $\Delta^\uparrow$ and $\Delta^\downarrow$, it holds that $\tilde{\varsigma}_j^+ \geq \varsigma_j^+$ and $\tilde{\varsigma}_j^- \geq \varsigma_j^-$, respectively;*
2. *for fixed Ψ_j^+ and Ψ_j^-, it holds that $\tilde{\Delta}_j^+ \leq \Delta_j^+$ and $\tilde{\Delta}_j^- \leq \Delta_j^-$, respectively.*

Proof. Follows directly from Equations (1)–(4). □

Thus, under the same preconditions, the standard pseudocosts will be an underestimation of the pseudocosts based on the cloud intervals, whereas the objective gain, on which the branching decision is made, will be an overestimation. Of course these quantities interact directly which each other: as soon as one of it gets altered, this will have an impact on all upcoming branching decisions and pseudocost computations. The effects of continuous over- and underestimation will amplify each other. The hope is that cloud branching helps to make better, more reliable predictions and thereby leads to better branching decisions.

3 Full strong branching with the cloud

In the present section we detail the extension of a full strong branching strategy to the cloud. The first problem is again how to generate a cloud of optimal LP solutions C. Following some preliminary computational results, we opted for a feasibility pump like objective function, minimizing the distance to the nearest integral point. More precisely, given a fractional solution $x^\star$, we define the objective function coefficient c_j of variable x_j as

$$c_j = \begin{cases} 1 & \text{if} \quad 0 < f_j < 0.5 \\ -1 & \text{if} \quad 0.5 \leq f_j < 1 \\ 0 & \text{otherwise} \end{cases}$$

where $f_j = x_j^\star - \lfloor x_j^\star \rfloor$ is the fractional part of $x_j^\star$. Using the primal simplex, we re-solve the LP (fixed to the optimal face) with this new objective function. We update the interval bound vectors l and u, and iterate, using the new optimum as $x^\star$. If, at a given iteration, the update did not yield a new integral interval bound, we stop.

As far as the three-way partition (F_2, F_1, F_0) is concerned, we perform full strong branching on all variables in the set F_2. If we can find even one variable in this set with a strictly improved dual bound in both child nodes, then we stop and pick the best variable within this set, completely

ignoring sets F_1 and F_0. In state-of-the-art solvers such as Cplex or SCIP the score of a variable is computed as the *product* of the objective gains in both directions (maybe using a minimum value of some epsilon close to zero for each factor). By this, the score of all variables in $F_1 \cap F_0$ will be (nearly) zero and therefore none of them will have maximum score.

Note that in this case cloud information is used essentially to filter out variables and solve a smaller number of LPs. If no such variable is found, different strategies can be devised, depending on how we deal with the remaining variables. One option is to proceed with performing strong branching on the variables in set F_1, but solving only one LP per variable (because by definition we already know that in one direction the dual bound change is zero). Note that variables in F_1 are not necessarily a subset of the fractional variables in $x^\star$: as such, while we may still have some speedup because we only solve one LP per variable, the number of variables may indeed be higher than what standard full strong branching would have done. If we can find at least one variable in $F_2 \cap F_1$ with a strictly improved dual bound in one direction, then we can stop and ignore set F_0 for the same reason as before. If this is not the case, then we know that for all variables in $F(C)$ no improvement can be obtained in any child node as far as the dual bound is concerned, and so the branching variable should be chosen with some other criterion.

Another, less computationally expensive, option is to always ignore variables in F_1 and stick to the variables in F_2. Apart from the obvious computational savings, this choice can be justified by the following argument: if there is a variable in F_2 with a strictly improved dual bound in both children, we will not consider $F_1 \cap F_0$ anyway. If there is none, this proves that the global dual bound will not improve independent of the branching decision: at least one of the two children will have the same dual bound as the current node. Therefore, we take the current set of points C as evidence that variables in F_2 are less likely to become integral than variables in F_1, and so should be given precedence as branching candidates.

Note that using additional points to filter out strong branching candidates is similar in spirit to the strategy called *nonchimerical branching* proposed in [8], where the optimal solutions of the strong branching LPs (which might have a different objective function value) were used for this purpose. The two strategies have complementary strengths: nonchimerical branching does not need to solve any additional LP w.r.t. strong branching, but needs the strong branching LPs to be solved to optimality, because of the usage of the dual simplex. Cloud branching on the other hand needs additional LPs, but these are in principle simpler (we are fixed to the optimal face), need not be solved to optimality (primal simplex is used), and do not impose any requirements to the solution of the final strong branching LPs. As such, the two techniques can be easily combined together and might synergize. Moreover, cloud branching can be used independent of strong branching, as argued in Section 2.

4 Computational experiments

For our computational experiments, we used SCIP 3.0.0.1 [22] compiled with SoPlex 1.7.0 [23] as LP solver. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSuse 12.1 with a gcc 4.6.2 compiler. Hyperthreading and Turboboost were disabled. We ran only one job per node to reduce random noise in the measured running time that might be caused by cache-misses if multiple processes share common resources.

We used two test sets of general, publically available MIP instances: the COR@L test set [24], which mainly contains instances that users worldwide submitted to the NEOS server [25] and the MMM test set which contains all instances from MIPLIB3.0 [26], MIPLIB2003 [17], and MIPLIB2010 [16]. We compare the performance of SCIP when using full strong branching versus a

Table 1: comparison of Cloud branching and full strong branching on MMM and COR@L instances, averages of success rate, cloud points, saved LPs per node, and rate of saved LPs; shifted geometric means of branch-and-bound nodes and running time in seconds

	cloud statistics				SCIP cloud branch		SCIP strong branch	
Test set	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
MMM	12.2	2.19	74.34	21.7	661	68.2	691	72.0
COR@L	40.8	2.71	70.97	51.8	569	118.3	593	157.3

cloud branching version of full strong branching as described in the previous section. In particular, we compare to the cloud branching variant that only considers variables in F_2 as possible branching candidates. Since we want to explicitly measure the impact of using the cloud for variable selection, we did not exploit the alternative LP optima by any other means, e.g. for cutting plane generation, primal heuristics, reduced cost domain propagation, etc. Results by Achterberg [21] indicate that this would be likely to give further improvements on the overall performance. Further, we used the default implementation of full strong branching in SCIP, which does not employ the methods suggested in [8] (yet). We expect that nonchimerical branching and cloud branching will complement each other nicely, however, this is left for future implementation and experiments. We used a time limit of one hour per instance. All other parameters were left at their default values.

For the MMM test set both, SCIP with cloud branching and with full strong branching, both solved the same number of instance; for the COR@L test set, one more instances was solved within the time limit when using cloud branching. Tables 2 and 3 in the Appendix show results for all instances which both variants could solve within the time limit, excluding those which were directly solved at the root node (hence no branching was performed). This leaves 68 instances for MMM and 104 instances for COR@L. Column "%Succ" shows the ratio of nodes on which cloud branching was run successfully, hence at least one additional cloud point was used. Considering those nodes, columns "Pts" and "LPs" depict of how many points the cloud consisted on average and how many strong branching LPs were saved on average per node, i.e., how many integral interval bounds could be found. The Column "%Sav" shows how many percent of all strong branching LPs could be saved for that instance. When the success rate is zero, these three columns show a dash. For both branching variants, "Nodes" and "Time" give the number of branch-and-bound nodes and the computation time needed to prove optimality.

Table 1 shows aggregated results. It gives averages over the corresponding numbers (the success rates, the used points, the saved LPs per node and the percentage of overall saved LPs) from Tables 2 and 3. Shifted geometric means are shown for the number of branch-and-bound nodes and the computation times, which are absolute performance measures. The shifted geometric mean of values $t_1, \ldots, t_n$ with shift s is defined as $\sqrt[n]{\prod(t_i + s)} - s$. We use a shift of $s = 10$ for time and $s = 100$ for nodes in order to reduce the effect of very easy instances in the mean values. Further, using a *geometric* mean prevents hard instances at or close to the time limit from having a huge impact on the measures. Thus, the shifted geometric mean has the advantage that it reduces the influence of outliers in both directions.

The results for the MMM test set show a slight improvement of 6% w.r.t. mean running time and 5% w.r.t. the mean number of nodes when using cloud branching. For COR@L, the mean number of nodes again is slightly larger, about 4%, when using full strong branching instead of cloud branching. The result when comparing computation times is much more explicit: the shifted geometric means differ by about 33%. As can be seen in Table 1, the success rate of cloud

branching is much better on the COR@L test set than it is on MMM; and even further, on the successful instances, the average ratio of saved LPs is much larger. Taking these observations together explains why the improvement is much more significant for the COR@L test set.

MIP solvers are known to be prone for an effect called performance variability. Loosely speaking, this term comprises unexpected changes in performance which are triggered by seemingly performance-neutral changes in the environment or the input format. Besides others, peformance variability is caused by imperfect tie breaking [16]. This results in small numerical differences caused by the use of floating point arithmetics which may lead to different decisions being taken during the solution process. A branch-and-bound search often amplifies these effects, which can be similarly observed for all major MIP (and also other optimization) softwares. As a consequence, small changes in performance might in fact be random noise rather than a real improvement or deterioration. This can, e.g., be seen for instance `cap6000` from MMM: Although cloud branching was never successful, the number of branch-and-bound nodes alters.[1]. Then again, improvements brought by single components of a MIP solver typically lie in the range of 5–10%, see, e.g., [4]. In addition, even if MIP solvers did not exhibit performance variability, we would have the issue of assessing whether the measured difference in performance is statistically significant, a problem common to all empirical studies.

We performed two additional experiments to validate our computational results. First, we ran identical tests on four more copies of the test sets, with perturbed models that were generated by permuting columns and rows of the original problem formulation. This has been introduced in [16] as a good variability generator that affects all types of problems and all components of a typical MIP solver. Another benefit of this experiment is that it counters overtuning since the evaluation testbed is no longer the same as the development test bed.

As can be expected, the results differ in detail from the default permutation run. For MMM, the improvements w.r.t. computation time were 3%, 4%, 4% and 7%, and w.r.t. branch-and-bound nodes -3%, 0%, 1% and 2%. On COR@L, the improvements w.r.t. time were 25%, 29%, 32%, and 42% and w.r.t. branch-and-bound nodes 3%, 5%, 8%, 14%. We conclude the cloud branching was faster in all five times two experiments (including the original ones) and also consistently reduced the number of branch-and-bound nodes on the COR@L test set. For MMM, it can be argued that the changes are performance neutral w.r.t. the number of branch-and-bound nodes.

As far as the statistical significance of these differences is concerned, we performed randomized tests [27] on the detailed results. Randomized tests are standard non-parametric statistics that do not make any assumptions on the underlying population distributions (assumptions are very likely to be violated in our computational settings) but are still as powerful as standard parametric tests. According to these tests, the performance difference, both w.r.t. time and nodes, measured on the MMM is *not* statistically significant. As far as COR@L is concerned, the difference in branch-and-bound nodes is again not significant, while the difference in running times is. Note that on heterogeneous test sets such as MMM and COR@L, it is rather difficult to pass statistical significance tests when testing single MIP solver components, because the improvements are almost always in the single digit range and standard test sets are relatively small. In other words, one method might indeed be better than the other, but not by enough to pass the statistical test. We also applied these randomized tests to the other four copies of the test sets, with consistent results.

Having a closer look at Tables 2 and 3, it can be seen that the success rate of cloud branching is negligible, i.e., close to zero, for a significantly higher ratio of the MMM test set than for

[1] This can be explained by the intermediate cloud LPs being solved – after this, the original LP basis gets installed again and a resolve without simplex iterations is performed. However, solution values, reduced costs etc. might be slightly different than before.

the COR@L test set. This is also reflected by the much smaller average success rate shown in Table 1. This partially explains why the differences on COR@L are much more significant than on MMM: there are simply more instances on which degenerate LP solutions are detected in the pump-reduce step of our algorithm. A reason for this might be that MIPLIB instances contain more industry-based models with real, perturbed data whereas COR@L has more combinatorial models which often contain symmetries and are prone for degeneracy.

Our interpretation of the given results therefore is that cloud branching does not hurt a test set where only few degeneracy is detected but is clearly superior on a test set which contains many highly degenerated problems, at least time-wise.

5 Conclusion and outlook

In this paper, we introduced branching strategies for mixed-integer programs that exploit the knowledge of a cloud of alternative optimal LP solutions. We discussed extensions of full strong branching and pseudocost branching that incorporate this idea. Our computational experiments showed that a version of full strong branching that uses cloud intervals is about 30% faster than default full strong branching on a standard test set with high dual degeneracy. Even the mean number of branch-and-bound nodes could be reduced, though not significantly.

The presented preliminary results are very encouraging for further research on cloud branching. A natural next step is to implement the described modifications on pseudocost branching and a development of hybrid strategies such as reliability branching that make use of the cloud. In this paper, we used multiple optima from a single relaxation as cloud set. In particular in the context of MINLP, employing optima from *multiple, alternative relaxations* seems promising. From the implementation point of view, it could be further exploited that the cloud LPs are solved by the primal simplex algorithm, hence also intermediate, suboptimal solutions will be feasible and could be used as cloud points. Finally, two other improvements of strong branching were suggested recently: nonchimerical branching [8] and a work of Gamrath [28] on using domain propagation in strong branching. It will be interesting to see how these ideas combine and whether it will even be possible to make full strong branching competitive to state-of-the-art hybrid branching rules w.r.t. mean running time.

Acknowledgements

Many thanks to Gerald Gamrath and four anonymous reviewers for their constructive criticism.

References

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

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

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

[4] Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007) http://vs24.kobv.de/opus4-zib/frontdoor/index/index/docId/1018.

[5] Bixby, R., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice – closing the gap. In Powell, M., Scholtes, S., eds.: Systems Modelling and Optimization: Methods, Theory, and Applications. Kluwer Academic Publisher (2000) 19–49

[6] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: Finding cuts in the TSP (A preliminary report). Technical Report 95-05, DIMACS (1995)

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

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

[9] 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

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

[11] Karamanov, M., Cornuéjols, G.: Branching on general disjunctions. Mathematical Programming **128**(1-2) (2011) 403–436

[12] Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Proc. of CP, Autriche, Springer (1997) 342–356

[13] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th annual Design Automation Conference (DAC '01). (2001) 530–535 doi:10.1145/378239.379017.

[14] Kılınç Karzan, F., Nemhauser, G.L., Savelsbergh, M.W.P.: Information-based branching schemes for binary linear mixed-integer programs. Mathematical Programming Computation **1**(4) (2009) 249–293

[15] Fischetti, M., Monaci, M.: Backdoor branching. In Günlük, O., Woeginger, G.J., eds.: Integer Programming and Combinatoral Optimization - 15th International Conference, IPCO 2011, New York, NY, USA, June 15-17, 2011. Proceedings. Volume 6655 of Lecture Notes in Computer Science., Springer (2011) 183–191

[16] 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 - Mixed Integer Programming Library version 5. Mathematical Programming Computation **3** (2011) 103–163 http://miplib.zib.de.

[17] Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters **34**(4) (2006) 1–12 http://miplib.zib.de/miplib2003/.

[18] Zamora, J.M., Grossmann, I.E.: A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. Journal of Global Optimization **14** (1999) 217–249 doi:10.1023/A:1008312714792.

[19] Caprara, A., Locatelli, M.: Global optimization problems and domain reduction strategies. Mathematical Programming **125** (2010) 123–137 doi:10.1007/s10107-008-0263-4.

[20] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming **104**(1) (2005) 91–104 doi:10.1007/s10107-004-0570-3.

[21] Achterberg, T.: LP basis selection and cutting planes. Presentation slides from MIP 2010 conference in Atlanta. http://www2.isye.gatech.edu/mip2010/program/program.pdf (2010)

[22] Achterberg, T.: SCIP: Solving Constraint Integer Programs. Mathematical Programming Computation **1**(1) (2009) 1–41 doi:10.1007/s12532-008-0001-1.

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

[24] COR@L: MIP Instances (2010) http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/.

[25] Czyzyk, J., Mesnier, M., Moré, J.: The NEOS server. Computational Science & Engineering, IEEE **5**(3) (1998) 68–75 http://www.neos-server.org/neos/.

[26] Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.: An updated mixed integer programming library: MIPLIB 3.0. Optima (58) (1998) 12–15 http://miplib.zib.de/miplib3/miplib.html.

[27] Cohen, P.R.: Empirical Methods for Artificial Intelligence. MIT Press (1995)

[28] Gamrath, G.: Improving strong branching by domain propagation. Accepted for publication in Proc. of CPAIOR 2013 (2013)

Appendix

Tables 2 and 3 show the detailed results for the computational evaluation given in Section 4. They report statistics on all instances from our two test sets MMM and COR@L which SCIP could solve to optimality in less than one hour for either strong branching variant, but needed more than one node in both cases.

Table 2: comparison of cloud branching and full strong branching on MMM instances, smaller (better) numbers are bold

		cloud statistics			SCIP cloud branch		SCIP strong branch	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
10teams	64.3	2.7	50.3	79.3	**129**	**105.3**	348	488.1
aflow30a	0.0	–	–	–	**166**	**19.6**	182	21.7
air04	91.3	2.0	9.1	32.3	**55**	2087.9	57	**2074.6**
air05	46.8	2.0	3.0	3.4	166	1597.0	**153**	**1541.6**
ash608gpia-3col	100.0	4.3	2240.1	86.7	**5**	**1072.1**	9	2406.8
bell3a	0.0	–	–	–	**26 588**	6.6	26 590	**6.3**
bell5	0.2	2.0	2.0	0.6	**851**	0.7	865	0.7
bienst2	25.5	2.4	6.6	34.1	21 729	**1586.4**	**21 210**	1707.6
binkar10_1	4.4	2.0	4.8	3.3	**45 080**	**1715.7**	48 835	1744.9
blend2	9.3	2.0	2.0	5.4	**108**	0.8	110	**0.7**
cap6000	0.0	–	–	–	1 601	3.3	**1 545**	**3.1**
dcmulti	0.0	–	–	–	120	**2.3**	120	2.5
dfn-gwin-UUM	0.0	–	–	–	**5 897**	435.1	5 918	**431.6**
eil33-2	0.0	–	–	–	484	739.8	**480**	**734.2**
enigma	5.2	2.0	9.2	14.6	**27**	**0.5**	249	0.6

Table 2 continued

instance	cloud statistics %Succ	Pts	LPs	%Sav	SCIP cloud branch Nodes	Time (s)	SCIP strong branch Nodes	Time (s)
fiber	0.0	–	–	–	16	**1.1**	16	1.3
fixnet6	0.0	–	–	–	9	2.3	9	**2.2**
flugpl	0.0	–	–	–	134	0.5	134	0.5
gesa2-o	0.0	–	–	–	5	**1.4**	5	1.5
gesa2	0.0	–	–	–	3	1.0	3	1.0
gesa3	0.0	–	–	–	**11**	**1.4**	15	1.5
gesa3_o	0.0	–	–	–	9	**1.5**	9	1.7
khb05250	0.0	–	–	–	4	0.5	4	0.5
l152lav	3.9	2.0	6.7	3.5	**53**	**4.7**	65	7.1
lseu	15.4	2.1	3.4	12.7	**364**	0.7	382	**0.5**
map18	0.0	–	–	–	103	**1454.7**	**101**	1701.6
map20	0.0	–	–	–	**87**	**1129.0**	91	1384.7
mas74	0.0	–	–	–	574 769	1389.5	574 769	**1321.8**
mas76	0.0	–	–	–	**81 106**	123.7	84 280	**123.0**
mik-250-1-100-1	0.0	–	–	–	**290 018**	1681.4	290 038	**1628.3**
mine-166-5	0.0	–	–	–	2 001	**142.6**	**1 994**	155.6
misc03	11.7	2.3	10.4	25.4	68	**1.4**	**65**	1.5
misc06	5.9	2.0	4.0	6.7	13	0.8	13	**0.6**
misc07	13.2	2.1	7.1	23.5	**2 300**	62.9	2 365	**57.9**
mod008	0.0	–	–	–	**104**	0.8	111	0.8
mod010	0.0	–	–	–	10	**1.0**	10	1.1
mod011	0.0	–	–	–	321	**989.9**	321	1069.9
modglob	0.0	–	–	–	299	2.9	299	**2.8**
neos-1109824	51.9	2.3	26.0	68.4	1 246	473.0	**1 023**	**390.9**
neos-1396125	69.4	2.2	8.3	55.1	**2 714**	**2355.7**	2 976	2653.2
neos-476283	0.0	–	–	–	445	887.5	**323**	**680.2**
neos-686190	3.7	2.0	9.7	4.8	**1 451**	**540.6**	2 085	774.5
noswot	86.9	2.4	16.5	74.2	337 012	957.6	**210 056**	**869.0**
ns1766074	0.0	2.1	5.5	0.1	**241 641**	492.3	241 801	**470.2**
nw04	0.0	–	–	–	5	54.8	5	**46.4**
p0033	0.0	–	–	–	5	0.5	5	0.5
p0201	47.0	2.3	23.2	64.6	52	**2.6**	**51**	3.0
p0282	0.0	–	–	–	3	0.5	3	0.5
p0548	0.0	–	–	–	5	0.5	5	0.5
p2756	2.6	2.0	4.0	2.5	**82**	**1.9**	146	2.0
pk1	0.1	2.8	16.4	0.6	**76 569**	257.8	77 616	**233.1**
pp08a	0.0	–	–	–	300	3.7	**251**	**3.0**
pp08aCUTS	0.2	2.0	2.0	0.1	**213**	**3.2**	284	4.2
qiu	10.3	2.1	10.3	17.9	**14 858**	**1515.7**	16 290	1895.5
qnet1	17.6	2.0	6.0	4.4	5	3.8	5	**3.4**
qnet1_o	20.0	2.0	3.8	2.8	22	**9.2**	22	10.3
ran16x16	4.4	2.0	2.3	2.6	28 684	1184.3	**27 051**	**964.4**
reblock67	0.0	–	–	–	**28 052**	**1528.8**	33 290	1773.3
rentacar	27.3	2.0	3.3	22.2	**13**	**3.4**	14	3.5
rmatr100-p10	0.1	2.0	2.0	0.0	**163**	952.8	164	**950.2**
rmatr100-p5	0.0	–	–	–	33	1327.1	33	**1321.6**
rout	32.6	2.2	11.8	46.0	**1 561**	**79.3**	1 712	85.8
set1ch	0.0	–	–	–	**16**	**0.9**	17	1.0
sp98ir	2.1	2.0	3.5	1.3	**609**	**404.1**	876	507.4
stein27	29.1	2.3	6.2	22.9	787	2.2	**775**	**2.0**
stein45	20.7	2.1	5.8	11.2	**7 909**	**73.8**	8 446	77.3
tanglegram2	0.0	–	–	–	2	**27.3**	2	34.3
vpm2	9.4	2.0	2.2	3.4	**46**	1.3	48	1.3

Table 3: comparison of cloud branching and full strong branching on Cor@l instances, smaller (better) numbers are bold

	cloud statistics				cloud branching		full strong branching	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
22433	0.0	–	–	–	4	**1.1**	4	1.2
23588	0.2	2.0	14.0	0.3	**148**	**6.0**	174	6.3
aligninq	1.8	2.0	9.0	1.1	**24**	**20.6**	32	23.1
bc1	0.0	–	–	–	**604**	132.6	616	**124.0**
bc	0.0	–	–	–	1 985	2437.7	1 985	**2323.3**
bienst1	31.7	2.2	6.0	39.0	**2 712**	**151.1**	2 737	172.1
bienst2	25.5	2.4	6.6	34.1	21 729	**1587.7**	**21 210**	1703.3
binkar10_1	4.4	2.0	4.8	3.3	**45 080**	**1714.4**	48 835	1744.8
dano3_3	0.0	–	–	–	9	235.5	9	**153.3**
dano3_4	0.0	–	–	–	4	**176.7**	4	177.4
haprp	0.0	–	–	–	20 289	1696.4	**19 844**	**1629.9**
neos-1053591	94.5	2.3	8.7	70.0	**1 794**	**19.2**	46 259	367.4
neos-1109824	51.9	2.3	26.0	68.4	1 246	482.7	**1 023**	**390.7**
neos-1120495	38.7	2.2	19.4	49.7	102	18.7	**75**	**17.6**
neos-1122047	100.0	3.5	42.0	96.6	3	80.6	**2**	**34.1**
neos-1200887	86.2	2.5	13.8	63.5	**981**	**234.4**	1 465	381.7
neos-1211578	74.9	2.4	10.9	74.6	49 619	**315.6**	**32 225**	323.5
neos-1224597	98.6	6.7	631.1	95.0	**70**	**406.8**	80	864.5
neos-1228986	74.8	2.4	11.7	70.2	42 072	**358.7**	**39 690**	400.6
neos-1281048	91.8	5.8	133.8	88.1	**59**	**49.8**	80	170.7
neos-1337489	74.9	2.4	10.9	74.6	49 619	**312.0**	**32 225**	320.5
neos-1367061	0.0	–	–	–	16	**1601.4**	16	1683.2
neos-1396125	69.4	2.2	8.3	55.1	**2 714**	**2359.8**	2 976	2659.6
neos-1413153	81.8	2.9	292.5	88.6	**192**	**219.9**	462	2546.4
neos-1415183	90.9	2.6	252.4	85.7	**19**	**6.6**	56	43.6
neos-1420205	82.5	2.1	8.0	33.5	10 674	46.8	**7 840**	**45.5**
neos-1437164	74.7	2.2	11.0	47.1	80	**1.8**	**47**	1.9
neos-1440225	92.3	4.7	381.7	96.6	**6**	**11.0**	134	1387.4
neos-1440447	88.7	2.8	18.4	80.6	**7 676**	**207.5**	22 496	747.9
neos-1441553	78.0	2.3	26.0	58.1	**133**	**16.2**	215	86.7
neos-1445743	0.0	–	–	–	2	101.1	2	**64.4**
neos-1445755	5.0	2.0	4.0	16.7	3	75.0	3	**57.6**
neos-1445765	1.6	2.0	2.0	0.3	5	374.9	5	**236.2**
neos-1460265	99.8	4.0	216.2	80.2	6 997	1354.8	**1 125**	**540.5**
neos-1480121	23.0	2.0	3.5	25.0	**1 288**	**3.3**	1 961	4.0
neos-1489999	0.0	–	–	–	21	**28.6**	21	32.0
neos-476283	0.0	–	–	–	445	885.7	**323**	**687.4**
neos-480878	24.2	2.0	3.2	7.3	**2 803**	**230.9**	3 517	279.0
neos-494568	94.2	3.0	181.7	76.8	291	**398.2**	**285**	1082.3
neos-501474	48.2	2.0	4.0	46.9	158	1.3	**104**	**0.7**
neos-504674	50.5	2.0	5.9	16.6	1 256	**399.7**	**1 230**	426.9
neos-504815	35.6	2.1	6.1	16.5	510	**75.4**	**502**	83.3
neos-506422	16.2	2.1	3.7	20.9	1 451	540.7	**959**	**337.3**
neos-512201	48.7	2.0	6.0	15.5	665	175.7	**436**	**149.2**
neos-522351	0.0	–	–	–	3	1.1	3	**1.0**
neos-525149	55.3	3.0	88.7	45.8	**46**	**17.9**	187	193.4
neos-530627	0.0	–	–	–	2	0.5	2	0.5
neos-538867	72.8	3.0	21.4	76.2	6 697	318.1	**4 358**	**208.9**
neos-538916	77.5	3.2	23.9	77.4	4 642	371.5	**3 496**	**294.6**
neos-544324	99.9	2.0	15.2	92.3	7	301.4	7	**149.9**
neos-547911	90.0	2.1	10.8	85.1	30	244.3	30	**184.5**
neos-555694	71.3	2.6	98.6	69.9	**65**	**36.5**	177	301.7
neos-555771	92.7	2.4	107.7	80.2	**32**	**17.8**	70	72.4
neos-570431	71.0	2.0	8.1	59.4	**60**	**290.2**	76	314.7
neos-584851	47.0	2.1	20.0	60.3	56	**778.1**	**38**	840.5
neos-585192	0.0	–	–	–	**333**	**40.1**	345	40.6
neos-585467	1.2	2.0	12.0	1.4	**125**	**10.6**	133	10.7
neos-593853	0.0	–	–	–	**10 157**	**52.4**	12 204	56.0

Table 3 continued

	cloud statistics				cloud branching		full strong branching	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
neos-595905	0.0	–	–	–	**418**	**25.2**	473	29.5
neos-595925	0.0	–	–	–	**1 166**	**51.6**	1 189	51.8
neos-598183	0.0	–	–	–	488	**6.8**	**486**	7.1
neos-611838	0.0	–	–	–	193	**89.5**	193	94.0
neos-612125	0.0	–	–	–	92	**47.1**	92	50.3
neos-612143	0.0	–	–	–	130	**55.1**	**128**	59.7
neos-612162	0.0	–	–	–	**122**	**74.6**	126	80.3
neos-631694	93.9	2.9	56.8	49.5	**94**	**57.3**	101	93.6
neos-686190	3.7	2.0	9.7	4.8	**1 451**	**537.9**	2 085	776.4
neos-709469	12.5	2.3	22.4	58.6	1 608	3.5	**28**	**1.6**
neos-717614	0.0	–	–	–	**1 059**	65.9	1 061	**65.3**
neos-775946	95.4	2.8	93.4	81.3	**234**	**140.6**	413	343.6
neos-785899	93.0	2.8	94.2	77.7	**179**	**130.7**	266	247.6
neos-785914	83.8	3.4	135.0	92.0	109	**123.4**	**20**	296.6
neos-801834	0.0	–	–	–	11	841.5	11	**817.6**
neos-803219	0.1	2.0	2.0	0.0	**4 131**	72.8	4 231	**70.9**
neos-803220	0.0	–	–	–	18 713	179.3	**17 175**	**166.5**
neos-806323	28.1	2.0	2.7	11.0	**3 258**	**137.6**	3 645	142.6
neos-807639	4.1	2.0	2.6	3.1	1 130	18.6	**1 120**	**17.2**
neos-807705	20.3	2.0	2.2	6.3	2 373	88.3	**2 241**	**80.0**
neos-808072	72.1	2.3	32.3	51.7	**43**	**379.0**	90	1905.3
neos-810326	34.9	2.0	4.0	6.2	267	**2394.3**	**266**	2431.7
neos-820879	45.1	2.0	3.8	9.6	127	281.1	**114**	**210.3**
neos-825075	84.6	3.9	60.0	80.5	**18**	**3.0**	49	7.8
neos-839859	0.1	2.0	12.0	0.1	**1 084**	**773.1**	1 628	938.6
neos-862348	35.8	2.1	19.8	21.9	99	**33.9**	**70**	38.4
neos-863472	32.8	2.2	15.6	63.2	88 330	2330.9	**68 169**	**2264.0**
neos-880324	62.8	2.4	22.2	78.7	62	1.8	**15**	**1.0**
neos-892255	100.0	2.5	278.6	97.3	8	**720.1**	**5**	1590.8
neos-906865	0.0	–	–	–	7 079	462.4	**7 065**	**453.8**
neos-912015	93.2	5.1	130.8	94.2	791	473.3	**209**	**322.1**
neos-916173	0.0	–	–	–	1 497	**390.2**	**1 478**	392.0
neos-933550	83.3	8.4	638.8	96.6	**5**	**10.1**	25	58.2
neos-933815	47.7	2.0	5.7	32.3	61 797	801.5	**55 797**	**661.4**
neos-934531	99.3	3.4	89.8	96.1	**27**	**293.0**	51	1432.9
neos-941698	97.7	6.1	357.2	95.8	**19**	**14.2**	44	70.5
neos-942323	99.7	4.0	187.8	97.7	**189**	**64.0**	2 205	1240.4
neos-955215	68.4	2.1	9.1	53.0	7 574	61.3	**6 593**	**52.9**
neos-957270	83.1	2.8	159.9	89.0	**14**	471.3	17	**228.4**
nsa	0.0	–	–	–	258	**2.8**	258	3.0
nug08	0.0	–	–	–	3	24.9	3	**23.2**
prod1	0.0	2.0	8.0	0.0	4 053	33.8	**3 820**	**31.4**
prod2	0.0	–	–	–	**25 200**	361.6	25 227	**354.0**
qap10	33.3	2.0	2.0	40.0	2	177.3	2	**157.3**
sp98ir	2.1	2.0	3.5	1.3	**609**	**403.5**	876	503.2
Test3	0.0	–	–	–	10	8.0	10	8.0