Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

GERALD GAMRATH, ANNA MELCHIORI, TIMO BERTHOLD,
AMBROS M. GLEIXNER, DOMENICO SALVAGNIN

Branching on multi-aggregated variables

This paper is to appear in the Proceedings of the 12th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR 2015) held May 18-22, 2015, in Barcelona, Spain. The final publication is available at http://www.springerlink.com/.

ZIB Report 15-10 (February 2015)

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

Branching on multi-aggregated variables *

Gerald Gamrath[†], Anna Melchiori[‡], Timo Berthold[§],
Ambros M. Gleixner[¶], Domenico Salvagnin[‖]

Abstract

In mixed-integer programming, the branching rule is a key component to a fast convergence of the branch-and-bound algorithm. The most common strategy is to branch on simple disjunctions that split the domain of a single integer variable into two disjoint intervals. Multi-aggregation is a presolving step that replaces variables by an affine linear sum of other variables, thereby reducing the problem size. While this simplification typically improves the performance of MIP solvers, it also restricts the degree of freedom in variable-based branching rules.

We present a novel branching scheme that tries to overcome the above drawback by considering general disjunctions defined by multi-aggregated variables in addition to the standard disjunctions based on single variables. This natural idea results in a hybrid between variable- and constraint-based branching rules. Our implementation within the constraint integer programming framework SCIP incorporates this into a full strong branching rule and reduces the number of branch-and-bound nodes on a general test set of publicly available benchmark instances. For a specific class of problems, we show that the solving time decreases significantly.

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

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

1 Introduction

Since the invention of the branch-and-bound method for solving mixed-integer linear programming in the 1960s [1, 2], branching rules have been an important field of research, being one of its core components. For surveys, see [3, 4, 5]. In this paper we address branching strategies for mixed-integer linear programs (MIPs) of the form

$$\min\{c^T x : Ax \leq b, \ell \leq x \leq u, x_i \in \mathbb{Z} \text{ for all } i \in \mathcal{I}\} \tag{1}$$

with $c \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $\ell, u \in \bar{\mathbb{R}}^n$ where $\bar{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$, and $\mathcal{I} \subseteq \mathcal{N} = \{1, \ldots, n\}$ being the index set of integer variables. When removing the integrality restrictions, we obtain the *linear programming (LP) relaxation* of the problem.

*This article is to appear in the Proceedings of the 12th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR 2015) held May 18-22, 2015, in Barcelona, Spain.

†Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, gamrath@zib.de
‡University of Padua, Via Trieste 63, 35121 Padua, Italy, melch.anna@gmail.com
§Fair Isaac Europe Ltd, c/o ZIB, Takustr. 7, 14195 Berlin, Germany, timoberthold@fico.com
¶Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, gleixner@zib.de
‖DEI, University of Padua, Via Gradenigo 6/b, 35121 Padua, Italy, salvagni@dei.unipd.it

If the solution $\tilde{x}$ to the LP relaxation of (1) is fractional, i.e., if the index set $\tilde{\mathcal{I}} := \{i \in \mathcal{I} : \tilde{x}_i \notin \mathbb{Z}\}$ of *fractional variables* is not empty, the task of a branching rule is to split the problem into two or more subproblems. The strategy is typically to exclude the LP solution from all subproblems while keeping the feasible integer solutions, each being present in exactly one subproblem.

The choice of which subproblems to create is crucial for the performance of the algorithm. The approach most widely used by MIP solvers is to branch on *simple disjunctions*

$$x_k \leq \lfloor \tilde{x}_k \rfloor \quad \bigvee \quad x_k \geq \lceil \tilde{x}_k \rceil. \tag{2}$$

each side being enforced in one subproblem. As this procedure splits the domain of a single variable at a time, it is also called *branching on variables.* Alternatively, branching can be performed on a *general disjunction*

$$\pi^T x \leq \pi_0 \quad \bigvee \quad \pi^T x \geq \pi_0 + 1. \tag{3}$$

where $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}$, and $\pi_i = 0$ for all $i \notin \mathcal{I}$.

Branching on variables can be seen as the special case in which all considered disjunctions are of the form $(\pi, \pi_0) = (e_j, \lfloor \tilde{x}_j \rfloor)$, e_j being the j-th unit vector. Note that for branching on variables the set of *branching candidates* among which a branching rule chooses is usually the list of fractional variables $\tilde{\mathcal{I}}$. For branching on general disjunctions, the branching candidates consist of a potentially much larger list of disjunctions of form (3). Research on general branching disjunctions has largely been dedicated to determine a short list of promising candidates, see our literature overview in Sec. 2.

Another key component of state-of-the-art MIP solvers is *presolving.* It is applied before the branch-and-bound process and transforms a given MIP instance into a typically smaller instance with a tighter relaxation, which is hopefully easier to solve. These reductions can be based on pure feasibility arguments (keeping the set of feasible solutions unchanged) as well as optimality arguments (excluding also feasible solutions as long as one optimal solution remains).

Important presolving operations are fixings, aggregations, and multi-aggregations of variables. Here, *fixing* means that a variable gets permanently assigned to a constant value, *aggregation* means that a variable is replaced by (a constant value plus a scalar multiple of) another variable, and *multi-aggregation* means that a variable gets replaced by an affine linear combination of several variables. Hence, a *multi-aggregated* variable is a variable that is present in the original formulation, but is represented by an affine linear sum of variables in the presolved problem.

Contribution. The intuitive appeal of branching on general disjunctions is the increased degree of freedom that promises the creation of more balanced subproblems with tighter relaxations. This obvious advantage comes with the main challenge of determining promising candidate disjunctions. We address this difficulty by considering specifically the subset of disjunctions that are defined by the affine combinations stemming from multi-aggregations performed during the presolving stage. These disjunctions are naturally available in state-of-the-art MIP solvers at no cost and branching on them mimics branching on decision variables in the original model formulation.

Note that while the set of all general disjunctions of form (3) is exponentially large even when restricting π to $\{-1, 0, 1\}^n$, the set of multi-aggregated variables provides a list of potential candidates that is linear w.r.t. the size of the original model. Our experiments show that—in combination with standard single-variable disjunctions—this restriction yields not only a managable, but also computationally promising set of candidate disjunctions.

The remainder of the article is organized as follows. In Sec. 2, we give an overview of the literature on branching in MIP, with a particular focus on branching on general disjunctions. Sec. 3 introduces in more detail the concept of multi-aggregation, and Sec. 4 describes the idea of our new branching strategy and details about the implementation in the constraint integer programming framework SCIP [6, 7]. In Sec. 5 we presents our computational study and Sec. 6 contains our conclusions and gives an outlook on potential extensions of branching on multi-aggregated variables.

2 Related work

Various criteria for selecting fractional variables for branching on simple disjunctions have been presented in the literature. Most selection rules focus on the improvement in the dual bound that the branching restrictions produce in the created child nodes since this helps to tighten the global dual bound and prune nodes early. A fundamental strategy of this type is *strong branching* [8], which tentatively restricts the bound of a candidate variable and explicitly computes the resulting dual bound of the potential child node by solving the LP relaxation.

The *full strong branching* rule applies this at every node for each fractional variable. This typically leads to very small branch-and-bound trees, but on the other hand invests considerable effort in analyzing candidates. On average, this usually results in an overall performance deterioration w.r.t. computing time [5]. Nevertheless, the default branching rules in most state-of-the-art MIP solvers use some restricted form of strong branching and combine it with history information to reduce the computational effort for branching in later solving stages. Further strategies based on the same criteria can be found in [9, 4, 7, 10, 11, 12]. Recent research efforts on different criteria for variable-based branching rules include, e.g., [13, 14, 15, 16, 17, 18].

Branching on general disjunctions dates back to the 1980s [19], and has been addressed by various researchers in the last 15 years, see, e.g., [20, 21, 22, 23, 24]. The main challenge is to find a good class of general disjunctions that can lead to a better and more accurate tightening process of the feasible region, and consequently to a faster convergence of the dual bound to the optimal solution value, ideally without requiring a high computational effort for its generation and evaluation.

Owen and Mehrotra [20] present an algorithm that determines the branching disjunction via a neighborhood search heuristic. They prove that their algorithm is finite, if all variables have finite bounds and the size of the coefficients in the used disjunctions is bounded. As a consequence, they restrict their search to coefficients $\pi_i \in \{-1, 0, 1\}$. Combining this idea with [13], Mahmoud and Chinneck [24] choose a constraint that is active for the current LP optimum and construct a general disjunction with coefficients in $\{-1, 0, 1\}$ that is as perpendicular or as parallel as possible to the chosen active constraint.

Karamanov and Cornuéjols [22] consider disjunctions which correspond to Gomory mixed integer cuts (GMICs) [25]. They filter the GMICs to only keep the ten deepest cuts, and apply a strong-branching-like procedure on the corresponding candidate disjunctions. An extension of [22] is proposed by Cornuéjols et al. [23] who not only consider GMICs on tableau rows, but also on linear combinations of the tableau rows.

On the theoretical side, Mahajan and Ralphs proved that the problem of finding a general disjunction with maximal objective gain is $\mathcal{NP}$-hard [26]. Finally, Local Branching by Fischetti and Lodi [27] is a strategy to interleave variable-based branching with branching on general $\{-1, 0, 1\}$-disjunctions. These disjunctions measure the distance to the incumbent solution.

A typical result when branching on general disjunctions in MIP is that the generated branching trees are smaller on average, but the performance deteriorates w.r.t. running time. One major reason for this computational overhead is that the set of candidate disjunction for branching is much larger, so that a lot of time is spent determining the best one to choose at each node. However, this could in principle be overcome if we had more efficient (implicit) algorithms for evaluating the set of candidates, and it is of course not an issue when such set is still relatively small.

Another, more structural reason is that branching on variables changes a variable bound, which often fixes the variable to the other bound (in particular when branching on binary variables). This *decreases* the size of the LP relaxation for the subproblems by (at least) one column, whereas branching on general disjunctions potentially *increases* the LP's size by one row. This affects the simplex algorithm, which in most cases is the method of choice for solving the LP relaxations during LP-based branch-and-bound. Because the dimension of the basis matrix increases when adding a new row, most simplex implementations will have to recompute its factorization, causing computational overhead. In addition, many performance-relevant components of state-of-the-art MIP solvers such as domain propagation and conflict analysis are currently designed to benefit from branching on variables and become less effective when branching is performed on general disjunctions.

3 Multi-aggregations of variables

Before the branch-and-bound process is started, state-of-the-art MIP solvers perform a presolving phase during which they analyze the problem and remove redundancies, tighten the formulation, and collect information about the problem structure, see [28, 29, 30, 7, 31] for examples. This procedure is exact in the sense that each optimum of the simplified problem can be mapped to an optimal solution of the original problem.

The presolving technique which forms the basis of our newly developed branching rule is the multi-aggregation of variables. It reduces the number of variables by

1. detecting that—in at least one optimal solution—variable x_k equals an affine linear combination of other variables, i.e.,

$$x_k = \sum_{j \in \mathcal{S}_k} \alpha_j^k x_j + \beta^k, \tag{4}$$

   with $\mathcal{S}_k \subseteq \mathcal{N}$, $k \notin \mathcal{S}_k$,

2. replacing every occurrence of x_k in constraints and objective function by the right-hand side in (4), and

3. enforcing the bounds on x_k—if finite—by adding the new constraint

$$\ell_k \leq \sum_{j \in \mathcal{S}_k} \alpha_j^k x_j + \beta^k \leq u_k. \tag{5}$$

Equation (4) may either be explicitly present as one of the problem constraints[1] or implied by a combination of constraints and optimality conditions. An example for the latter is the case when x_k appears in exactly one constraint and its objective function coefficient ensures that this constraint will be fulfilled with equality in an optimal solution. The constraint integer programming

[1] Although in (1) we have formulated MIPs in terms of inequalities, this also includes equality constraints formulated via two inequalities.

framework SCIP, which we use for our computational experiments, has five different presolving operations in which multi-aggregation is performed.

After this step, one of the constraints implying (4) usually becomes void or is modified to enforce (5). If x_k is an integer variable, multi-aggregations are only performed if the integrality is enforced by the multi-aggregation. This holds, e.g., if (4) is an integer combination of integer variables, i.e., $\mathcal{S}_k \subseteq \mathcal{I}$, $\alpha_j^k \in \mathbb{Z}$ for all $j \in \mathcal{S}_k$, and $\beta^k \in \mathbb{Z}$.

In order to avoid a deterioration of performance and potential numerical problems during LP solving, it is crucial to safe-guard against fill-in in the constraint matrix. This can be done a priori by comparing the number of non-zeros that would be removed to the number of non-zeros that would get introduced in the constraint matrix, the latter of which can be bounded from above by the cardinality of $\mathcal{S}$ times the number of occurences of x_k.

To the best of our knowledge, all state-of-the-art MIP solvers use some form of multi-aggregation. For a test set of general MIP instances consisting of the last three MIPLIB [32, 33, 34] benchmark sets, the performance of SCIP is deteriorated by 3 % on average when disabling multi-aggregation. Taking into account that multi-aggregations are performed for no more than 15 % of the instances in this test set, this shows that multi-aggregations significantly improve the performance of MIP solvers when applicable.

In the following, we call a variable *inactive*, if presolving removed it from the problem. This includes variables which are already fixed to some value as well as aggregated and multi-aggregated variables. All other variables are called *active*. During the subsequent solving process, inactive variables are disregarded since their solution value is uniquely defined by the value of the active variables. In the remainder of this article, a MIP of form (1) always refers to the presolved problem containing only active variables. When referencing the original problem, we are using the following notation: the index sets of original and corresponding integer variables are denoted by $\mathcal{N}'$ and $\mathcal{I}'$, respectively. Original variables are written as x'_i and the variable on the left-hand side of a multi-aggregation (4) is an original variable x'_k, while all variables on the right-hand side are active variables x_j.[2]

4 Branching on multi-aggregated variables

Simple aggregations of form $x'_k = \alpha_j^k x_j + \beta^k$ performed during presolving do not restrict the choices of variable-based branching rules since branching on the subsequently inactive variable x'_k remains implicitly possible by branching on x_j. In contrast, branching on multi-aggregated variables cannot be realized via branching on active variables. We are not aware of any study that has investigated the effect of multi-aggregation on the performance of branching rules and note that this restriction may indeed have negative performance impact—especially since this effect is currently not considered during presolving.

Our new branching strategy considers the general disjunctions defined by all multi-aggregations (4) for which $k \in \mathcal{I}'$ but $\sum_{j \in \mathcal{S}_k} \alpha_j^k x_j + \beta^k$ evaluates to a fractional value in the current LP solution. In a strong branching fashion, we tentatively test which improvement in the local dual bounds we would obtain by adding one part of the corresponding general disjunction. We compare this to the improvements obtained by simple disjunctions on fractional active variables and choose the best among all branching disjunctions.

The motivation is twofold: first, to compensate for the above drawback, and second, to obtain a set of candidates for general branching disjunctions that is available at no cost in state-of-the-

[2] Note that nested multi-aggregations can be transferred into this form by (recursively) replacing inactive variables in the right-hand side of a multi-aggregation (4) by the corresponding constant or affine linear combination of variables.

art MIP solvers and computationally managable. As mentioned earlier, the set of all general disjunctions of form (3) is exponentially large even when restricting π to $\{-1,0,1\}^n$, in contrast to that, the number of multi-aggregations is linear w.r.t. the size of the original model.

In an LP-based branch-and-bound algorithm, the multi-aggregated branching rule is called whenever the optimal solution $\tilde{x}$ to the linear relaxation of the current node is fractional. Its procedure is outlined in Algorithm 1.

First, strong branching is performed on all elements in the set of fractional variables $\tilde{\mathcal{I}}$. For each candidate variable x_i, two auxiliary LPs are solved to compute dual bounds $\tilde{z}^-$ and $\tilde{z}^+$ for the potential child nodes. If both are larger than or equal the given upper bound (usually the objective function value of the incumbent solution), we can stop since no better solution can be found in the current subproblem and the node can be cut off. If only one of the two dual bounds is smaller than the upper bound, the corresponding bound change can directly be applied at the current problem, since the other child node does not contain an improving solution. If both dual bounds are smaller than the upper bound, the score for the candidate variable is computed and the simple disjunction $(e_i, \lfloor \tilde{x}_i \rfloor)$ corresponding to branching on this variable is stored as new best candidate if its score exceeds the best one found so far. The branching score used in SCIP is the product of the objective gains of the two child nodes, more specifically,

$$\text{score}(\tilde{z}^-, \tilde{z}^+) = \max\{\Delta_j^-, \epsilon\} \cdot \max\{\Delta_j^+, \epsilon\} \tag{6}$$

with $\epsilon = 10^{-6}$ and $\Delta_j^- = \tilde{z}^- - c^T\tilde{x}$ and $\Delta_j^+ = \tilde{z}^+ - c^T\tilde{x}$ being the objective gains in the child nodes when branching on x_j.

In the second step of the algorithm, full strong branching is performed on the general disjunctions defined by the multi-aggregated variables of the original problem. To this end, all integer multi-aggregated variables x'_k are taken into account for which the LP solution translates into a fractional solution $\tilde{x}'_k$. Analogously to the first step, two auxiliary LPs are solved with the potential branching disjunction added and the computed dual bounds are compared to the upper bound in order to prune the node or identify valid constraints. The score of the candidate disjunction is evaluated and compared to the best score found so far. If it is higher, the candidate disjunction is updated. Note that possible ties are broken in favor of candidate variables, since those are evaluated first and we are looking for strict improvements.

In the case that a valid bound change or inequality was found, we stop the branching rule, tighten the formulation, and return to the MIP solving process, which will continue by applying domain propagation, reoptimizing the LP, and calling the branching rule again if needed. After the evaluation of all candidate variables and disjunctions, and if no such valid bound or inequality was found, the best disjunction is returned and branching is performed on it.

5 Computational results

In the following, we present our experiments with branching on multi-aggregated variables. We used the academic constraint integer programming framework SCIP 3.1.0 [6, 7] with SoPlex 1.7.0.4 [35] as underlying LP solver and implemented Algorithm 1 as a branching rule plug-in. Our new method builds on the full strong branching scheme and extends it by choosing as the set of candidates to evaluate via strong branching not only candidate variables, but also candidate disjunctions given by multi-aggregations. Therefore, it is consequential to compare our strategy with the basic full strong branching rule of SCIP.

All results were obtained on a cluster of 3.2 GHz Intel Xeon X5672 CPUs with 48 GB main memory, running each job exclusively on one node. To keep the computation time under control, a time limit of 7200 seconds for each instance was imposed.

Algorithm 1: Multi-aggregated branching rule

input :
- a MIP of form (1),
- an optimal solution $\tilde{x}$ of the LP relaxation,
- an upper bound z^* on the objective value of solutions, and
- the index set $\mathcal{A}' \subseteq \mathcal{N}'$ of multi-aggregations of form (4), $x'_k = \sum_{j \in \mathcal{S}^k} \alpha_j^k x_j + \beta^k$, $k \in \mathcal{A}'$, $\mathcal{S}^k \subseteq \mathcal{N}$

output:
- a branching disjunction of form (3) given as $(\tilde{\pi}, \tilde{\pi}_0) \in \mathbb{Z}^n \times \mathbb{Z}$, or
- a valid inequality, or
- the conclusion that the current node can be pruned

begin
    `// 0. initialization`
    for $k \in \mathcal{A}' \cap \mathcal{I}'$ **do** `// compute LP values of multi-aggregated vars`
        $\tilde{x}'_k := \sum_{j \in \mathcal{S}^k} \alpha_j^k x_j + \beta^k$
    $\tilde{\mathcal{I}} := \{i \in \mathcal{I} : \tilde{x}_i \notin \mathbb{Z}\}$ `// single-variable candidates`
    $\tilde{\mathcal{A}} := \{k \in \mathcal{A}' \cap \mathcal{I}' : \tilde{x}'_k \notin \mathbb{Z}\}$ `// multi-aggregated candidates`
    $(\tilde{\pi}, \tilde{\pi}_0) := (0, 0)$ `// incumbent disjunction`
    $s_{(\tilde{\pi}, \tilde{\pi}_0)} := -\infty$ `// incumbent score`
    `// 1. full strong branching on simple disjunctions`
    for $i \in \tilde{\mathcal{I}}$ **do**
        $\tilde{z}^- \leftarrow \min\{c^T x : Ax \le b, \ell \le x \le u, x_i \le \lfloor \tilde{x}_i \rfloor\}$
        $\tilde{z}^+ \leftarrow \min\{c^T x : Ax \le b, \ell \le x \le u, x_i \ge \lfloor \tilde{x}_i \rfloor + 1\}$
        if $\min\{\tilde{z}^-, \tilde{z}^+\} \ge z^*$ **then return** *current node can be pruned*
        else if $\tilde{z}^- \ge z^*$ **then return** *valid inequality* $x_i \ge \lfloor \tilde{x}_i \rfloor + 1$
        else if $\tilde{z}^+ \ge z^*$ **then return** *valid inequality* $x_i \le \lfloor \tilde{x}_i \rfloor$
        else if $\mathrm{score}(\tilde{z}^-, \tilde{z}^+) > s_{(\tilde{\pi}, \tilde{\pi}_0)}$ **then**
            $(\tilde{\pi}, \tilde{\pi}_0) := (e_i, \lfloor \tilde{x}_i \rfloor)$
            $s_{(\tilde{\pi}, \tilde{\pi}_0)} := \mathrm{score}(\tilde{z}^-, \tilde{z}^+)$

    `// 2. full strong branching on multi-aggregated disjunctions`
    for $k \in \tilde{\mathcal{A}}$ **do**
        $\tilde{z}^- \leftarrow \min\{c^T x : Ax \le b, \ell \le x \le u, \sum_{j \in \mathcal{S}^k} \alpha_j^k x_j \le \lfloor \tilde{x}'_k \rfloor - \beta^k\}$
        $\tilde{z}^+ \leftarrow \min\{c^T x : Ax \le b, \ell \le x \le u, \sum_{j \in \mathcal{S}^k} \alpha_j^k x_j \ge \lfloor \tilde{x}'_k \rfloor - \beta^k + 1\}$
        if $\min\{\tilde{z}^-, \tilde{z}^+\} \ge z^*$ **then return** *current node can be pruned*
        else if $\tilde{z}^- \ge z^*$ **then return** $\sum_{j \in \mathcal{S}^k} \alpha_j^k x_j \ge \lfloor \tilde{x}'_k \rfloor - \beta^k + 1$ *valid*
        else if $\tilde{z}^+ \ge z^*$ **then return** $\sum_{j \in \mathcal{S}^k} \alpha_j^k x_j \le \lfloor \tilde{x}'_k \rfloor - \beta^k$ *valid*
        else if $\mathrm{score}(\tilde{z}^-, \tilde{z}^+) > s_{(\tilde{\pi}, \tilde{\pi}_0)}$ **then**
            $(\tilde{\pi}, \tilde{\pi}_0) := (\sum_{j \in \mathcal{S}^k} \alpha_j^k e_j, \lfloor \tilde{x}'_k \rfloor - \beta^k)$
            $s_{(\tilde{\pi}, \tilde{\pi}_0)} := \mathrm{score}(\tilde{z}^-, \tilde{z}^+)$

    return *branching disjunction* $(\tilde{\pi}, \tilde{\pi}_0)$
end

Settings. We compare the methods for two different settings. The first one, called *pure*, focuses on the main goal of a branching rule, namely proving the optimality of a solution. To this end, it disables cutting plane separation, primal heuristics, domain propagation, restarts, and conflict analysis. Additionally, we provide the optimal objective value as a cutoff bound at the beginning of the solving process. This is done in order to measure only the impact of branching without side-effects to and from other solver components. In particular, this reduces performance variability, cf. [34]. The second setting is called *default* and runs full strong branching (SB) and multi-aggregated branching (MA) in the SCIP default environment.

Instances. Our first experiments were performed on a test set of scheduling [36, 37] instances. More specifically, we were investigating *resource allocation and scheduling* problems, where jobs are assigned to machines, thereby minimizing the processing costs which depend on the machine on which a job is performed. Given sets $\mathcal{J}$ of jobs and $\mathcal{M}$ of machines, the capacity $C \in \mathbb{N}$ of the machines, and assignment cost $c_{j,m}$, resource allocation and scheduling can be expressed via the following MIP model [38]:

$$
\begin{aligned}
\min \sum_{m\in\mathcal{M}}\sum_{j\in\mathcal{J}} c_{j,m}x_{j,m} & & \\
\text{s.t.} \qquad \sum_{m\in\mathcal{M}} x_{j,m} &= 1 && \text{for all } j \in \mathcal{J}, \\
\sum_{t\in\mathcal{T}_{j,m}} x^t_{j,m} &= x_{j,m} && \text{for all } m \in \mathcal{M},\ j \in \mathcal{J}, \\
\sum_{j\in\mathcal{J}}\sum_{\bar{t}\in\mathcal{T}^t_{j,m}} c_j x^{\bar{t}}_{j,m} &\le C && \text{for all } m \in \mathcal{M},\ t \in \mathcal{T}, \\
x^t_{j,m} &\in \{0,1\} && \text{for all } m \in \mathcal{M},\ j \in \mathcal{J},\ t \in \mathcal{T}_{j,m}, \\
x_{j,m} &\in \{0,1\} && \text{for all } m \in \mathcal{M},\ j \in \mathcal{J}.
\end{aligned}
$$

The formulation uses binary variables $x_{j,m}$ and $x^t_{j,m}$, which represent the decision whether job $j \in \mathcal{J}$ is processed on machine $m \in \mathcal{M}$, and whether the processing of job $j \in \mathcal{J}$ on machine $m \in \mathcal{M}$ is started at time $t \in \mathcal{T}$, respectively. We use two subsets of the time periods: $\mathcal{T}_{j,m}$ which contains all time steps in which job j can start on machine m, and $\mathcal{T}^t_{j,m}$ which further restricts $\mathcal{T}_{j,m}$ to those starting times causing j to be (still) running in period t. When solving these instances, the $x_{j,m}$ variables are frequently multi-aggregated, which makes this problem an interesting test case for our first experiments.

We used a collection of 335 scheduling instances modeled this way in [38]. We excluded all instances that were solved either during presolving or at the root node. This left a total of 263 problem instances with the default setting and 276 instances with the pure setting.

In our second experiment, we used a test set of general MIP instances from different sources, including MIPLIB [32, 33, 34] and the Cor@l test set [39]. We removed some instances which to the best of our knowledge have never been solved so far and two numerically unstable instances giving slightly different results with both branching rules. Additionally, we restricted the test set to instances in which presolving performed multi-aggregations and removed instances which were solved during presolving or at the root node without branching. This gave us two test sets for the pure and default settings of 76 and 107 instances, respectively.

In the following, we present aggregated results over these test sets. Detailed computational results for each instance can be found in the appendix.

5.1 Results for scheduling instances

Table 1 compares the multi-aggregated branching strategy (MA) against the basic version of full strong branching (SB) available in SCIP with both pure and default settings, as indicated in the first column.

The remainder of the table is split into two parts: The four columns below the "scheduling test set" label display numbers about the performance on the complete scheduling test set. Column "size" shows the number of instances in the test set, "solved" gives the number of instances solved to proven optimality within the time limit of two hours. Column "faster" ("slower") show the number of instances that the MA strategy solved at least 10 % faster (slower) than standard full strong branching.

The right side of the table, labeled "all optimal", shows results for the subset of instances that both variants in the respective setting solved to optimality. Column "size" shows the number of instances in this subset, "nodes" the shifted geometric mean of the B&B nodes and "time (s)" the shifted geometric mean of the running time in seconds. We use shifts of 100 and 10 for the number of nodes and the solving time, respectively. For a discussion of the shifted geometric mean, we refer to [41, Appendix A3].

Let us first look at the results with the pure settings, which focus on the plain branch-and-bound performance. They are promising: 25 more instances (142 vs. 117) can be solved by branching on multi-aggregated variables compared to standard strong branching; this corresponds to an increase of more than 20 %. Furthermore, 100 instances are solved at least 10 % faster with the new method, compared to 13 which slow down by 10 % or more. This corresponds to 70 % of the instances being solved faster with branching on multi-aggregated variables. Looking at the instances that were solved to optimality by both variants, both the number of nodes and the requested time are reduced by a factor of two on average: 58 % less nodes are needed and 49 % less time.

When looking at the results with default settings, the effect is smaller, but still significant: the multi-aggregated branching strategy is able to solve 9 more instances to optimality, with 56 instances being solved faster and 31 slower. On instances that both variants solve to optimality, it needs 37 % less nodes and reduces the solving time by 17 %.

One might argue that the multi-aggregation of variables itself could have a negative impact on the performance for the scheduling instances as it restricts standard branching rules from branching on the $x_{j,m}$ variables which can be seen as first-level decisions. However, this is only partly true: When disabling multi-aggregations, the shifted geometric mean of the number of branch-and-bound nodes is indeed decreased by 14 % for the instances solved to optimality both with and without multi-aggregations. On the other hand, the average solving time is increased slightly by 2 %. This shows that the gains obtained by having more branching opportunities with multi-aggregation disabled are compensated by not being able to reduce the problem size so much and having more effort, e.g., in LP solving. Our proposed branching scheme takes the best of both variants, allowing the problem size reductions while still providing the potentially more powerful branching possibilities given by the multi-aggregated variables. This helps to improve both the number of nodes as well as the solving time significantly over the individual best of the two other variants.

Let us note that the positive effect of branching on multi-aggregated variables grows stronger the harder an instance is. This seems reasonable since the additional overhead might not pay off if a standard strong branching is able to solve an instance within a few nodes. When taking into account only instances which needed more than 100 seconds to solve by at least one setting, the reduction in the number of nodes and the solving time goes up to 42 % and 25 %, respectively.

This first computational experiment shows that branching on multi-aggregated variables can

Table 1: Results for scheduling instances with default and pure settings

	scheduling test set				**all optimal**		
setting	size	solved	faster	slower	size	nodes	time (s)
SB-pure	276	117			115	472	51.8
MA-pure	276	142	100	13	115	196	26.4
SB-default	263	126			122	349	84.6
MA-default	263	135	56	31	122	221	70.3

significantly improve the performance of SCIP compared to a pure variable-based branching rule: more instances are solved, with less enumeration, in shorter time. Note that in all cases the relative reduction in running time was smaller than the relative reduction in the number of branch-and-bound nodes, which is a typical result for branching strategies that involve general disjunctions (see Sec. 2).

In order to analyze the impact of the new branching rule in more detail, we collected some statistics during the execution of SCIP. On average over the test set, the number of integer multi-aggregations is only 5.7 % of the number of integer variables. Thus, the list of branching candidates is only slightly extended in most cases, which overcomes a typical issue for branching on general disjunctions. Interestingly, despite this relatively small number of multi-aggregations, 39 % of the branching decisions select a multi-aggregated disjunction for branching. Even more, in 85 % of the cases, the first branching on a multi-aggregated disjunction was performed at the root node.

Finally, each time we perform a multi-aggregated branching, we store the ratio of the gain that we would have obtained when branching on the best fractional variable compared to the gain obtained by branching on the current multi-aggregated variable. The gain is computed as the square root of the SCIP branching score value and thus measures the improvement in the score SCIP tries to maximize. On average over all calls where we branched on a multi-aggregated disjunction, the gain would have been reduced to 22 % by branching on the best variable instead.

5.2 Results for general MIP instances

The results for our collection of general MIP instances are presented in Table 2. The columns and rows show the same statistics as described in Sec. 5.1. We can see that on these instances, multi-aggregated branching is significantly slower and solved one less instance in both settings, compared to standard strong branching. With pure settings, the solving time increases by 25 % while the number of branch-and-bound nodes is decreased by 13 %. Compared to the scheduling instances, multi-aggregated variables are much less effective for branching. That the increased effort in strong branching outweighs the observed node reduction seems plausible. These results confirm our observation from the scheduling instances in the sense that the impact on the number of branch-an-bound nodes was better than the impact on the overall running time. For the scheduling instances, the additional candidates were structurally different and allowed different, higher-level decisions which had an enormous effect on the tree size that even allowed for a running time reduction. For standard MIPs, however, such a large effect is apparently obtained rarely, thus, the performance deteriorates on average. The picture looks even worse for the default settings. Here, the solving time increases by 26 % and the number of nodes now increases by 6 % as well.

Table 2: Results for general MIP instances with default and pure settings

	MIP test set				**all optimal**		
setting	size	solved	faster	slower	size	nodes	time (s)
SB-pure	76	33			32	983	150.9
MA-pure	76	32	0	26	32	852	188.9
SB-default	107	55			49	253	100.4
MA-default	107	57	1	33	49	269	126.3

Again, we collected statistics to analyze the impact of the multi-aggregated branching scheme. On average over the test set, the amount of integer multi-aggregations is almost twice as high as for the scheduling set, namely 14.4 % of the number of integer variables. However, multi-aggregated variables are selected less often (only for 1.84 % of the branchings) and consequently, also the first branching on a multi-aggregated disjunction was less often performed at the root node (only for 7.4 % of the instances for which multi-aggregated branching was performed). If a multi-aggregated disjunction was selected, selecting the best fractional variable instead would have decreased the gain by 31 % on average, compared to 78 % for the scheduling instances. This shows that multi-aggregated disjunctions play a smaller role for branching on this test set, but can still be used to improve the quality of branching disjunctions.

Even more surprising is the increase in the number of nodes, which can be explained, however, by the tailoring of many MIP solving algorithms towards variable-based branching. Domain propagation (or *node preprocessing*, see, e.g., [28] for MIP), for example, tries to tighten the local domains of variables by inspecting the constraints and current domains of other variables at the local subproblem. Tightening or fixing variables by branching is naturally beneficial for domain propagation, the impact of adding general disjunctions is rather opaque. Furthermore, techniques like primal heuristics, cutting plane separation, or conflict analysis profit from tightened variable bounds rather than from added general disjunctions. Since all these techniques help to reduce the size of the branch-and-bound tree, branching on general disjunctions with a high branching score can even increase the number of nodes, since as a side effect it makes the named procedure less effective.

We see our results for general MIPs as an important negative result that confirms previous observations by other authors that it is hard to find a branching rule on general disjunctions which is competitive on standard MIP benchmarks. Our results indicate that this holds even when restricting the selection to relatively few additional candidates that are naturally obtained from the problem structure. Finally, adapting procedures like primal heuristics or conflict analysis in such a way that they benefit from added constraints as much as from tightened or fixed variables might be a prerequisite to excel with constraint-based branching schemes in state-of-the-art MIP solvers.

6 Conclusions and outlook

In this paper, we presented a new branching rule which takes into account a specific type of general disjunctions. These general disjunctions, so-called multi-aggregations, are the affine linear sums of active variables in the presolved problem, which correspond to a decision variable in the original problem. We extended the full strong branching rule of SCIP by taking additionally into account all general disjunctions induced by multi-aggregations. On a set of scheduling instances, this significantly improved the performance of SCIP w.r.t. the tree size as well as the solving time and the number of solved instances.

We tested the same branching rule on standard MIP benchmark sets. The results were much less convincing, but a certain potential for branching on multi-aggregated variables was indicated by the observation that in a "pure" setting, it led to a reduction in the number of branch-and-bound nodes for general MIPs. However, before this potential can be harnessed, we conclude that many advanced solution techniques applied in state-of-the-art MIP solvers—domain propagation, conflict analysis, etc.—must be extended towards a more efficient handling of general disjunctions. An additional performance bias is the slow-down in current simplex implementations when adding and removing constraints. This bottleneck may be alleviated by the recent developments of [42, 43], which improve the underlying linear algebra routines such that the factorization of the basis matrix is preserved when adding new rows. We identify these points as important directions for future research.

Another field for future research would be to find criterions to assess the structure of multi-aggreagations and predict the power of the new scheme for the current instance in order to decide on whether to use it or not. A first basic variant of this would be to heuristically detect scheduling substructures and turn on the branching scheme for the involved multi-aggregations. Many improvements for MIP solving in recent years are based on specific structures, cf. [44, 45]. If this structure is detected, they lead to a significant improvement—as is the case for our scheme for scheduling problems—while the detection is typically fast enough that the performance on other problems is not deteriorated. Therefore, we are convinced that the new strategy can also improve the performance of MIP solvers for general MIP test sets.

The proposed strategy has been studied and implemented for the first time in the constraint integer programming framework SCIP. Since it proved its effectiveness for certain problem classes, it will be available in the next release of SCIP.

7 Acknowledgements

The work for this article has been conducted within the *Research Campus Modal* funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM). The authors would like to thank the three anonymous reviewers for helpful comments on the conference proceedings version of the paper.

References

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

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

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

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

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

[6] Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: A new approach to integrate CP and MIP. In Perron, L., Trick, M.A., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008. Volume 5015 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (May 2008) 6–20

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

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

[9] Gauthier, J.M., Ribière, G.: Experiments in mixed-integer linear programming using pseudo-costs. Mathematical Programming **12**(1) (1977) 26–47

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

[11] Berthold, T., Salvagnin, D.: Cloud branching. In Gomes, C., Sellmann, M., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 7874 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2013) 28–43

[12] Gamrath, G.: Improving strong branching by domain propagation. EURO Journal on Computational Optimization **2**(3) (2014) 99 – 122

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

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

[15] 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

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

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

[18] Pryor, J., Chinneck, J.W.: Faster integer-feasibility in mixed-integer linear programs by branching to force change. Computers & OR **38**(8) (2011) 1143–1152

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

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

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

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

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

[24] Mahmoud, H., Chinneck, J.W.: Achieving milp feasibility quickly using general disjunctions. Computers & OR **40**(8) (2013) 2094–2102

[25] Gomory, R.E.: An algorithm for the mixed integer problem. Technical report, RAND Corporation (1960)

[26] Mahajan, A., Ralphs, T.K.: On the complexity of selecting disjunctions in integer programming. SIAM Journal on Optimization **20**(5) (2010) 2181–2198

[27] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**(1-3) (2003) 23–47

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

[29] Brearley, A.L., Mitra, G., Williams, H.P.: Analysis of mathematical programming problems prior to applying the simplex algorithm. Mathematical Programming **8**(1) (1975) 54–83

[30] Bixby, R.E., Wagner, D.K.: A note on detecting simple redundancies in linear systems. Operation Research Letters **6**(1) (1987) 15–17

[31] Gamrath, G., Koch, T., Martin, A., Miltenberger, M., Weninger, D.: Progress in presolving for mixed integer programming. Technical Report 13-48, ZIB, Takustr.7, 14195 Berlin (2013)

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

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

[34] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation **3**(2) (2011) 103–163

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

[36] Baker, K.: Introduction to sequencing and scheduling. Wiley (1974)

[37] Baker, K.R., Trietsch, D.: Principles of Sequencing and Scheduling. Wiley (2009)

[38] Heinz, S., Ku, W.Y., Beck, J.C.: Recent improvements using constraint integer programming for resource allocation and scheduling. In Gomes, C., Sellmann, M., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 7874 of LNCS. Springer (2013) 12–27

[39] COR@L: MIP Instances (2014) `http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/`.

[40] Gamrath, G., Melchiori, A., Berthold, T., Gleixner, A.M., Salvagnin, D.: Branching on multi-aggregated variables. Technical Report 15-10, ZIB, Takustr. 7, 14195 Berlin (2015)

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

[42] Gleixner, A.M.: Factorization and update of a reduced basis matrix for the revised simplex method. ZIB-Report 12-36, Zuse Institute Berlin (October 2012)

[43] Wunderling, R.: The kernel simplex method. Talk at the 21st International Symposium on Mathematical Programming, Berlin, Germany (August 2012)

[44] Achterberg, T., Raack, C.: The MCF-separator: detecting and exploiting multi-commodity flow structures in MIPs. Mathematical Programming Computation **2**(2) (2010) 125–165

[45] Salvagnin, D.: Detecting and exploiting permutation structures in MIPs. In Simonis, H., ed.: Integration of AI and OR Techniques in Constraint Programming. Volume 8451 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2014) 29–44

A Detailed Computational Results

In this section, instance-wise results of the computational experiments described in Sec. 5 are presented. The following four tables list the results for the set of scheduling instances with default (Table 3) and pure settings (Table 4), as well as for the MIP instances with default (Table 5) and pure settings (Table 6).

Each table shows the number of branch-and-bound nodes and the total solution time for both standard strong branching (column "SB") and multi-aggregated branching ("MA"). If the time limit was reached for an instance, the number of nodes up to this point is listed and both nodes and time are preceded by a ">"-sign. Reductions in the node number or solving time of MA branching by a factor of at least two compared to SB are marked green, increases by a factor of at least two are marked red. At the end of each table, we list geometric, shifted geometric, and arithmetic means over the complete test set as well as the subset of instances solved to optimality by both variants.

Table 3: Detailed computational results on scheduling instances comparing full strong branching (SB) and branching on multi-aggregated variables (MA) with SCIP's default settings

	SB		MA	
	Nodes	Time	Nodes	Time
c10j2m1	2	2.4	2	2.3
c10j2m3	12	1.4	15	1.7
c10j2m4	23	0.6	25	0.6
c10j2m5	2	2.9	2	2.9
c12j2m2	19	8.0	19	7.4
c12j2m3	70	3.0	32	2.6
c12j2m4	47	2.8	47	2.9
c12j2m5	1 336	44.3	272	8.7
c12j3m2	44	2.1	15	2.0
c12j3m3	52	4.2	59	4.7
c12j3m4	2	1.2	2	1.2
c12j3m5	18	2.4	28	3.0
c12j4m1	49	0.9	44	1.0
c12j4m2	27	2.2	9	1.9
c14j2m1	143	14.0	113	10.4
c14j2m2	8 278	124.3	8 276	70.7
c14j2m4	186	5.6	113	5.6
c14j3m1	849	44.1	332 053	3256.3
c14j3m2	3	2.5	3	2.5
c14j3m4	4	2.8	4	2.7
c14j3m5	550	29.9	181	16.7
c14j4m1	69	14.0	43	12.0
c14j4m2	4	3.7	4	3.8
c14j4m3	400	26.9	148	16.3
c14j4m4	4	4.2	4	4.1
c14j4m5	54	4.4	30	4.1
c16j2m1	87	13.3	149	40.0
c16j2m2	362	22.0	294	29.3
c16j2m3	1 315	270.5	12 653	1191.3
c16j2m4	756	123.0	186	69.2
c16j2m5	2 723	207.5	6 405	642.0

	SB		MA	
	Nodes	Time	Nodes	Time
c16j3m1	4 168	636.8	4 674	826.0
c16j3m2	1 478	341.5	669	210.1
c16j3m3	1 567	136.9	1 676	110.5
c16j3m4	170	24.7	291	35.8
c16j3m5	278	39.6	46	9.5
c16j4m1	2	7.5	2	7.4
c16j4m2	904	55.7	282	28.7
c16j4m4	124	18.6	94	17.5
c16j4m5	191	31.1	66	18.2
c18j2m1	1 724	595.9	>42 133	>7200.0
c18j2m2	776	294.8	719	412.2
c18j2m3	842	153.5	>43 311	>7200.0
c18j2m4	3 229	442.3	657	206.1
c18j2m5	1 111	256.6	417	452.6
c18j3m1	2 841	1851.2	325	205.7
c18j3m2	1 141	290.7	218	260.3
c18j3m3	4 139	2182.3	31	58.2
c18j3m4	>2 550	>7200.0	>4 094	>7200.0
c18j3m5	1 276	559.9	856	598.3
c18j4m1	460	44.5	46	27.8
c18j4m2	859	368.2	505	389.6
c18j4m3	76	20.3	36	19.1
c18j4m4	920	267.0	496	146.8
c18j4m5	729	189.4	919	234.6
c20j2m1	>22 223	>7200.0	>15 939	>7200.0
c20j2m2	2 391	1296.4	153	135.1
c20j2m3	>18 414	>7200.0	>16 818	>7200.0
c20j2m4	>19 744	>7200.0	744	699.2
c20j2m5	14 533	2176.0	1 279	299.4
c20j3m1	10 577	3128.5	701	1234.5
c20j3m2	5 375	3359.7	>453 479	>7200.0
c20j3m3	2 545	1429.4	284	625.5
c20j3m4	>9 300	>7200.0	>19 578	>7200.0
c20j3m5	851	917.1	360	511.4
c20j4m1	1 647	479.6	205	132.9
c20j4m2	9 100	2191.2	1 118	509.7
c20j4m3	>4 928	>7200.0	1 324	2702.9
c20j4m4	11 231	736.7	137	82.6
c20j4m5	2 038	565.4	436	301.4
c22j2m1	>8 402	>7200.0	>8 511	>7200.0
c22j2m2	>8 048	>7200.0	>21 287	>7200.0
c22j2m3	>7 728	>7200.0	>7 045	>7200.0
c22j2m4	>7 767	>7200.0	>11 445	>7200.0
c22j2m5	>18 858	>7200.0	>19 014	>7200.0
c22j3m1	>3 161	>7200.0	1 517	5172.3
c22j3m2	>5 213	>7200.0	915	2425.7
c22j3m3	>6 892	>7200.0	>5 015	>7200.0
c22j3m4	2 401	3807.5	358	829.0
c22j3m5	>11 350	>7200.0	>21 621	>7200.0
c22j4m1	>18 818	>7200.0	>18 183	>7200.0
c22j4m2	>2 947	>7200.0	>6 622	>7200.0
c22j4m3	>22 315	>7200.0	18 800	5501.4
c22j4m4	2 239	575.5	746	392.0
c22j4m5	564	228.7	351	397.7
c24j2m1	>7 123	>7200.0	>7 012	>7200.0
c24j2m2	>6 148	>7200.0	>14 904	>7200.0
c24j2m3	>3 509	>7200.0	>6 818	>7200.0
c24j2m4	>741	>7200.0	207	2476.8
c24j2m5	>5 424	>7200.0	>3 475	>7200.0
c24j3m1	>3 159	>7200.0	>1 374	>7200.0

	SB		MA	
	Nodes	Time	Nodes	Time
c24j3m2	>2 678	>7200.0	>1 274	>7200.0
c24j3m3	>4 634	>7200.0	>16 898	>7200.0
c24j3m4	2 126	2913.4	1 587	3516.0
c24j3m5	>942	>7200.0	>292	>7200.0
c24j4m1	>2 128	>7200.0	>831	>7200.0
c24j4m2	>3 824	>7200.0	256	1068.3
c24j4m3	>9 086	>7200.0	>4 468	>7200.0
c24j4m4	>1 477	>7200.0	>884	>7200.0
c24j4m5	>12 410	>7200.0	>9 140	>7200.0
c26j2m1	>7 861	>7200.0	>2 415	>7200.0
c26j2m2	>4 045	>7200.0	>3 025	>7200.0
c26j2m3	>5 792	>7200.0	>2 263	>7200.0
c26j2m4	>4 816	>7200.0	>13 202	>7200.0
c26j2m5	>4 370	>7200.0	1 902	4991.4
c26j3m1	>3 749	>7200.0	>6 695	>7200.0
c26j3m2	>2 145	>7200.0	>4 316	>7200.0
c26j3m3	>1 327	>7200.0	>312	>7200.0
c26j3m4	>910	>7200.0	>1 039	>7200.0
c26j3m5	>2 237	>7200.0	>339	>7200.0
c26j4m1	>6 737	>7200.0	>1 574	>7200.0
c26j4m2	>1 213	>7200.0	>534	>7200.0
c26j4m3	>4 614	>7200.0	>2 096	>7200.0
c26j4m4	>1 472	>7200.0	>1 033	>7200.0
c26j4m5	>2 340	>7200.0	>680	>7200.0
c28j2m1	>4 284	>7200.0	>3 383	>7200.0
c28j2m2	>4 520	>7200.0	>10 543	>7200.0
c28j2m3	>8 027	>7200.0	>3 431	>7200.0
c28j2m4	>6 849	>7200.0	>1 877	>7200.0
c28j2m5	>4 145	>7200.0	>745	>7200.0
c28j3m1	>2 606	>7200.0	1 322	6764.1
c28j3m2	>4 129	>7200.0	9 623	5805.1
c28j3m3	>1 156	>7200.0	>283	>7200.0
c28j3m4	>546	>7200.0	>571	>7200.0
c28j3m5	>3 795	>7200.0	>3 190	>7200.0
c28j4m1	>3 664	>7200.0	>1 383	>7200.0
c28j4m2	>4 160	>7200.0	>1 127	>7200.0
c28j4m3	>1 562	>7200.0	>297	>7200.0
c28j4m4	>3 098	>7200.0	>2 203	>7200.0
c28j4m5	>5 623	>7200.0	>512	>7200.0
c30j2m1	>3 477	>7200.0	>812	>7200.0
c30j2m2	>3 783	>7200.0	1 898	6424.0
c30j2m3	>5 227	>7200.0	>1 687	>7200.0
c30j2m4	>2 931	>7200.0	>547	>7200.0
c30j2m5	>6 242	>7200.0	>637	>7200.0
c30j3m1	>675	>7200.0	>155	>7200.0
c30j3m2	>1 121	>7200.0	>665	>7200.0
c30j3m3	>2 309	>7200.0	>1 314	>7200.0
c30j3m4	>780	>7200.0	>145	>7200.0
c30j3m5	>768	>7200.0	>142	>7200.0
c30j4m1	>1 145	>7200.0	>134	>7200.0
c30j4m2	>920	>7200.0	>250	>7200.0
c30j4m3	>4 085	>7200.0	>1 264	>7200.0
c30j4m4	>2 479	>7200.0	>480	>7200.0
c30j4m5	>884	>7200.0	>464	>7200.0
c32j2m1	>2 626	>7200.0	>697	>7200.0
c32j2m2	>3 301	>7200.0	>1 190	>7200.0
c32j2m3	>1 954	>7200.0	>480	>7200.0
c32j2m4	>3 375	>7200.0	>968	>7200.0
c32j2m5	>1 779	>7200.0	>1 985	>7200.0
c32j3m1	>1 675	>7200.0	>651	>7200.0

	SB		MA	
	Nodes	Time	Nodes	Time
c32j3m2	>1 700	>7200.0	>845	>7200.0
c32j3m3	>290	>7200.0	>70	>7200.0
c32j3m4	>263	>7200.0	>89	>7200.0
c32j3m5	>899	>7200.0	>131	>7200.0
c32j4m1	>1 517	>7200.0	>171	>7200.0
c32j4m2	>376	>7200.0	>150	>7200.0
c32j4m3	>2 009	>7200.0	>1 289	>7200.0
c32j4m4	>1 699	>7200.0	>442	>7200.0
c32j4m5	>659	>7200.0	>150	>7200.0
c34j2m1	>2 737	>7200.0	>557	>7200.0
c34j2m2	>3 748	>7200.0	>1 178	>7200.0
c34j2m3	>1 017	>7200.0	>872	>7200.0
c34j2m4	>1 786	>7200.0	>461	>7200.0
c34j2m5	>2 641	>7200.0	>910	>7200.0
c36j2m1	>880	>7200.0	>161	>7200.0
c36j2m2	>2 801	>7200.0	>698	>7200.0
c36j2m3	>1 812	>7200.0	>336	>7200.0
c36j2m4	>589	>7200.0	>202	>7200.0
c36j2m5	>838	>7200.0	>189	>7200.0
c38j2m1	>619	>7200.0	>191	>7200.0
c38j2m2	>848	>7200.0	>161	>7200.0
c38j2m3	>1 311	>7200.0	>287	>7200.0
c38j2m4	>404	>7200.0	>83	>7200.0
c38j2m5	>414	>7200.0	>79	>7200.0
de14j3m3	15	6.7	25	7.3
de14j3m5	4	3.4	4	3.3
de16j3m2	4	5.0	4	5.1
de16j3m3	2	3.9	2	3.8
de18j3m3	2	13.1	2	13.2
de18j3m4	9	11.5	12	11.4
de18j3m5	279	74.9	208	77.5
de20j3m1	3	11.6	3	11.4
de20j3m2	4	6.5	4	6.6
de20j3m3	59	33.1	4	30.1
de20j3m4	304	213.9	464	355.4
de20j3m5	431	85.9	426	84.7
de22j3m1	562	343.8	407	721.0
de22j3m2	2	11.3	2	11.3
de22j3m3	1 178	1636.6	889	963.5
de22j3m4	448	127.2	5	13.1
de24j3m1	725	862.1	2 620	3673.6
de24j3m2	422	479.3	475	695.4
de24j3m3	>11 104	>7200.0	>11 778	>7200.0
de24j3m4	538	1242.3	706	1242.9
de24j3m5	3	15.9	3	16.1
de26j3m1	1 797	3747.1	>1 067	>7200.0
de26j3m2	>4 754	>7200.0	>1 571	>7200.0
de26j3m3	3 915	726.5	1 877	1344.7
de26j3m4	2 421	6535.2	1 487	4657.1
de26j3m5	45	111.5	53	119.1
de28j3m1	>2 949	>7200.0	>1 372	>7200.0
de28j3m2	121	296.1	99	290.8
de28j3m3	2 876	5334.6	2 287	6578.1
de28j3m4	>1 431	>7200.0	>185	>7200.0
de28j3m5	>3 364	>7200.0	>3 769	>7200.0
df16j3m1	57	2.8	57	2.5
df22j3m2	3	19.8	3	19.7
df22j3m3	15	18.5	10	17.9
df24j3m1	4	13.2	4	13.5
df24j3m2	3	13.7	4	13.7

	SB		MA	
	Nodes	Time	Nodes	Time
df24j3m5	4	36.0	4	36.5
df26j3m2	6	45.5	6	45.9
df28j3m2	2	22.3	2	21.2
df28j3m3	610	1014.0	633	1068.5
df28j3m4	3	24.9	3	25.7
df28j3m5	5	17.4	5	16.7
e10j2m1	139	5.8	241	6.6
e10j2m2	51	11.0	18	9.6
e10j2m3	3	2.6	3	2.4
e12j2m1	196	22.2	211	22.6
e12j2m2	2	2.9	2	2.9
e12j2m3	18	1.7	18	1.6
e12j2m4	136	37.8	190	62.8
e12j2m5	3	7.0	3	7.2
e15j3m1	5	7.1	6	7.5
e15j3m3	143	57.3	173	58.8
e15j3m4	188	21.7	153	16.2
e15j3m5	54	4.4	78	11.2
e20j4m1	3 889	505.9	1 223	115.7
e20j4m2	>6 403	>7200.0	2 738	5437.1
e20j4m3	28	18.6	6	15.7
e20j4m4	4 053	1348.4	311	151.3
e20j4m5	2 076	693.2	454	214.7
e25j5m1	>40 736	>7200.0	>28 460	>7200.0
e25j5m3	>9 568	>7200.0	7 970	3298.2
e25j5m4	20 133	1536.5	7 271	747.2
e25j5m5	381	832.9	1 099	1334.1
e30j6m1	2 048	3433.6	2 329	3069.1
e30j6m2	2 038	2356.0	930	2026.3
e30j6m3	1 703	813.2	1 479	908.0
e30j6m4	>3 100	>7200.0	>1 840	>7200.0
e30j6m5	>10 452	>7200.0	>8 108	>7200.0
e35j7m1	>2 418	>7200.0	>3 510	>7200.0
e35j7m2	>5 670	>7200.0	>2 926	>7200.0
e35j7m3	>3 128	>7200.0	>1 798	>7200.0
e35j7m4	>2 923	>7200.0	>1 741	>7200.0
e35j7m5	>1 462	>7200.0	>883	>7200.0
e40j8m1	>2 137	>7200.0	>472	>7200.0
e40j8m2	>632	>7200.0	>488	>7200.0
e40j8m3	>658	>7200.0	>340	>7200.0
e40j8m4	>1 545	>7200.0	>1 026	>7200.0
e40j8m5	>949	>7200.0	>465	>7200.0
e45j9m1	>1 299	>7200.0	>408	>7200.0
e45j9m2	>819	>7200.0	>297	>7200.0
e45j9m3	>1 729	>7200.0	>720	>7200.0
e45j9m4	>845	>7200.0	>334	>7200.0
e45j9m5	>770	>7200.0	>170	>7200.0
e50j10m1	>964	>7200.0	>273	>7200.0
e50j10m2	>667	>7200.0	>343	>7200.0
e50j10m3	>3 280	>7200.0	>531	>7200.0
e50j10m4	>2 028	>7200.0	>678	>7200.0
e50j10m5	>820	>7200.0	>343	>7200.0
geom. mean	663	736.9	363	672.1
sh. geom. mean	1 080	928.1	630	852.3
arithm. mean	2 866	4011.0	5 364	3893.7
all optimal				
geom. mean	138	56.4	85	47.0
sh. geom. mean	349	84.6	221	70.3
arithm. mean	1 299	497.0	3 368	407.1

Table 4: Detailed computational results on scheduling instances comparing full strong branching (SB-pure) and branching on multi-aggregated variables (MA-pure) with pure SCIP settings

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
c10j2m1	49	2.1	3	1.0
c10j2m3	221	3.1	25	1.0
c10j2m4	19	0.5	15	0.5
c10j2m5	217	7.0	7	2.2
c10j3m1	1 965	3.4	149	1.1
c10j3m2	17	0.5	9	0.5
c10j3m3	63	1.0	63	1.0
c10j4m2	3	0.5	3	0.5
c10j4m3	5	0.5	3	0.5
c12j2m1	21	2.3	3	1.1
c12j2m2	31 407	367.6	25 043	292.5
c12j2m3	743	3.7	21	1.4
c12j2m4	49	1.5	3	0.5
c12j2m5	115	2.7	5	0.8
c12j3m1	3	1.5	3	1.5
c12j3m2	633	5.8	47	1.8
c12j3m3	101	3.1	5	1.3
c12j3m4	8 995	37.6	31	1.1
c12j3m5	2 023	9.1	199	2.6
c12j4m1	33	0.5	39	0.5
c12j4m2	77	1.4	21	1.2
c12j4m3	25	0.5	5	0.5
c12j4m4	3	0.7	5	0.8
c12j4m5	49	1.9	5	1.8
c14j2m1	59	4.6	3	1.6
c14j2m2	8 257	120.5	1 579	34.0
c14j2m3	27	3.4	3	1.4
c14j3m1	747 215	4278.8	>687 239	>7200.0
c14j3m2	155	2.9	45	1.7
c14j3m3	83	5.0	353	8.7
c14j3m4	3 071	54.8	625	14.7
c14j3m5	839	32.3	2 357	22.2
c14j4m1	4 489	74.6	677	19.8
c14j4m2	59	2.1	23	1.9
c14j4m3	1 609	32.4	983	20.5
c14j4m4	323 933	3720.7	35 473	461.2
c14j4m5	23	1.6	213	5.8
c16j2m2	45 181	3039.5	39 179	4962.0
c16j2m3	>88 241	>7200.0	>96 923	>7200.0
c16j2m5	359 745	6731.8	>194 747	>7200.0
c16j3m1	>136 196	>7200.0	>262 540	>7200.0
c16j3m2	>180 160	>7200.0	>160 720	>7200.0
c16j3m3	3 267	212.7	1 021	78.5
c16j3m4	47 529	832.2	19 781	312.5
c16j3m5	11	1.0	3	1.1
c16j4m1	15	7.6	13	7.1
c16j4m2	9 067	132.8	1 561	27.4
c16j4m3	53	2.6	43	2.3
c16j4m4	127	6.0	21	4.8
c16j4m5	>453 201	>7200.0	125 369	2285.4
c18j2m1	113	42.4	5	6.4
c18j2m2	>34 343	>7200.0	3 479	773.6
c18j2m3	3 327	149.5	32 171	3610.8
c18j2m4	14 661	1932.6	28 781	4170.4
c18j2m5	169	21.6	7	9.4

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
c18j3m1	>21 044	>7200.0	5 475	633.0
c18j3m2	>91 581	>7200.0	>56 864	>7200.0
c18j3m3	>31 348	>7200.0	>67 788	>7200.0
c18j3m4	>39 936	>7200.0	>70 184	>7200.0
c18j3m5	>120 800	>7200.0	>132 452	>7200.0
c18j4m1	12 003	278.6	201 583	2112.7
c18j4m2	>87 114	>7200.0	>147 087	>7200.0
c18j4m3	13 699	478.9	211	34.4
c18j4m4	1 451	107.9	2 409	70.1
c18j4m5	>186 435	>7200.0	54 089	2461.0
c20j2m1	>25 471	>7200.0	>29 028	>7200.0
c20j2m2	75	47.6	3	7.2
c20j2m3	>23 541	>7200.0	>16 730	>7200.0
c20j2m4	>14 355	>7200.0	19	36.4
c20j2m5	>27 333	>7200.0	695	131.1
c20j3m1	>79 222	>7200.0	>172 405	>7200.0
c20j3m2	>21 298	>7200.0	153 015	3338.9
c20j3m3	>11 757	>7200.0	>77 074	>7200.0
c20j3m4	>24 433	>7200.0	>8 486	>7200.0
c20j3m5	>78 909	>7200.0	>56 921	>7200.0
c20j4m1	>50 021	>7200.0	>82 151	>7200.0
c20j4m2	>106 193	>7200.0	>289 541	>7200.0
c20j4m3	>17 774	>7200.0	>20 886	>7200.0
c20j4m4	4 147	438.1	281	62.8
c20j4m5	>320 482	>7200.0	>256 111	>7200.0
c22j2m1	>7 131	>7200.0	>8 990	>7200.0
c22j2m2	655	346.1	5	14.0
c22j2m3	>11 865	>7200.0	>15 339	>7200.0
c22j2m4	>4 134	>7200.0	>10 231	>7200.0
c22j2m5	>24 475	>7200.0	>16 757	>7200.0
c22j3m1	>10 405	>7200.0	>6 737	>7200.0
c22j3m2	>14 831	>7200.0	>10 536	>7200.0
c22j3m3	>16 106	>7200.0	>8 157	>7200.0
c22j3m4	>9 399	>7200.0	>9 079	>7200.0
c22j3m5	>23 371	>7200.0	>10 625	>7200.0
c22j4m1	>28 142	>7200.0	>19 973	>7200.0
c22j4m2	>10 135	>7200.0	>79 549	>7200.0
c22j4m3	>27 578	>7200.0	>34 021	>7200.0
c22j4m4	105	32.9	7	17.0
c22j4m5	7 049	1509.7	55	80.3
c24j2m1	>1 564	>7200.0	>3 752	>7200.0
c24j2m2	>6 016	>7200.0	>4 865	>7200.0
c24j2m3	109	467.5	3	39.1
c24j2m4	>5 973	>7200.0	>9 990	>7200.0
c24j2m5	>3 430	>7200.0	205	950.7
c24j3m1	>6 925	>7200.0	>3 480	>7200.0
c24j3m2	>9 850	>7200.0	>24 348	>7200.0
c24j3m3	>8 767	>7200.0	>5 818	>7200.0
c24j3m4	>7 122	>7200.0	>10 951	>7200.0
c24j3m5	>4 626	>7200.0	>4 262	>7200.0
c24j4m1	>6 976	>7200.0	>10 005	>7200.0
c24j4m2	>13 842	>7200.0	>10 898	>7200.0
c24j4m3	>7 688	>7200.0	>4 193	>7200.0
c24j4m4	>7 289	>7200.0	>3 802	>7200.0
c24j4m5	>11 632	>7200.0	>11 571	>7200.0
c26j2m1	>2 038	>7200.0	>5 318	>7200.0
c26j2m2	>4 846	>7200.0	>3 787	>7200.0
c26j2m3	>2 650	>7200.0	>4 884	>7200.0
c26j2m5	1 309	3204.2	7	53.3
c26j3m1	>2 097	>7200.0	>1 476	>7200.0

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
c26j3m2	>2 807	>7200.0	>2 844	>7200.0
c26j3m3	>1 009	>7200.0	>2 335	>7200.0
c26j3m4	>4 899	>7200.0	>2 286	>7200.0
c26j3m5	>1 763	>7200.0	>730	>7200.0
c26j4m1	>3 432	>7200.0	>4 278	>7200.0
c26j4m2	>3 019	>7200.0	>2 044	>7200.0
c26j4m3	>20 552	>7200.0	>3 075	>7200.0
c26j4m4	>6 668	>7200.0	>5 077	>7200.0
c26j4m5	>3 959	>7200.0	>4 184	>7200.0
c28j2m1	1 025	2107.0	5	35.3
c28j2m2	333	1199.0	7	37.9
c28j2m4	>661	>7200.0	>2 603	>7200.0
c28j2m5	>1 632	>7200.0	>4 127	>7200.0
c28j3m1	>3 791	>7200.0	>961	>7200.0
c28j3m2	283	4014.0	7	110.2
c28j3m3	>3 355	>7200.0	>2 253	>7200.0
c28j3m4	>2 523	>7200.0	>1 019	>7200.0
c28j3m5	>2 006	>7200.0	>1 978	>7200.0
c28j4m1	>3 906	>7200.0	>1 041	>7200.0
c28j4m2	>3 290	>7200.0	55	503.2
c28j4m3	>1 625	>7200.0	>835	>7200.0
c28j4m4	>3 552	>7200.0	>3 234	>7200.0
c28j4m5	>4 178	>7200.0	>2 890	>7200.0
c30j2m1	>1 165	>7200.0	13	261.7
c30j2m2	>683	>7200.0	11	158.1
c30j2m3	>1 228	>7200.0	>710	>7200.0
c30j2m4	>1 541	>7200.0	>372	>7200.0
c30j2m5	>1 608	>7200.0	13	328.5
c30j3m1	>1 151	>7200.0	>415	>7200.0
c30j3m2	>1 660	>7200.0	>548	>7200.0
c30j3m3	>540	>7200.0	15	587.9
c30j3m4	>1 175	>7200.0	>280	>7200.0
c30j3m5	>304	>7200.0	>346	>7200.0
c30j4m1	>2 009	>7200.0	>1 134	>7200.0
c30j4m2	>2 483	>7200.0	>726	>7200.0
c30j4m3	>1 971	>7200.0	>922	>7200.0
c30j4m4	>1 263	>7200.0	>1 023	>7200.0
c30j4m5	>3 365	>7200.0	>1 758	>7200.0
c32j2m1	>522	>7200.0	51	919.5
c32j2m2	>518	>7200.0	>329	>7200.0
c32j2m3	>186	>7200.0	5	196.7
c32j2m4	>874	>7200.0	>995	>7200.0
c32j3m1	>1 206	>7200.0	283	5718.5
c32j3m2	>848	>7200.0	>487	>7200.0
c32j3m3	>1 309	>7200.0	>288	>7200.0
c32j3m4	>745	>7200.0	>393	>7200.0
c32j3m5	>2 255	>7200.0	>325	>7200.0
c32j4m1	>1 238	>7200.0	>809	>7200.0
c32j4m2	>1 133	>7200.0	>322	>7200.0
c32j4m3	>1 354	>7200.0	>308	>7200.0
c32j4m4	>1 331	>7200.0	>894	>7200.0
c32j4m5	>1 622	>7200.0	>308	>7200.0
c34j2m2	>143	>7200.0	>166	>7200.0
c34j2m3	>197	>7200.0	5	220.0
c34j2m4	>113	>7200.0	7	373.2
c34j2m5	>808	>7200.0	>403	>7200.0
c36j2m1	>358	>7200.0	5	428.9
c36j2m2	>70	>7200.0	>147	>7200.0
c36j2m3	>74	>7200.0	3	402.6
c36j2m5	>30	>7200.0	3	656.2

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
c38j2m1	>292	>7200.0	>195	>7200.0
c38j2m3	>581	>7200.0	45	1946.5
c38j2m5	183	3238.2	3	160.4
de12j3m5	11	2.4	3	2.2
de14j3m3	11	4.1	9	4.0
de14j3m4	3	2.5	3	2.4
de14j3m5	6 053	87.5	925	20.8
de16j3m2	287	8.9	147	7.2
de16j3m3	283	16.5	65	6.5
de16j3m5	71	3.5	71	3.3
de18j3m2	3	7.5	3	8.4
de18j3m3	277	22.8	23	14.6
de18j3m4	131	20.5	45	14.2
de18j3m5	5	5.3	5	5.0
de20j3m1	3 431	1047.1	19	39.3
de20j3m2	69	32.0	13	12.4
de20j3m3	105	19.4	25	16.5
de20j3m4	345	119.2	311	118.8
de22j3m1	21 295	4328.4	1 069	393.6
de22j3m2	12 707	6118.5	975	574.1
de22j3m3	181	140.1	181	141.3
de22j3m4	105	42.5	273	111.7
de22j3m5	3	4.3	3	4.2
de24j3m1	>17 580	>7200.0	17 871	3860.2
de24j3m2	>35 958	>7200.0	>30 527	>7200.0
de24j3m5	>23 010	>7200.0	>20 387	>7200.0
de26j3m1	>2 432	>7200.0	>3 440	>7200.0
de26j3m2	>3 383	>7200.0	>3 227	>7200.0
de26j3m3	>8 309	>7200.0	>80 935	>7200.0
de26j3m4	>8 940	>7200.0	>11 715	>7200.0
de26j3m5	>8 597	>7200.0	2 009	5441.8
de28j3m1	>3 138	>7200.0	>1 711	>7200.0
de28j3m2	>2 379	>7200.0	>2 813	>7200.0
de28j3m4	>788	>7200.0	>597	>7200.0
de28j3m5	>2 158	>7200.0	>7 978	>7200.0
df14j3m5	21	0.5	7	0.6
df16j3m5	3	1.1	3	1.1
df18j3m2	37	13.9	13	12.4
df18j3m3	5	6.2	5	6.4
df18j3m4	25	3.2	3	2.5
df18j3m5	5	5.4	5	5.4
df20j3m4	399	19.7	5	3.0
df20j3m5	23	10.0	23	10.2
df22j3m1	3	17.1	3	17.3
df22j3m2	29	20.8	13	19.9
df22j3m3	317	49.1	107	32.4
df22j3m5	433	81.4	39	9.1
df24j3m1	141	58.5	169	97.0
df24j3m2	1 059	258.4	171	94.3
df24j3m3	3	7.4	3	7.5
df24j3m5	21	27.4	9	26.6
df26j3m1	9	11.6	9	11.5
df26j3m2	3 097	2377.0	359	592.1
df26j3m3	163	41.7	13	12.0
df28j3m1	21	27.3	3	12.7
df28j3m2	539	1097.3	3	91.9
df28j3m3	2 175	1460.3	741	1867.0
df28j3m5	>3 846	>7200.0	>1 593	>7200.0
e10j2m2	305	10.3	55	8.9
e10j2m3	145	2.1	7	1.4

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
e10j2m5	101	2.1	25	2.1
e12j2m1	15 329	453.6	5 781	127.9
e12j2m4	311	26.0	75	12.1
e12j2m5	167	9.9	19	6.2
e15j3m1	179 591	3125.2	29 345	175.0
e15j3m3	>248 801	>7200.0	341 075	3219.7
e15j3m5	5 057	126.9	2 455	63.3
e20j4m1	40 549	3837.0	16 241	1447.6
e20j4m2	>28 054	>7200.0	>17 905	>7200.0
e20j4m3	>113 688	>7200.0	3 293	311.8
e20j4m4	2 925	280.1	343	100.6
e20j4m5	>92 555	>7200.0	18 853	982.3
e25j5m1	>25 764	>7200.0	>42 414	>7200.0
e25j5m3	>27 015	>7200.0	>47 234	>7200.0
e25j5m4	>150 463	>7200.0	>146 369	>7200.0
e25j5m5	>27 133	>7200.0	>95 311	>7200.0
e30j6m1	>4 854	>7200.0	>13 887	>7200.0
e30j6m2	>46 756	>7200.0	>199 679	>7200.0
e30j6m3	181	52.1	99 795	6381.3
e30j6m4	>8 288	>7200.0	>9 959	>7200.0
e30j6m5	>11 604	>7200.0	>26 337	>7200.0
e35j7m1	>10 736	>7200.0	>6 618	>7200.0
e35j7m2	>8 566	>7200.0	>6 067	>7200.0
e35j7m3	>6 386	>7200.0	>4 220	>7200.0
e35j7m4	>5 896	>7200.0	>3 385	>7200.0
e35j7m5	>8 469	>7200.0	>3 899	>7200.0
e40j8m1	>6 330	>7200.0	>3 180	>7200.0
e40j8m2	>3 620	>7200.0	>3 028	>7200.0
e40j8m3	>2 408	>7200.0	>2 957	>7200.0
e40j8m4	>8 758	>7200.0	>5 774	>7200.0
e40j8m5	>4 323	>7200.0	>3 041	>7200.0
e45j9m1	>2 966	>7200.0	>1 679	>7200.0
e45j9m2	>1 990	>7200.0	>1 679	>7200.0
e45j9m3	>1 354	>7200.0	>1 091	>7200.0
e45j9m4	>3 103	>7200.0	>1 927	>7200.0
e45j9m5	>2 352	>7200.0	>2 627	>7200.0
e50j10m1	>1 479	>7200.0	>1 099	>7200.0
e50j10m2	>1 896	>7200.0	>1 513	>7200.0
e50j10m3	>1 900	>7200.0	>850	>7200.0
e50j10m4	>3 342	>7200.0	>1 815	>7200.0
e50j10m5	>1 057	>7200.0	>1 364	>7200.0
geom. mean	1 442	704.8	565	415.0
sh. geom. mean	2 106	980.3	1 256	625.7
arithm. mean	20 449	4383.1	19 582	3737.8
all optimal				
geom. mean	227	27.4	52	13.2
sh. geom. mean	472	51.8	196	26.4
arithm. mean	7 425	468.8	4 828	258.2

Table 5: Detailed computational results on general MIP instances comparing full strong branching (SB) and branching on multi-aggregated variables (MA) with SCIP's default settings

	SB		MA	
	Nodes	Time	Nodes	Time
neos-1061020	>22	>7200.0	>22	>7200.0
neos-1215259	840	4763.3	>809	>7200.0
neos-1223462	>1	>7200.0	>1	>7200.0
neos-1224597	80	2170.6	76	3152.2
neos-1281048	345	234.6	322	270.4
neos-1445755	6	61.8	6	61.8
neos-1445765	5	133.0	5	134.4
neos-530627	3	0.5	3	0.5
neos-555298	>1 293	>7200.0	582	2797.3
neos-555343	>75	>7200.0	>75	>7200.0
neos-555424	>353	>7200.0	>338	>7200.0
neos-555694	86	76.1	86	101.2
neos-555884	>1 128	>7200.0	>504	>7200.0
neos-555927	>5 790	>7200.0	>4 814	>7200.0
neos-780889	>10	>7200.0	>9	>7200.0
neos-785899	200	115.5	249	210.2
neos-785912	>3 314	>7200.0	>594	>7200.0
neos-785914	15	443.2	19	513.3
neos-799838	46	4393.7	50	6098.0
neos-848845	>67	>7200.0	>55	>7200.0
neos-849702	>118	>7200.0	>115	>7200.0
neos-850681	>2 150	>7200.0	>1 257	>7200.0
neos-881765	18	8.1	18	10.1
neos-905856	>2 823	>7200.0	>1 398	>7200.0
neos-912015	281	431.7	187	440.5
neos-912023	19	26.6	28	48.7
neos-916173	723	233.0	1 470	267.5
neos-941313	>2	>7200.0	>2	>7200.0
neos-953928	>28	>7200.0	>29	>7200.0
neos-954925	>4	>7200.0	>4	>7200.0
neos-957143	>26	>7200.0	>26	>7200.0
Test3	4	7.5	4	8.0
air04	76	2870.5	78	4133.2
air05	151	1406.9	157	1771.2
dsbmip	21	2.1	19	2.3
fiber	35	2.0	17	1.9
gesa3_o	10	2.4	8	2.3
harp2	223 578	1193.3	405 262	2814.2
lseu	110	0.5	119	0.7
nw04	5	38.0	5	42.3
p0033	2	0.5	2	0.5
qnet1_o	18	9.7	18	11.9
atlanta-ip	>15	>7200.0	>2	>7200.0
momentum2	>97	>7200.0	>47	>7200.0
msc98-ip	>19	>7200.0	>8	>7200.0

	SB		MA	
	Nodes	Time	Nodes	Time
mzzv11	>241	>7200.0	>151	>7200.0
mzzv42z	>83	>7200.0	>59	>7200.0
roll3000	>9 834	>7200.0	>4 713	>7200.0
30_70_45_095_100	>3	>7200.0	>3	>7200.0
bab5	>232	>7200.0	>135	>7200.0
bnatt350	>547	>7200.0	>434	>7200.0
bnatt400	>393	>7200.0	>198	>7200.0
co-100	>29	>7200.0	>6	>7200.0
eil33-2	465	497.1	489	857.5
eilA101-2	>8	>7200.0	>4	>7200.0
eilB101	13	6714.9	>1	>7200.0
gmu-35-40	>1 372 877	>7200.0	>1 110 750	>7200.0
gmu-35-50	>955 757	>7200.0	>867 157	>7200.0
lrsa120	>15 176	>7200.0	>10 363	>7200.0
neos16	>40 083	>7200.0	>23 125	>7200.0
ns894244	>1	>7200.0	>1	>7200.0
ns894788	>123	>7200.0	>100	>7200.0
pigeon-10	>2 738 385	>7200.0	>2 517 457	>7200.0
pigeon-11	>2 141 627	>7200.0	>1 804 096	>7200.0
pigeon-12	>1 736 097	>7200.0	>1 550 643	>7200.0
pigeon-13	>1 704 951	>7200.0	>1 482 784	>7200.0
pw-myciel4	>2 454	>7200.0	>856	>7200.0
rocII-4-11	>2 139	>7200.0	>1 362	>7200.0
rococoB10-011000	>321	>7200.0	>9	>7200.0
rococoC10-001000	>8 903	>7200.0	>381	>7200.0
rococoC11-011100	>3	>7200.0	>1	>7200.0
satellites1-25	>1	>7200.0	>1	>7200.0
triptim1	>1	>7200.0	>1	>7200.0
p1_cpa_ilpi_mip.3	953	13.4	969	19.1
p18_cpa_ilpi_mip.6	10	3.8	8	3.8
p9_cpa_ilpi_mip.22	218	50.5	160	55.4
dfn6_cost	10	13.4	8	13.2
gwin10_f_l	73	18.2	88	23.2
gwin10_f_l_nonp	250	109.1	376	154.6
gwin8_f_l_nonp	21	4.6	30	7.7
gwin8_nos_l_nonp	1 256	32.9	2 227	42.2
aa06	2	17.9	2	18.1
neos20	293	92.6	404	166.6
air02	4	2.1	4	2.0
air06	2	2.8	2	3.2
30:30:4.5:0.95:98	>15	>7200.0	>15	>7200.0
p16_cpa_ilpi_mip.29	35 814	4306.2	27 426	6631.5
p4_cpa_ilpi_mip.37	30 600	6514.4	>18 732	>7200.0
p6_cpa_ilpi_mip.72	>25 818	>7200.0	>13 941	>7200.0
p8_cpa_ilpi_mip.30	>42 855	>7200.0	>26 669	>7200.0
gwin7_f_cost	>1	>7200.0	>1	>7200.0
gwin7_nos_load	8 520	524.5	18 507	1819.8
pubtrans2	2	715.7	2	715.4
aa01	76	2855.2	78	4148.2
bs-2006-03-24-17-15-01	>2	>7200.0	>2	>7200.0
t0415pre	>14	>7200.0	>8	>7200.0
v1619	5	136.0	4	141.3
v1620	3	65.4	3	66.2
v1620pre	26	165.1	23	172.5

	SB		MA	
	Nodes	Time	Nodes	Time
county04-normal-bicrit	>29 636	>7200.0	>33 619	>7200.0
net-5-3-7	47 598	3969.5	44 700	5970.1
net-5-3-7b	88 538	2846.7	84 184	3791.9
neos10	4	9.7	4	9.9
neos7	122 399	832.8	75 500	1140.8
30:30:4.5:0.95:100	>11	>7200.0	>11	>7200.0
30:70:4.5:0.95:98	>2	>7200.0	>2	>7200.0
bmc-ibm-5	5	14.1	14	38.1
geom. mean	163	815.6	126	911.1
sh. geom. mean	586	1048.0	499	1151.2
arithm. mean	106 633	4160.3	94 784	4292.6
all optimal				
geom. mean	72	62.7	76	80.4
sh. geom. mean	253	100.4	269	126.3
arithm. mean	10 864	635.9	13 539	941.0

Table 6: Detailed computational results on general MIP instances comparing full strong branching (SB-pure) and branching on multi-aggregated variables (MA-pure) with pure SCIP settings

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
neos-1061020	15	2355.6	15	2938.1
neos-1215259	121	795.5	127	1229.8
neos-1281048	155	106.7	137	121.3
neos-530627	3 289 186	212.6	852 689	231.4
neos-555884	>457	>7200.0	>315	>7200.0
neos-555927	>5 985	>7200.0	>5 455	>7200.0
neos-785899	3	0.6	1	0.7
neos-785912	>1 081	>7200.0	>2 326	>7200.0
neos-799838	>16	>7200.0	>13	>7200.0
neos-850681	>2 158	>7200.0	>1 567	>7200.0
neos-905856	1 561	5086.6	999	5425.7
neos-912023	>3 797	>7200.0	>2 419	>7200.0
neos-916173	633	293.1	1 399	356.3
neos-953928	>10	>7200.0	>10	>7200.0
neos-954925	>54	>7200.0	>55	>7200.0
Test3	167	12.9	139	17.0
air05	7	116.0	7	198.3
fiber	1 909	36.4	1 761	57.4
gesa3_o	237	6.1	225	7.8
harp2	>624 682	>7200.0	>327 587	>7200.0
lseu	1 125	0.7	1 017	1.0
nw04	19	40.3	15	46.5
p0033	105	0.5	51	0.5
qnet1_o	29	1.1	19	1.0
atlanta-ip	>3	>7200.0	>2	>7200.0
momentum2	>131	>7200.0	>69	>7200.0
msc98-ip	>25	>7200.0	>13	>7200.0
mzzv11	31	6279.9	>15	>7200.0
mzzv42z	3	3180.4	3	4401.9
roll3000	>9 582	>7200.0	>5 733	>7200.0
30n20b8	>35	>7200.0	>25	>7200.0
bab5	>256	>7200.0	>189	>7200.0
bnatt400	>448	>7200.0	>271	>7200.0
co-100	>35	>7200.0	>22	>7200.0
eil33-2	333	218.4	325	301.2
eilA101-2	>1	>7200.0	>1	>7200.0
eilB101	165	941.7	143	1436.8
gmu-35-40	>782 899	>7200.0	>544 528	>7200.0
gmu-35-50	>413 065	>7200.0	>283 401	>7200.0
lrsa120	>8 517	>7200.0	>5 747	>7200.0
neos16	>51 047	>7200.0	>34 746	>7200.0
ns1745726	7	25.3	7	27.3
pigeon-10	>1 540 274	>7200.0	>1 200 339	>7200.0
pigeon-11	>1 621 469	>7200.0	>1 398 439	>7200.0
pigeon-12	>1 279 702	>7200.0	>906 861	>7200.0
pigeon-13	>1 247 952	>7200.0	>981 960	>7200.0
pw-myciel4	>1 450	>7200.0	>833	>7200.0
rocII-4-11	>2 649	>7200.0	>1 576	>7200.0
rococoB10-011000	>676	>7200.0	>198	>7200.0
rococoC10-001000	>1 244	>7200.0	>274	>7200.0
rococoC11-011100	>432	>7200.0	>191	>7200.0
satellites1-25	>1	>7200.0	>1	>7200.0
p1_cpa_ilpi_mip.3	5 769	30.6	4 767	39.0
p18_cpa_ilpi_mip.6	>138 913	>7200.0	>101 450	>7200.0
p9_cpa_ilpi_mip.22	887	247.0	765	319.4

	SB-pure		MA-pure	
	Nodes	Time	Nodes	Time
dfn6_cost	3	5.8	3	12.8
gwin10_f_l	31	10.0	27	17.4
gwin10_f_l_nonp	59	36.3	9	54.7
gwin8_nos_l_nonp	>1 126 823	>7200.0	>1 073 159	>7200.0
neos20	95 151	3198.7	73 015	3786.4
p16_cpa_ilpi_mip.29	202 995	4105.4	140 791	6868.5
p4_cpa_ilpi_mip.37	>49 188	>7200.0	>25 953	>7200.0
p6_cpa_ilpi_mip.72	>28 084	>7200.0	>14 193	>7200.0
p8_cpa_ilpi_mip.30	>37 229	>7200.0	>21 411	>7200.0
gwin7_f_cost	>2 396	>7200.0	>11	>7200.0
gwin7_nos_load	21 555	3430.3	20 497	4517.2
t0415pre	>14	>7200.0	>8	>7200.0
v1619	>1 961	>7200.0	>1 477	>7200.0
v1620	>1 415	>7200.0	>886	>7200.0
v1620pre	>1 289	>7200.0	>677	>7200.0
county04-normal-bicrit	>15 706	>7200.0	>5 746	>7200.0
net-5-3-7	41 067	5489.6	39 415	7021.8
net-5-3-7b	89 331	1863.4	85 585	2260.7
neos10	3	5.7	1	6.0
neos7	185 461	602.1	95 015	771.4
bmc-ibm-5	43	12.3	41	16.2
geom. mean	1 057	1151.8	688	1291.7
sh. geom. mean	2 251	1441.7	1 682	1579.8
arithm. mean	170 280	4583.5	108 804	4727.5
all optimal				
geom. mean	393	93.1	296	121.7
sh. geom. mean	983	150.9	852	188.9
arithm. mean	123 066	1014.6	41 219	1327.9