# Lexicographic Branch-and-Bound Column Search

Andreas Bärmann[1], Alexander Müller[1] and Dieter Weninger[1]

[1] Andreas.Baermann@fau.de
Alexander.AMR.Mueller@fau.de
Dieter.Weninger@fau.de
Lehrstuhl für Analytics & Mixed-Integer Optimization,
Department of Data Science / Department Mathematik,
Friedrich-Alexander-Universität Erlangen-Nürnberg,
Cauerstraße 11, 91058 Erlangen, Germany

**Abstract**

We present an exact generic method for solving the pricing problem in a column generation approach, which we call *branch-and-bound column search*. It searches the space of all feasible columns via a branch-and-bound tree search and returns all columns with a reduced-cost value below a certain threshold. The approach is based on an idea from Krumke et al. (2002) for the solution of vehicle routing problems. In this work, we formalize and generalize this method such that it can be applied to a large variety of different problem classes. We further derive strong bounds to effectively prune the search tree, which allows us to significantly speed up the generation of columns and thus reduces the overall solution time. In addition, we show that our column search methods can be combined with lexicographic optimization easily and efficiently to solve problems with multiple objectives.

Finally, the theoretical results are corroborated in an extensive computational study for two different applications: the sequence-dependent cutting stock problem and the optimized assignment of transport orders to employees in a hospital. On the basis of real-world data, we can show in both cases that it is superior to use the branch-and-bound column search as pricing algorithm in the presented *restricted master heuristic* than conventional column generation. In particular, for the second application the heurstic based on the branch-and-bound column search even outperforms both an industry-standard heuristic for the problem as well as a standard solver on a polynomial-size MIP formulation within a real-world problem setting.

**Keywords:** Column Generation, Pricing Problem, Branch-and-Bound Column Search, Lexicographic Optimization, Cutting Stock Problem, Machine Scheduling

**Mathematics Subject Classification:** 90C57 – 90C10 – 90C29 – 90C90

## Acknowledgements

We are grateful for continuous support from the OrgaCard Siemantel & Alt GmbH and for many fruitful and stimulating discussions with Klaus Müller, Ralf Alt and Roland Rieß. Finally, we acknowledge financial support by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics – Data – Applications (ADA-Center) within the framework of "BAYERN DIGITAL II". We would also like to express our gratitude to Wim Vancroonenburg, Michele Garraffa, Fabio Salassa, Greet Vanden Berghe and Tony Wauters for providing us the data for our computational study on the sequence-dependent cutting stock problem.

# 1 Introduction

*Column generation* (*CG*) has nowadays become a prevalent method to deal with a large number of variables in *linear programs* (*LP*) and *mixed-integer programs* (*MIPs*), cf. Lübbecke and Desrosiers (2005). As its working principle, column generation exploits the fact that the majority of columns will not be part of an optimal solution, i.e. their associated variables are equal to zero in the optimum anyway. It therefore only starts with a subset of columns and generates new ones that have the potential to improve the current solution, namely columns with negative reduced costs (in case of a minimization problem), until an optimal solution is reached. At each iteration a *restricted master problem* (*RMP*) and a *pricing subproblem* are solved, where (RMP) is the continuous relaxation of the original problem formulation – called *master problem* (*MP*) – and is restricted to a small subset of the variables of (MP). The subproblem aims to find columns with negative reduced costs with respect to the dual solution of the current (RMP) which then enter the restricted master problem as new columns of the constraint matrix.

The structure of the pricing problem strongly depends on the individual problem at hand, which is why very often specialized algorithms are derived for its efficient solution. In particular, Krumke et al. (2002) developed an enumerative method to solve the subproblem for a real-world vehicle dispatching problem. Their method builds a search tree whose nodes correspond to feasible columns of the master problem constraint matrix. In order to reduce the number of nodes to be visited in such a tree, they derive an efficient application-specific bound for pruning. The resulting solution procedure for this particular application was then revisited by Westphal and Krumke (2008) and further improved by implementing an efficient solution method for an auxiliary combinatorial problem that is solved in order to reduce the search space in the pricing problem. Among others, Friese and Rambau (2006), Van Huigenbosch et al. (2011) and Hiller et al. (2014) have also used enumerative methods to generate columns. All the mentioned approaches were developed with the aim of solving a very specific application. However, we discovered that the method by Krumke et al. (2002) has the potential to work particularly well not only for the application studied there, but also in other, very general problem contexts. This motivated us to develop it towards a generic method for solving pricing subproblems, as outlined in the following.

**Contribution** Inspired by the work of Krumke et al. (2002), we derive a generally applicable method called *branch-and-bound column search* for solving the pricing subproblem in column generation. It can be used in a wide variety of application areas and problem classes. Our approach enumeratively explores the search space of feasible columns by building a special kind of enumeration tree containing columns that could potentially lead to a better solution within the column generation process and are thus candidates to enter the constraint matrix of the master problem. To this end, we develop a dedicated enumeration algorithm to ensure any column that is required for an optimal solution of the LP-relaxed master problem is visited. At the same time, our algorithm guarantees that none of the required columns is enumerated more than once. We further give an extension of this method which is especially suited for *sequence-dependent* subproblems, where certain items have to be ordered (as in the travelling salesman problem, for example). A pure enumeration of all columns is of course computationally intractable. Therefore, we prune the search space using provably exact bounds that allow us to dynamically estimate the reduced cost of all columns in the subtree below a given column in the enumeration process. In this way, we manage to prune the search tree effectively and thus considerably accelerate the overall solution process. The branch-and-bound column search is integrated into a *restricted master heuristic* (cf. Joncour et al. (2010)) based on column generation. The heuristic developed in this way makes it possible to solve integer problems efficiently and very often even to optimality.

Moreover, we show that our branch-and-bound column search can also be coupled very easily with a *lexicographic optimization* approach to solve multi-criteria optimization problems (see e.g. Isermann (1982)). Due to the additional constraints of the master problems in multi-stage lexicographic optimization (cf. Zykina (2004)), we will see that there even is additional dual information that can be exploited for the generation of columns in the pricing problem. In particular, columns found in one lexicographic stage can be used in other stages as well, and the bounds for pruning the search tree can be transferred to this setting. Further, we present a top-level algorithm for the solution of the master problem that is able to combine the lexicographic optimization approach very efficiently with our branch-and-bound column search algorithm.

In an extensive empirical study, we evaluate the efficiency of branch-and-bound column search and its combination with the lexicographic optimization approach at the hand of two indicative applications. These are the sequence-dependent cutting stock problem and the optimized assignment of transport orders to employees in a hospital, where the latter is modelled as a machine scheduling problem. Our exposition includes a tutorial on how to compute the pruning bounds in each case. For problem instances stemming from real-world data, we clearly demonstrate the computational benefits of our method.

**Distinction from other enumerative pricing methods** Another enumerative approach for the pricing subproblem in the literature is to use *constraint programming* (*CP*), see Hooker and van Hoeve (2018) for a short overview. Constraint programming is a paradigm for solving combinatorial search problems. Capone et al. (2010) give an introduction to this technique. Frameworks for using CP to solve pricing problems with constrained shortest-path problems were presented by Junker et al. (1999) as well as Yunes et al. (1999) and Yunes et al. (2005). While CP tries to reduce the domains of the variables in order to reduce the search space, which typically is a very application-specific procedure, the central goal in our approach is to prune the entire search tree using bounds on the reduced costs. Fahle et al. (2002) and Gendron et al. (2005) describe the idea of a so-called *negative-reduced-cost constraint* for CP, which works in a similar fashion, but is again problem-specific. Another frequently used method for solving the pricing problem is dynamic programming (DP). The idea of dynamic programming is to simplify a complex problem by breaking it down into simpler subproblems, which are then solved recursively using the optimal solutions to the subproblems until a solution to the original problem is found. In many applications, dynamic programming is used to solve pricing problems, which appear as shortest path problems, see Desrochers et al. (1992), Ceselli et al. (2009) or Engineer et al. (2011). However, dynamic programming can also be used for generating columns solving other problem classes such as cutting stock (cf. Cintra et al. (2008)) or bin packing problems (cf. Sadykov and Vanderbeck (2013)). There are also some parallels between DP and the branch-and-bound column search, but in the method at hand there are no relationships between the subproblems that have to be solved recursively nor are any optimal substructures exploited when solving the pricing problem. Therefore, it is not a typical dynamic programming method. Another approach for solving the pricing problem is using binary decision diagrams (BDD). Binary decision diagrams are directed acyclic graphs that can be generated by reducing binary decision trees. BDDs have already been used to solve the pricing problem of parallel machine scheduling problems (cf. Leus and Kowalczyk (2016)) and graph colouring problems (cf. Morrison et al. (2016)).

**Structure** The remainder of this work is structured as follows. In Section 2, our exposition begins with the introduction of a generic master problem and the description of the classical approach for solving the pricing subproblem in a column generation algorithm. Afterwards, Section 3 formalizes enumerative column generation by defining enumeration trees and devising the method of branch-and-bound column search. Then we explain how these enumeration trees can be efficiently pruned to reduce the runtime of the corresponding search algorithms. We also investigate one variation of branch-and-bound column search and explain their integration into a column generation framework. Section 4 then introduces lexicographical optimization and shows how it can be naturally coupled with branch-and-bound column search. In Section 5, we present two tutorials to illustrate how to apply the developed methods to real-world industrial problems, before the superior performance of the presented methods is shown in a detailed computational study in both cases. We finish with our conclusion in Section 6.

## 2 A Basic Column Generation Setting

We begin by introducing a very general framework for column generation in which branch-and-bound column search can be applied. To this end, we define a generic master problem and derive a formula for the reduced costs of the columns in its LP relaxation which we will use throughout this work.

**Notation** In the following, vectors $x \in \mathbb{R}^n$ are always column vectors, while for row vectors we write $x^\top$. For a matrix $A = (a_{i,j})_{i=1,\dots,m;j=1,\dots,n} \in \mathbb{R}^{m\times n}$, the $i$-th row and the $j$-th column are given by $A_{i,\cdot} := (a_{i,1}, \dots, a_{i,n})$ and $A_{\cdot,j} := (a_{1,j}, \dots, a_{m,j})^\top$ respectively. Further, for two subsets $I \subseteq \{1, \dots, m\}$ and $J \subseteq \{1, \dots, n\}$ let $A_{I,J} := (a_{i,j})_{i\in I;j\in J}$ denote the submatrix of $A$ consisting only of the rows from $I$ and the columns from $J$. The subvectors $A_{I,j}$ and $A_{i,J}$ are then defined as the column vector $A_{I,j} := (a_{i,j})_{i\in I}$ for $j \in J$ and the row vector $A_{i,J} := (a_{i,j})_{j\in J}$ for $i \in I$ respectively.

### 2.1 A Generic Master Problem for Column Generation

We start by introducing a generic setting for column generation by defining a master problem (MP) which is suitable for many different applications. As this master problem will take the form of a *generalized assignment problem*, we adopt the common terminology used in this context. We consider a set of *tasks* $D := \{1, \ldots, |D|\}$, a set of *agents* $I := \{|D|+1, \ldots, m\}$ and a set of *feasible plans* $P := \{1, \ldots, n\}$ with $m, n < \infty$. A plan $p \in P$ consists of several tasks $d \in D$, where $a_{d,p} \in \mathbb{Z}_+$ indicates if task $d$ is part of plan $p$ (if $a_{d,p} > 0$) and how often this task is to be executed. Further, the parameter $a_{i,p} \in \{0,1\}$ for a plan $p \in P$ and an agent $i \in I$ takes a value of 1, if agent $i$ participates in the execution of plan $p$. For most applications, it will be exactly one agent who executes the plan, thus we restrict ourselves to the case $\sum_{i \in I} a_{i,p} = 1$ for all $p \in P$ for ease of exposition. Furthermore, parameters $b_d \in \mathbb{Z}_+$, $d \in D$ and $b_i \in \mathbb{Z}_+$, $i \in I$ state how often task $d$ needs to be executed in total and how many plans agent $i$ shall execute. Finally, each plan $p \in P$ incurs some non-negative costs $c_p \in \mathbb{R}_+$ if it is chosen for execution. The column generation master problem (MP) is then given by

$$\begin{aligned}
\text{(MP)} \qquad \min_x \quad & \sum_{p \in P} c_p x_p && && (1)\\
\text{s.t.} \quad & \sum_{p \in P} a_{d,p} x_p \geqslant b_d && \forall d \in D && (2)\\
& \sum_{p \in P} a_{i,p} x_p = b_i && \forall i \in I && (3)\\
& x_p \geqslant 0 && \forall p \in P && (4)\\
& x_p \in \mathbb{Z} && \forall p \in P, && (5)
\end{aligned}$$

where the decision variables $x_p \in \mathbb{Z}_+$, $p \in P$, determine how often plan $p$ is executed. The problem contains two types of constraints, indexed by $d \in D$ and $i \in I$ respectively, with the constraint matrix $A = (a_{j,p})_{j \in D \cup I, p \in P}$ and the right-hand side $b = (b_j)_{j \in D \cup I}$. We will later also use the notation $\mathcal{A} := \{A_{\cdot,p} \in \mathbb{Z}_+^{|D|} \times \{0,1\}^{|I|} \mid p \in P\}$ for the set of columns of $A$. Constraint (2) ensures that task $d \in D$ is executed at least as often as required while Constraint (3) guarantees that each agent $i \in I$ executes exactly the required number of plans. The objective function (1) minimizes the overall cost of all executed plans.

The costs of a plan depend on the contained tasks $d \in D$ and the executing agent $i \in I$. Moreover, in many applications, the costs may also be impacted by other properties of a plan, like the order in which the tasks are executed. Therefore, we assume that the costs $c_p$ of a plan $p \in P$ are determined by a function $f\colon \mathbb{Z}_+^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}_+$, i.e.

$$c_p := f\left(A_{D,p}, A_{I,p}, w_{S,p}\right), \tag{6}$$

which takes the corresponding column of $A$ and the additional properties of the corresponding plan as arguments. Here the index set $S$ consists of criteria that affect the costs of a plan $p \in P$ and which depend specifically on the application at hand. The tuple $w_{S,p} := (w_{s,p})_{s \in S}$ contains the realizations of these properties like the order of tasks in a plan, or the scheduling of the tasks. These parameters of a plan $p \in P$ are computed as part of the solution of the pricing subproblem. In the master problem (MP), they appear only implicitly via the cost function.

**Example 2.1** (Cost function)**.** *Consider the example of the* vehicle routing problem (VRP)*, which is to be solved using a column generation approach. In this case, the tasks $d \in D$ are customers that need to be visited, and the agents $i \in I$ correspond to the vehicles that have to perform these customer visits along tours $p \in P$. Obviously, the order of the customers to be visited is an essential structural property of a tour. Using the parameters $w_{S,p}$, the costs of a tour $p \in P$ can be calculated via the linear function $f\colon \mathbb{Z}_+^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}_+$ with $f(x,y,z) \mapsto c_D^\top x + c_I^\top y + c_S^\top z$, i.e.*

$$f(A_{D,p}, A_{I,p}, w_{S,p}) = \sum_{d \in D} c_d a_{d,p} + \sum_{i \in I} c_i a_{i,p} + \sum_{s \in S} c_s w_{s,p},$$

*with $S := D \times D$ and weights $c_d, c_i, c_s \in \mathbb{R}$, $d \in D$, $i \in I$ and $s \in S$. We set $w_{(d_1,d_2),p} = 1$ if customer $d_1 \in D$ is followed by customer $d_2 \in D$ in tour $p \in P$, and $w_{(d_1,d_2),p} = 0$ otherwise. The first summand in $f$ then models costs (or gains) for visiting a customer, the second represents costs for using a vehicle, and the final summand incorporates the actual routing costs (such as working time or fuel).*

**Remark 2.2.** *The master problem (MP) remains linear even for non-linear cost functions $f\colon \mathbb{Z}_+^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}$, because $f$ itself does not appear in the objective (1), but rather its value at a column $(A_{D,p}, A_{I,p})$ with structure $w_{S,p}$, $p \in P$. Non-linear cost functions can occur, for example, if time windows must be respected in the problem, i.e. if the tasks $d \in D$ must be executed within certain time intervals. Then it is usual to penalize the delayed execution of a task in the objective function, e.g. via a quadratic penalty term to avoid single very large delays.*

Usually, the number of columns in the constraint matrix $A$ of (MP) is too large to compute them all in advance. Therefore, column generation is used to dynamically generate the columns during the solution process. The first aim is then to solve the linear relaxation of (MP). For this purpose, one typically starts by solving a restricted master problem (RMP), which contains only a small subset $\tilde{P} \subset P$ of all columns, and in which the integrality constraint (5) is neglected. It is solved in each iteration of the column generation procedure. Obviously, the optimal value of the restricted master problem (RMP) is bounded from below by 0. Assuming (RMP) is also feasible, there is an optimal dual solution $(\bar{\pi}_D, \bar{\pi}_I)$ to (RMP). Further, let $f\colon \mathbb{Z}_+^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}$ be the cost function of the columns $p \in P$ in (MP), then the reduced costs $\bar{c}_p$ for column $A_{\cdot,p}$ with $p \in P\backslash\tilde{P}$ are calculated as

$$\bar{c}_p := c_p - \sum_{d\in D} a_{d,p}\bar{\pi}_d - \sum_{i\in I} a_{i,p}\bar{\pi}_i \overset{(6)}{=} \underbrace{f\left(A_{D,p}, A_{I,p}, w_{S,p}\right)}_{\text{primal costs}} - \underbrace{\sum_{d\in D} a_{d,p}\bar{\pi}_d}_{\substack{\text{cumulative}\\ \text{dual costs of tasks}}} - \underbrace{\sum_{i\in I} a_{i,p}\bar{\pi}_i}_{\substack{\text{dual costs due to}\\ \text{allocation to agent}}}.$$

The dual costs of a column $p \in P$ result from the cumulated dual values $\bar{\pi}_d$ of the tasks $d \in D$ forming the corresponding plan. In contrast, the dual costs corresponding to the set $I$ only depend on the agent the plan is assigned to. Since we can solve one pricing subproblem for each agent $i \in I$ separately, we can consider the dual cost of each agent as a constant in the corresponding subproblem.

A new column $p \in P\backslash\tilde{P}$ can only improve the current solution of (RMP) if $\bar{c}_p < 0$. If there is no column with $\bar{c}_p < 0$, the optimal value of (RMP) equals the optimal value of the linear relaxation of (MP), cf. Chvátal (1983). It is the purpose of the pricing subproblem to check whether there are columns with negative reduced costs. It is a common approach to find new columns using an MIP formulation. A generic MIP formulation of the pricing problem for the restricted master problem (RMP) can be found in Appendix A1. In the following section, we will devise branch-and-bound column search as a novel way to solve the pricing problem, as an alternative to solving an MIP.

# 3 Branch-and-Bound Column Search

In general, it is computationally very expensive to solve the pricing problem as a mixed-integer program. Therefore, it is often attempted to exploit the problem-specific properties of the subproblem, e.g. in a cutting-plane procedure, or to solve it with other methods such as dynamic programming, constraint programming or heuristics. The idea of *branch-and-bound column search* is to find new columns for (RMP) by enumerating them within a search tree. Very importantly, it does not only return one improving column at a time as the traditional approach, but rather a whole set of improving columns whose size can be adjusted. This allows for a much more efficient solution process in many cases as the subproblem does not have to be called as often and feasible integer solutions to the master problem can be found earlier. Certainly, a plain enumeration of all possible columns $A_{\cdot,p}$, $p \in P$, is inefficient as there might be exponentially many. Thus, an important ingredient of our procedure is a strong rule to prune the search tree. In this section, we describe the details of branch-and-bound column search. Further, we show how to integrate branch-and-bound column search in a column generation framework in order to solve linear and mixed-integer programs.

## 3.1 Enumeration Trees for Column Search

The core of our column generation procedure is a search tree. Starting from an initial column (or plan, in terms of the generalized assignment problem), which corresponds to the root node $v_r$, all columns with reduced costs lower than an acceptance threshold $\theta \leqslant 0$ and that belong to feasible plans shall be generated as nodes of this tree. The acceptance threshold is used both to control the effort of finding new columns and to regulate the number of columns that should enter the restricted master problem. Note that it is necessary to set this threshold to $\theta = 0$ at some point in the procedure to obtain an exact solution algorithm for the LP relaxation of (MP).

In constructing the search tree, only a few (or even only one) column coefficients $a_{d,p}$, $d \in D$, are changed between a node and its children, such that the corresponding columns are similar with respect to their structure and their costs. The columns that are enumerated in our search tree are all selected from a box in $\mathbb{R}^{|D|+|I|}$ that contains all feasible columns. Since the number of those columns is finite, i.e. $|P| < \infty$, the following minimum and maximum values

$$a_d^{\min} := \min\left\{a_{d,p} \;\middle|\; p \in P\right\} \quad \text{and} \quad a_d^{\max} := \max\left\{a_{d,p} \;\middle|\; p \in P\right\} \tag{7}$$

exist for all $d \in D$, and therefore we can define the box of column candidates as

$$\bar{\mathcal{A}} := \left\{A_{\cdot,p} \in \mathbb{Z}^{|D|} \times \{0,1\}^{|I|} \;\middle|\; a_d^{\min} \leqslant a_{d,p} \leqslant a_d^{\max},\ \forall d \in D, p \in \bar{P}\right\},$$

where $\bar{P}$ is the corresponding index set of columns. In many applications, it is relatively easy to calculate such upper and lower bounds for the entries of $A_{D,\cdot}$ as a submatrix of $A$. Furthermore, for some problem classes it may also be necessary to determine the minimum or maximum number of tasks in a plan depending on the agent $i \in I$, i.e. $a_{d,i}^{\min} := \min\{a_{d,p} \mid p \in P, a_{i,p} = 1\}$ and $a_{d,i}^{\max} := \max\{a_{d,p} \mid p \in P, a_{i,p} = 1\}$. For ease of exposition, we use the definitions in (7). Note that the set $\bar{\mathcal{A}}$ typically also contains infeasible columns, i.e. for the feasible columns we have $\mathcal{A} \subseteq \bar{\mathcal{A}}$ and $P \subseteq \bar{P}$.

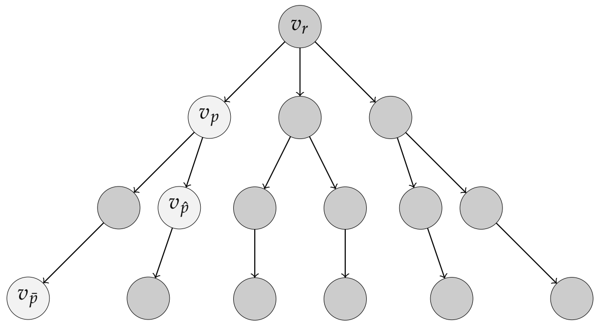

Figure 1: Illustration of an enumeration tree.

Prior to describing the essential components of branch-and-bound column search, we formally define the aforementioned search tree constructed in the process. From now on, we will refer to it as an *enumeration tree*.

**Definition 3.1** (Enumeration Tree). *Let $\bar{\mathcal{A}} \supseteq \mathcal{A}$ be a box of column candidates for master problem* (MP) *with corresponding index set $\bar{P} \supseteq P$. Then an* enumeration tree $T = (G, \circ)$ *for* (MP) *is a pair consisting of an arborescence $G = (V, E)$ and a binary operation $\circ \colon V \times 2^D \to V$, where*

(*i*) *the set of nodes is some subset*

$$V \subseteq \left\{v_p := (A_{D,p}, A_{I,p}, w_{S,p}) \;\middle|\; p \in \bar{P}\right\},$$

(*ii*) *the binary operation $\circ \colon V \times 2^D \to V$ fulfils*

$$(v_p, v_{\hat{p}}) \in E \Rightarrow \exists!\, \bar{D} \subseteq D : v_p \circ \bar{D} = v_{\hat{p}}.$$

The definition above implies that the node set $V$ of an enumeration tree $T$ contains those plans $(A_{\cdot,p}, w_{S,p})$ with $p \in \bar{P}$ which are candidates for entering the master problem (MP). In order to obtain an exact solution algorithm for (MP), we will construct the enumeration trees in such a way that all required columns for an optimal solution to the LP relaxation of (MP) are found.

Each edge $e = (v_p, v_{\hat{p}}) \in E$ of an enumeration tree $T$ (cf. Figure 1) describes a subset of rows $\bar{D} \subseteq D$ in which the coefficients change between the columns corresponding to its two incident nodes. At each node $v_p$, it is possible to specify how to get to one of its child nodes $v_{\hat{p}}$ via the binary operation $\circ$ and a subset $\bar{D} \subseteq D$. Thus, the binary operation $\circ$ can be seen as a description of the type of branching used and the choice of $\bar{D}$ describes the branching strategy.

In our enumerative solution approach, we want to set up an enumeration tree $T_i = (V_i, E_i, \circ_i)$ for each agent $i \in I$. Therefore, in Definition 3.1, an edge from a parent node $v_p$ to one of its child nodes $v_{\hat{p}}$ only changes the assignment of tasks $d \in D$ and concomitantly, of course, the structure of $w_{S,p}$ to $w_{S,\hat{p}}$. The assignment of the plans $p \in P$ to an agent, which is encoded in the partial column $A_{I,p}$ of the constraint matrix, remains constant for a given tree $T_i$. Furthermore, we assumed in Section 2.1 that $\sum_{i \in I} a_{i,p} = 1$ for all $p \in P$, so we can conclude that $A_{I,p} \in \{e_i\}_{i \in I}$ for all $p \in P$. Therefore, given an agent $i \in I$, we have

$$\forall v_p = (A_{D,p}, A_{I,p}, w_{S,p}) \in V_i : A_{I,p} = e_i.$$

We now provide a definition of the distance between a given node $v_p \in V$ and the root node $v_r \in V$ within an enumeration tree $T$.

**Definition 3.2** (Depth of a Node). *Let $T = (V, E, \circ)$ be an enumeration tree with root node $v_r \in V$ and let $v_p \in V$ be an arbitrary node in $T$. Then the* depth *$\phi(p)$ of $v_p$ in $T$ is defined as the number of edges of the unique directed path from $v_r$ to $v_p$.*

Using the notion of depth, we can order the nodes of an enumeration tree as follows. Let $T = (V, E, \circ)$ be an enumeration tree with root node $v_r \in V$ and let $v_l \in V$. Further, let $\mathcal{P} = (v_r, \ldots, v_{\bar{p}}, \ldots, v_l)$ be the unique path from $v_r$ to $v_l$. If $\phi(p) < \phi(\bar{p})$ holds for two nodes $v_p, v_{\bar{p}}$ on $\mathcal{P}$, we write $v_p \leqslant_T v_{\bar{p}}$, and we say $v_p$ is an *ascendant* of $v_{\bar{p}}$ or $v_{\bar{p}}$ is a *descendant* of $v_p$.

In the remainder, we focus on a specific class of enumeration trees, as defined next.

**Definition 3.3** (Monotonically Increasing Enumeration Tree). *An enumeration tree $T$ is called* monotonically increasing *if for any two nodes $v_p = (A_{D,p}, A_{I,p}, w_{S,p})$, $v_{\bar{p}} = (A_{D,\bar{p}}, A_{I,\bar{p}}, w_{S,\bar{p}}) \in V$ with $v_p \leqslant_T v_{\bar{p}}$ the following condition is fulfilled componentwise:*

$$A_{D,p} \leqslant A_{D,\bar{p}}.$$

In the following sections, we will describe how monotonically increasing enumeration trees can be used effectively to solve the pricing subproblem of a column generation algorithm. In order to describe an exact solution algorithm for the pricing problem, it is necessary to ensure that the method finds all plans $p \in P$ with negative reduced costs.

**Definition 3.4** (Complete Enumeration Tree). *Let $(\bar{\pi}_D, \bar{\pi}_I) \in \mathbb{R}^{|D|} \times \mathbb{R}^{|I|}$ be an optimal solution of the dual subproblem* (RDP). *Then an enumeration tree $T_i = (V_i, E_i, \circ_i)$ of an agent $i \in I$ is called* complete *if it contains all nodes $v_p = (A_{D,p}, e_i, w_{S,p}) \in V_i$ with $i \in I$ and $p \in P$ whose corresponding columns $(A_{D,p}, e_i)$ are feasible for master problem* (MP) *and which have negative reduced costs $\bar{c}_p < 0$, i.e.*

$$\{v_p = (A_{D,p}, e_i, w_{S,p}) \mid A_{\cdot,p} \in \mathcal{A},\ \bar{c}_p < 0,\ p \in P\} \subseteq V_i.$$

In order to show at the end of the column generation procedure that an optimal solution of the LP relaxation of (MP) has been found, it is necessary to verify that no further columns with negative reduced costs exist. To this end, we will use the definition of the complete enumeration tree. After all, if our column search constructs a complete enumeration tree at some point in the procedure and if this tree does not contain any columns with negative reduced costs, then we also know that the optimal solution of (RMP) found in the last iteration is also an optimal solution of the LP relaxation of (MP).

## 3.2 Determination of Lower Bounds for the Reduced Costs in a Branch

The pricing subproblem must be solved alternately with the master problem at each iteration of a column generation procedure to find new columns with negative reduced costs. Similarly, the enumeration trees $T_i = (V_i, E_i, \circ_i)$ with $i \in I$ must also be reconstructed in each iteration of the column generation procedure. In most cases, of course, it is far too time-consuming to construct the tree completely with all feasible columns which have negative reduced costs, or even only those whose reduced costs are lower than the given acceptance threshold $\theta \leqslant 0$. Consequently, it is necessary to prune certain subtrees in the course of the enumeration process for which we can prove that they do not contain any columns with reduced costs lower than $\theta$ and thus no columns that enter the master problem. The main idea is therefore to estimate at a node $v_p$ in the enumeration process by how much the reduced costs in the subtree under this node $v_p$ can still change. If we know, based on the estimate, that there are no columns with reduced costs lower than $\theta$ among the descendants of this node $v_p$, the corresponding subtree can be pruned without loosing the global optimality of the procedure.

In our method, it is important to construct monotonically increasing enumeration trees for each agent $i \in I$. Thus, for describing an edge between a parent node $v_p \in V_i$ and its child node $v_{\hat{p}} \in V_i$, we only use one concrete realization of the binary operation $\circ_i$. With this specific operation, denoted by $\oplus_i \colon V_i \times 2^D \to V_i$, a child node $v_{\hat{p}}$ is constructed from its parent node $v_p$ by increasing the number of executions of one specific task $\bar{d} \in D$ from plan $p$ to plan $\hat{p}$ by one and by adjusting the structural properties $w_{S,p}$ to $w_{S,\hat{p}}$ appropriately. In our example of the (VRP), this would mean that a tour $\hat{p}$ is constructed from another tour $p$ by adding exactly one customer $\bar{d} \in D$ to tour $p$.

For monotonically increasing enumeration trees $T_i = (V_i, E_i, \oplus_i)$, $i \in I$, we observe that it is possible to calculate the depth $\phi(p)$ of a node $v_p \in V_i$ as

$$\phi(p) = \sum_{d \in D} a_{d,p} - a_{d,r}, \tag{8}$$

with $r$ as the index of the root node $v_r = (A_{D,r}, A_{I,r}, w_{S,r}) \in V_i$.

A possibility to control the search effort for new columns is to construct not every child node $v_{\hat{p}}$ for a node $v_p \in V_i$, but only child nodes for a specifically chosen subset $\tilde{D} \subseteq D$ of row coefficients. For example, one possible criterion to determine the tasks $d \in \tilde{D} \subseteq D$ is to choose the $|\tilde{D}|$ tasks with the highest dual costs $\bar{\pi}_d$, $d \in D$. Furthermore, it is possible to modulate the effort by only generating plans up to a certain maximum number of tasks. With the chosen binary operation $\oplus$, this means that the enumeration tree $T_i$ is only constructed up to a certain search depth $\ell$. These two strategies can provide performance advantages in the overall column generation procedure as we will see later in Section 3.4. Similar considerations have already been made in Krumke et al. (2002) and Westphal and Krumke (2008) for the special case of the vehicle dispatching problem. The principle of estimating reduced costs explained there is now formalized and, especially, generalized to arbitrary optimization problems for the first time.

Now, we provide two lower bounds on the reduced cost $\bar{c}_{\bar{p}}$ of all columns $\bar{p} \in P$ that are descendants of a node $v_p \in V_i$, $p \in P$, i.e. $v_p \leqslant_{T_i} v_{\bar{p}}$. First, we consider a generic lower bound that can always be used, regardless of the concrete application under consideration, if the problem to be solved has the described generalized assignment structure (cf. Section 2.1).

**Definition 3.5** (Generic Total Maximal Gain). *Let $T_i = (V_i, E_i, \circ_i)$ with $i \in I$ be a monotonically increasing enumeration tree for the generation of new columns of master problem* (MP). *Let $\bar{\pi}$ be an optimal solution of the restricted dual problem* (RDP), *and let $a_d^{\max} := \max\{a_{d,p} \mid p \in P\}$ be an upper bound for the column entries $a_{d,p}$ for all $d \in \tilde{D}$, $p \in P$, where $\tilde{D} \subseteq D$. Then the* generic total maximal gain *of a node $v_p \in V_i$ with depth $\phi(p)$, $p \in P$, for a specific search depth $\ell$ is defined as follows:*

$$\operatorname{gtmg}(p, \ell, \tilde{D}, \bar{\pi}) := \max_{\chi \in \mathbb{Z}^{\tilde{D}}} \left\{ \sum_{d \in \tilde{D}} \bar{\pi}_d \chi_d \;\middle|\; 0 \leqslant \chi_d \leqslant (a_d^{\max} - a_{d,p}),\ d \in \tilde{D},\ \sum_{d \in \tilde{D}} \chi_d \leqslant \ell - \phi(p) \right\}.$$

The generic total maximal gain can be seen as the maximum value by which the reduced cost $\bar{c}_p$ of a column $p \in P$ at the corresponding node $v_p$ can still decrease by adding at most $\ell - \phi(p)$ additional tasks from a given subset $\tilde{D} \subseteq D$ if the primal costs would not increase at all.

Thus, with the generic total maximal gain it is now possible to calculate the aforementioned lower bound on the reduced costs $c_{\bar{p}}$ of all nodes $v_{\bar{p}}$ with $v_p \leqslant_T v_{\bar{p}}$ which have a maximum node depth of $\phi(\bar{p}) \leqslant \ell$ and which arise by solely adding tasks from the subset $\tilde{D}$.

**Theorem 3.6.** *Let $T_i = (V_i, E_i, \circ_i)$ be a monotonically increasing enumeration tree for some $i \in I$, and let $v_p, v_{\bar{p}} \in V_i$ be any two nodes with $v_p \leqslant_{T_i} v_{\bar{p}}$ and $\phi(\bar{p}) \leqslant \ell$ where $a_{d,p} = a_{d,\bar{p}}$ for all $d \in D \backslash \tilde{D}$ (i.e. node $v_{\bar{p}}$ only has additional tasks from subset $\tilde{D}$). Furthermore, the cost function $f$ is monotonically increasing, i.e. for $v_p \leqslant_{T_i} v_{\bar{p}}$ we have $f(v_{\bar{p}}) \geqslant f(v_p)$. Then a lower bound for the reduced costs $\bar{c}_{\bar{p}}$ of node $v_{\bar{p}} \in V_i$ is given by*

$$\bar{c}_{\bar{p}} \geqslant \bar{c}_p - \operatorname{gtmg}(p, \ell, \tilde{D}, \bar{\pi}).$$

*Proof.* Let $\bar{c}_{\bar{p}}$ and $\bar{c}_p$ be the reduced costs of two columns $\bar{p}, p \in P$ such that the properties of the claim are fulfilled. Then it is possible to bound the reduced costs $\bar{c}_{\bar{p}}$ from below in the following way:

$$\begin{aligned}
\bar{c}_{\bar{p}} &= c_{\bar{p}} - \sum_{d \in D} a_{d,\bar{p}} \bar{\pi}_d - \sum_{i \in I} a_{i,\bar{p}} \bar{\pi}_i \\
&\overset{(6)}{=} f(v_{\bar{p}}) - \sum_{d \in D} a_{d,\bar{p}} \bar{\pi}_d - \sum_{i \in I} a_{i,\bar{p}} \bar{\pi}_i \\
&\geqslant f(v_p) - \sum_{d \in D} (a_{d,\bar{p}} - a_{d,p}) \bar{\pi}_d - \sum_{d \in D} a_{d,p} \bar{\pi}_d - \sum_{i \in I} a_{i,p} \bar{\pi}_i \\
&= \underbrace{f(v_p) - \sum_{d \in D} a_{d,p} \bar{\pi}_d - \sum_{i \in I} a_{i,p} \bar{\pi}_i}_{=\bar{c}_p} - \sum_{d \in D} (a_{d,\bar{p}} - a_{d,p}) \bar{\pi}_d
\end{aligned}$$

$$
\begin{aligned}
&\overset{\substack{a_{d,p}=a_{d,\bar{p}} \\ \forall d \in D \setminus \tilde{D}}}{=} \quad \bar{c}_p - \sum_{d \in \tilde{D}} (a_{d,\bar{p}} - a_{d,p}) \bar{\pi}_d \\
&\geqslant \bar{c}_p - \max_{\chi \in \mathbb{Z}^{\tilde{D}}} \left\{ \sum_{d \in \tilde{D}} \bar{\pi}_d \chi_d \;\middle|\; 0 \leqslant \chi_d \leqslant (a_d^{\max} - a_{d,p}),\ d \in \tilde{D}\ \sum_{d \in \tilde{D}} \chi_d \leqslant \ell - \phi(p) \right\} \\
&= \bar{c}_p - \operatorname{gtmg}(p, \ell, \tilde{D}, \bar{\pi}).
\end{aligned}
$$

The first inequality holds, because function $f$ is monotonically increasing w.r.t. to the enumeration tree and $a_{i,\bar{p}} = a_{i,p}$ for all $i \in I$. The second inequality is correct, because from here on we no longer consider a specific node $v_{\bar{p}}$, but the node with minimal reduced costs which can be reached by adding the best tasks $d \in \tilde{D}$ with respect to the dual costs and up to a depth of $\ell$. □

The generic total maximum gain does not contain any information about the objective function $f$. The only assumption is that $f$ grows monotonically as more tasks are added to a plan. However, it is highly plausible for a master problem like (MP) with set-covering condition (3), that additional tasks in a plan at least do not reduce the total cost. This assumption that can be fulfilled easily makes the generic lower bound applicable independently of the concrete application.

In the following, a lower bound will be introduced that contains additional information about the objective function of the concrete application. This should make the new bound more effective, i.e. it should be possible to cut off larger parts of the enumeration tree than with the generic total maximum gain. However, the bound is to be derived in a generic framework so that its use can still be transferred to all applications with the generalized assignment structure. In order to calculate this lower bound, we use a family of application-specific functions $F = (F_d)_{d \in D}$ with $F_d \colon \mathbb{Z} \times \bigcup_{i \in I} V_i \to \mathbb{R}_+$ to obtain lower estimates of the costs of a task $d \in D$ within a node $v_p \in V_i$, $p \in P$, in an enumeration tree $T_i$, $i \in I$. For the vehicle routing problem from Example 2.1, we can simply choose $F_d(a_{d,\bar{p}} - a_{d,p}, v_p) = c_d \cdot (a_{d,\bar{p}} - a_{d,p})$ for each customer $d \in D$ to determine the costs exactly. In the example of the vehicle routing problem, there is an additional cost for returning to the depot after visiting the last customer, which is not represented in any of the functions $F_d$. Therefore, in general the estimation has to include further constants $C = (C_p)_{p \in P}$, with $C_p \in \mathbb{R}_+$, that cover such additional costs in a node $v_p \in \bigcup_{i \in I} V_i$, $p \in P$. The family of functions $F$ and constants $C$ has to fulfil the following bound for the primal costs:

$$
f\left(A_{D,\bar{p}}, A_{I,\bar{p}}, w_{S,\bar{p}}\right) \geqslant f\left(A_{D,p}, A_{I,p}, w_{S,p}\right) + \sum_{d \in D} F_d\left(a_{d,\bar{p}} - a_{d,p}, (A_{D,p}, A_{I,p}, w_{S,p})\right) + C_p, \tag{9}
$$

where $v_p, v_{\bar{p}} \in V_i$ such that $v_p \leqslant_{T_i} v_{\bar{p}}$, $i \in I$. Furthermore, we demand that $F_d(0, v_p) = 0$ for all $d \in D$.

If we consider the reduced cost $\bar{c}_p = c_p - \sum_{d \in D} a_{d,p} \bar{\pi}_d - \sum_{i \in I} a_{i,p} \bar{\pi}_i$ at a given node $v_p$, then for any task $\bar{d} \in D$ the following observation can be made. If $F_{\bar{d}}(1, v_p) \geqslant \bar{\pi}_{\bar{d}}$, the reduced costs will increase by adding one additional task $\bar{d}$ to the current plan at node $v_p$, as function $F_{\bar{d}}$ underestimates the cost contribution of task $\bar{d} \in D$ to the primal costs $c_p$ of column $p$. However, if $\bar{\pi}_{\bar{d}} > F_{\bar{d}}(1, v_p)$ for some $\bar{d} \in D$, then the reduced cost may also decrease if $a_{\bar{d},p}$ is increased by one. In order to calculate the next lower bound for the reduced costs, one must determine by how much the reduced costs can still change in this way when constructing a descendant of node $v_p$, i.e. how small the reduced costs $\bar{c}_{\bar{p}}$ for nodes $v_p \leqslant_{T_i} v_{\bar{p}}$ can still become by adding further tasks $d \in D$ to plan $p \in P$.

**Definition 3.7** (Total Maximal Gain). *Let $T_i = (V_i, E_i, \circ_i)$ with $i \in I$ be a monotonically increasing enumeration tree for the generation of new columns of master problem (MP). Let $F = (F_d)_{d \in D}$ with $F_d \colon \mathbb{Z} \times \bigcup_{i \in I} V_i \to \mathbb{R}_+$ further be a family of functions and $(C_p)_{p \in P}$ with $C_p \in \mathbb{R}_+$ a family of constants which together fulfil (9), let $\bar{\pi}$ be an optimal solution of the restricted dual problem (RDP), and let $a_d^{\max} := \max\{a_{d,p} \mid p \in P\}$ be an upper bound for the column entries $a_{d,p}$ for all $d \in \tilde{D}$, $p \in P$, where $\tilde{D} \subseteq D$. Then the* total maximal gain *of a node $v_p \in V_i$ with depth $\phi(p)$, $p \in P$, for a specific search depth $\ell$ is defined as follows:*

$$
\operatorname{tmg}(p, \ell, \tilde{D}, \bar{\pi}; F, C) :=
$$

$$
\max_{\chi \in \mathbb{Z}^{\tilde{D}}} \left\{ \sum_{d \in \tilde{D}} \left(\bar{\pi}_d \chi_d - F_d\left(\chi_d, v_p\right)\right) - C_p \;\middle|\; 0 \leqslant \chi_d \leqslant (a_d^{\max} - a_{d,p}),\ d \in \tilde{D},\ \sum_{d \in \tilde{D}} \chi_d \leqslant \ell - \phi(p) \right\}.
$$

The total maximal gain can again be seen as the maximum value by which the reduced cost $\bar{c}_p$ of a column $p \in P$ at the corresponding node $v_p$ can still decrease by adding at most $\ell - \phi(p)$ additional tasks from a given subset $\tilde{D} \subseteq D$. In contrast to the generic total maximum gain, primal costs are now also taken into account. With the total maximal gain it is now possible to calculate a new lower bound.

**Theorem 3.8.** *Let $T_i = (V_i, E_i, \circ_i)$ be a monotonically increasing enumeration tree for some $i \in I$, and let $v_p, v_{\bar{p}} \in V_i$ be any two nodes with $v_p \leqslant_{T_i} v_{\bar{p}}$ and $\phi(\bar{p}) \leqslant \ell$ where $a_{d,p} = a_{d,\bar{p}}$ for all $d \in D \backslash \tilde{D}$ (i.e. node $v_{\bar{p}}$ only has additional tasks from subset $\tilde{D}$). Furthermore, let $F = (F_d)_{d\in D}$ be a family of functions and $C = (C_p)_{p\in P}$ a family of constants providing lower bounds for the cost $c_p$ of a plan $p \in P$, given by a specific function $f$, as in (9). Then a lower bound for the reduced costs $\bar{c}_{\bar{p}}$ of node $v_{\bar{p}} \in V_i$ is given by*

$$\bar{c}_{\bar{p}} \geqslant \bar{c}_p - \operatorname{tmg}(p, \ell, \tilde{D}, \bar{\pi}; F, C).$$

*Proof.* Let $\bar{c}_{\bar{p}}$ and $\bar{c}_p$ be the reduced costs of two columns $\bar{p}, p \in P$ such that the properties of the claim are fulfilled. Then it is possible to bound the reduced costs $\bar{c}_{\bar{p}}$ from below in the following way:

$$\begin{aligned}
\bar{c}_{\bar{p}} &= c_{\bar{p}} - \sum_{d\in D} a_{d,\bar{p}}\bar{\pi}_d - \sum_{i\in I} a_{i,\bar{p}}\bar{\pi}_i \\
&\overset{(6)}{=} f\left(A_{D,\bar{p}}, A_{I,\bar{p}}, w_{S,\bar{p}}\right) - \sum_{d\in D} a_{d,\bar{p}}\bar{\pi}_d - \sum_{i\in I} a_{i,\bar{p}}\bar{\pi}_i \\
&\overset{a_{i,\bar{p}}=a_{i,p}}{=} f\left(\left((a_{d,\bar{p}} - a_{d,p}) + a_{d,p}\right)_{d\in D}, A_{I,p}, w_{S,\bar{p}}\right) - \sum_{d\in D}(a_{d,\bar{p}} - a_{d,p})\bar{\pi}_d - \sum_{d\in D} a_{d,p}\bar{\pi}_d - \sum_{i\in I} a_{i,p}\bar{\pi}_i \\
&\overset{(9)}{\geqslant} \sum_{d\in D} F_d\left(a_{d,\bar{p}} - a_{d,p}, v_p\right) - C_p - \sum_{d\in D}(a_{d,\bar{p}} - a_{d,p})\bar{\pi}_d \\
&\qquad + \underbrace{f\left(A_{D,p}, A_{I,p}, w_{S,p}\right) - \sum_{d\in D} a_{d,p}\bar{\pi}_d - \sum_{i\in I} a_{i,p}\bar{\pi}_i}_{=\bar{c}_p} \\
&\overset{\substack{a_{d,p}=a_{d,\bar{p}} \\ \forall d\in D\backslash\tilde{D}}}{=} \bar{c}_p + \sum_{d\in\tilde{D}} F_d\left(a_{d,\bar{p}} - a_{d,p}, v_p\right) - \sum_{d\in\tilde{D}}(a_{d,\bar{p}} - a_{d,p})\bar{\pi}_d - C_p \\
&\geqslant \bar{c}_p - \max_{\chi\in\mathbb{Z}^{\tilde{D}}}\left\{\sum_{d\in\tilde{D}}\left(\bar{\pi}_d\chi_d - F_d(\chi_d, v_p)\right) - C_p \;\middle|\; 0 \leqslant \chi_d \leqslant (a_d^{\max} - a_{d,p}),\ d \in \tilde{D}\ \sum_{d\in\tilde{D}}\chi_d \leqslant \ell - \phi(p)\right\} \\
&= \bar{c}_p - \operatorname{tmg}(p, \ell, \tilde{D}, \bar{\pi}; F, C).
\end{aligned}$$

The second inequality holds, because from here on we no longer consider a specific node $v_{\bar{p}}$, but the node with minimal reduced costs which can be reached by adding the best tasks $d \in \tilde{D}$ with respect to the reduced costs and up to a depth of $\ell$. □

Certainly, an essential ingredient for a well-performing implementation of branch-and-bound column search with total maximal gain tmg is the quality of the lower bounds calculated via functions $F$ and constants $C$. The better they estimate the difference in primal costs of the corresponding nodes, i.e. the smaller $f(v_{\bar{p}}) - f(v_p) - \sum_{d\in D} F_d(a_{d,\bar{p}} - a_{d,p}, v_p) - C_p \geqslant 0$ is, the better the bound $\bar{c}_p - \operatorname{tmg}(p, \ell, \tilde{D}, \bar{\pi}; F, C)$ can be used to prune the enumeration trees $T_i$, $i \in I$, and the fewer nodes have to be visited explicitly in the overall procedure. Another important requirement is that the auxiliary optimization problem for calculating the total maximal gain at a given node $v_p$ in an enumeration tree can be solved efficiently. There is a trade-off here, because if it is faster to enumerate the remaining descendants of $v_p$ than to compute the above bound, then it is clearly not efficient to prune the tree at this point of the enumeration process.

## 3.3 Branch-and-Bound Column Search for Monotonically Increasing Enumeration Trees

We will now introduce a branch-and-bound column search algorithm that builds monotonically increasing enumeration trees $T_i = (V_i, E_i, \oplus_i)$ for agents $i \in I$.

We only consider an algorithm where the sequence of tasks in a plan is the defining attribute of the structure $w_{S,p}$ and thus also constitutive for the cost $c_p$ of this plan. We will refer to this as a *sequence-dependent enumeration* (*SDE*). In Appendix A.2, there is an analogous description of a branch-and-bound colum search, where the order of the tasks within a plan has no effect on the costs of this plan. Algorithm 1 enumerates feasible plans $p \in P$ of a master problem (MP) based on the principle of a breadth-first search algorithm. To this end, all sequences of tasks $d \in \tilde{D}$ for a given agent $i \in I$ in a box $\bar{\mathcal{A}}$ are generated within an enumeration tree $T_i = (V_i, E_i, \oplus_i)$ until a specific search depth $\ell$ (cf. Figure 2). As input parameters for Algorithm 1, we have a subset of all tasks $\tilde{D}$ that limits the breadth of $T_i$. The search depth $\ell$ specifies up to which node depth $\phi(p)$ nodes $v_p$ with $p \in \bar{P}$ are to be enumerated. Later, we will see (cf. Section 3.4) that this can also help to improve the performance of the column generation algorithm. The other input parameters of BaB – SDE are the corresponding agent $i \in I$ for which we want to generate new plans, the bounds $a_d^{\min}$ and $a_d^{\max}$ for all $d \in \tilde{D}$ that define our column boxes $\bar{\mathcal{A}}$, the acceptance threshold $\theta \leqslant 0$ to control the number of columns that enter (MP) and finally the optimal dual solution $\bar{\pi} \in R^{|D|+|I|}$ in the current iteration of the column generation procedure.

**Algorithm 1:** BaB – SDE

```
Input  : A subset of tasks $\tilde{D} \subseteq D$, bounds for column entries $(a_d^{min}, a_d^{max})\ \forall d \in \tilde{D}$, agent $i \in I$,
         search depth $\ell$, optimal solution $\bar{\pi} \in \mathbb{R}^{|D|+|I|}$ of $(RDP)$, acceptance threshold $\theta$.
Output: An enumeration tree $T_i$ and a set of all columns $\mathcal{C}$ with reduced costs smaller than $\theta$
         and a node depth of at most $\ell$.
1  Initialize:
2      $r \leftarrow 0$, $a_{d,r} \leftarrow 0\ \forall d \in D$, $S_r \leftarrow \emptyset$.
3      $w_{S,r} \leftarrow$ Determine by $S_r$ and application context
4      $v_r \leftarrow (A_{D,r}, e_i, w_{S,r})$
5      $Q \leftarrow \{r\}$, $\mathcal{C} \leftarrow \emptyset$, $V_i \leftarrow \{v_r\}$, $E_i \leftarrow \emptyset$
6  while $Q \neq \emptyset$ do
7      Choose $p \in Q$.
8      $Q \leftarrow Q \setminus \{p\}$.
9      for $d \in \tilde{D}$ do
10         if $a_{d,p} < a_d^{max}$ then
11             $p_{\text{new}} \leftarrow p + 1$
               // Apply binary operation $v_{p_{\text{new}}} = v_p \oplus_i \{d\}$
12             $A_{D,p_{\text{new}}} \leftarrow A_{D,p}$, $S_{p_{\text{new}}} \leftarrow S_p$
13             $a_{d,p_{\text{new}}} \leftarrow a_{d,p} + 1$
14             $w_{S_o,p_{\text{new}}} \leftarrow w_{S_o,p_{\text{new}}}$
15             $w_{(d,k_p^{\max}+1),p_{\text{new}}} \leftarrow 1, \quad k_p^{\max} := \max\{k \in \{1,\dots,\ell_{\max}\} \mid \sum_{d \in \tilde{D}} w_{(d,k),p} = 1\}$
16             Choose $w_{S,p_{\text{new}}} \in \operatorname{argmin}_{w \in \mathbb{R}^{|S|}} \{f(A_{D,p_{\text{new}}}, e_i, w) \mid w_{S_o} = w_{S_o,p_{\text{new}}}\}$
17             $v_{p_{\text{new}}} \leftarrow (A_{D,p_{\text{new}}}, e_i, w_{S,p_{\text{new}}})$
               // Add new node $v_{p_{\text{new}}}$ to tree $T_i$
18             $V_i \leftarrow V_i \cup \{v_{p_{\text{new}}}\}$
19             $E_i \leftarrow E_i \cup \{(v_p, v_{p_{\text{new}}})\}$
20             $\bar{c}_{p_{\text{new}}} = f(v_{p_{\text{new}}}) - \sum_{d \in D} a_{d,p_{\text{new}}} \bar{\pi}_d - \bar{\pi}_i$
               // Check for reduced costs and feasibility
21             if $\bar{c}_{p_{\text{new}}} < \theta$ and $A_{\cdot,p_{\text{new}}} \in \mathcal{A}$ then
22                 $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_{p_{\text{new}}}\}$
23             end
               // Check for bounding due to reduced costs or node depth
24             if lb $< \theta$ then
25                 if $\phi(p_{\text{new}}) < \ell$ then
26                     $Q \leftarrow Q \cup p_{\text{new}}$
27                 end
28             end
29         end
30     end
31 end
32 return $T_i = (V_i, E_i, \oplus_i)$, $\mathcal{C}$
```

The starting point of Algorithm 1 is the root node $v_r = (A_{D,r}, e_i, w_{S,r})$, where all coefficients are set to $a_{d,r} = 0$. The set $Q$ is used, like in the breadth-first search, as a queue to apply the first-in-first-out principle. The initially empty set $\mathcal{C}$ is filled over the course of the procedure with all generated columns $A_{\cdot,p}$ that have reduced costs lower than the acceptance threshold $\theta$ and that are feasible plans for the application (i.e. $A_{\cdot,p} \in \mathcal{A}$). As long as the set $Q$ is not empty, starting from a current node $v_p$, child nodes $v_{p_{\text{new}}}$ are created by increasing the coefficients $a_{d,p}$ by 1 according to the binary operation $\oplus_i$ for each task $d \in \tilde{D}$. This is done for all tasks $d \in \tilde{D}$ for which $a_{d,p} < a_d^{\max}$ holds. Then the structure $w_{S,p_{\text{new}}}$ of the new node $v_{p_{\text{new}}}$ for $p_{\text{new}} \in \bar{P}$ must be determined.

Since we consider a sequence-dependent enumeration algorithm, the order of tasks $d \in \tilde{D}$ must be stored somehow for a node $v_p$. To this end, we introduce the subset $S_o \subseteq S$ of structural criteria (cf. Section 2.1) with which the order of tasks within a plan $p$ can be modelled. This means that for $S_o := D \times \{1, \dots, \ell_{\max}\}$ we have binary variables $w_{(d,k),p} \in \{0, 1\}$ with $(d, k) \in S_o$, where

$$w_{(d,k),p} = \begin{cases} 1, & \text{if task } d \text{ is executed at position } k \text{ in plan } p, \\ 0, & \text{otherwise.} \end{cases}$$

The first node is initialized with $w_{s,r} = 0$ for all $s \in S_o$. In Lines 14 and 15, the order of the tasks is updated according to the newly added task. That is, the sequence of tasks $w_{S_o,p_{\text{new}}}$ of the child node $v_{p_{\text{new}}}$ is formed from the sequence $w_{S_o,p}$ of the parent node by appending the corresponding task $d$ at the end of the sequence. Among all structures that contain this order, the cost-minimal structure for the node $v_{p_{\text{new}}}$ is then determined in line 16. Our experience has shown that the minimum cost structure in the application context is usually relatively easy or fast to determine and therefore this is not a crucial criterion for the performance of the method.

Based on node $v_{p_{\text{new}}}$, it is then possible to calculate the reduced costs $\bar{c}_{p_{\text{new}}}$ of the new column $p_{\text{new}} \in \bar{P}$. If node $v_{p_{\text{new}}}$ contains a feasible column for the master problem with sufficiently small reduced costs, it is included in $\mathcal{C}$. If the lower bound lb, computed as $\bar{c}_{p_{\text{new}}} - \text{gtmg}(p_{\text{new}}, \ell, \tilde{D}, \bar{\pi})$ or $\bar{c}_{p_{\text{new}}} - \text{tmg}(p_{\text{new}}, \ell, \tilde{D}, \bar{\pi}; F, C)$, is smaller than the acceptance threshold $\theta$, it is also possible that descendants of node $v_{p_{\text{new}}}$ have reduced costs smaller than $\theta$. Thus, we cannot prune the enumeration tree and node $v_{p_{\text{new}}}$ is added to $Q$. Otherwise, we know from Theorem 3.6 (respective Theorem 3.8) that the reduced costs of the descendants of $v_{p_{\text{new}}}$ are greater than or equal to $\bar{c}_{p_{\text{new}}} - \text{tmg}(p_{\text{new}}, \ell, \tilde{D}, \bar{\pi}; F, C)$ and thus also greater than or equal to the acceptance threshold, such that we can prune the tree at this node. That is, $p_{\text{new}}$ is not added to $Q$ (cf. Lines 25 – 28).

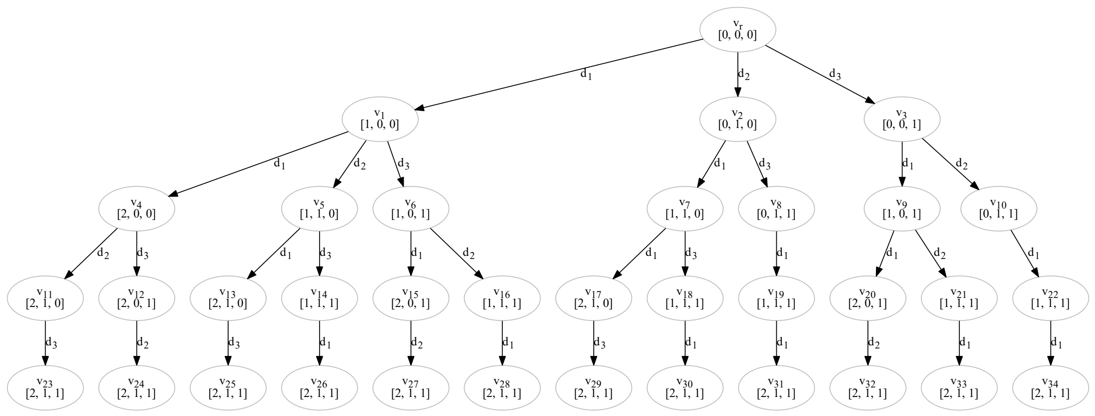

Figure 2: Illustration of an enumeration tree $T_i$ for a specific agent $i \in I$, with $\tilde{\mathcal{D}} = (d_1, d_2, d_3)$ and $a^{\max}_{d_1} = 2$, $a^{\max}_{d_2} = a^{\max}_{d_3} = 1$, constructed in the sequence-dependent enumeration algorithm BaB – SDE.

Next we show that the graph $T_i$ constructed in BaB – SDE is indeed a tree, which implies that no column is enumerated more than once. We say that two nodes $v_p, v_{\bar{p}}$ are *equal* if $A_{D,p} = A_{D,\bar{p}}$, $A_{I,p} = A_{I,\bar{p}}$ and if $w_{S_o,p} = w_{S_o,\bar{p}}$.

**Lemma 3.9.** *The graph $T_i = (V_i, E_i, \oplus_i)$ constructed by Algorithm BAB – SDE is an arborescence, i.e. for the initial node $v_r \in V_i$ in the algorithm and for any two nodes $v_p, v_{\bar{p}} \in V_i$ there are connecting paths $\mathcal{P} := (v_r, \dots, v_p)$, $\bar{\mathcal{P}} := (v_r, \dots, v_{\bar{p}})$ in $T_i$ and we have*

$$v_p = v_{\bar{p}} \Rightarrow \mathcal{P} = \bar{\mathcal{P}},$$

*and every node $v_p$ is visited exactly once via its unique path $\mathcal{P}$ in* BaB – SDE.

*Proof.* Every newly built node $v_{p_{\text{new}}}$ in Algorithm 1 is connected to one previously constructed node $v_p$. Since $v_r$ is the initial node in the algorithm, it follows inductively that every node is connected to $v_r$. It remains to show that $v_p = v_{\bar{p}}$ implies $\mathcal{P} = \bar{\mathcal{P}}$. If the paths are of different length, the sum over all coefficients $\sum_{d \in \tilde{D}} a_{d,p} \neq \sum_{d \in \tilde{D}} a_{d,\bar{p}}$ is different for $v_p$ and $v_{\bar{p}}$, such that $v_p \neq v_{\bar{p}}$. Now, suppose there are two different paths $\mathcal{P}, \bar{\mathcal{P}}$ from $v_r$ to the same node $v_p = v_{\bar{p}}$ which have the same length. Let $v_{\check{p}} \in \mathcal{P}$ and $v_{\hat{p}} \in \bar{\mathcal{P}}$ be the first nodes where $\mathcal{P}$ and $\bar{\mathcal{P}}$ differ, which also means $w_{S_o,\check{p}} \neq w_{S_o,\hat{p}}$. As a descendant of a node is formed by appending additional tasks to the existing task sequence of this node, cf. Line 15, $v_{\check{p}} \in \mathcal{P}$ and $v_{\hat{p}} \in \bar{\mathcal{P}}$ cannot have common descendants. Furthermore, we can rule out that one path is traversed more than once, because each task $d \in \tilde{D}$ is only added at most once to every node. This proves the claim. □

At some point in the column generation procedure, it is necessary to verify that BaB – SDE visits all nodes whose columns have negative reduced costs in order to preserve the exactness of the method. Therefore, we show now that Algorithm 1 can enumerate all feasible columns with negative reduced costs for a specific setting of input parameters.

**Theorem 3.10.** *Consider a master problem* (MP) *for a problem class where the sequence of tasks in a plan $p \in P$ affects its costs $c_p$, and let $\bar{\pi} \in \mathbb{R}^{|D|+|I|}$ be an optimal solution of the restricted dual problem* (RDP). *If the acceptance threshold is $\theta = 0$, the search depth is $\ell = \ell_{\max}$ and the tuple $\tilde{\mathcal{D}}$ contains all tasks $d \in D$, i.e. $\tilde{D} = D$, then Algorithm BaB – SDE constructs a complete enumeration tree $T_i = (V_i, E_i, \oplus_i)$ for an agent $i \in I$ (cf. Definition 3.4).*

*Proof.* By Lemma 3.9, the graph constructed in Algorithm BaB – SDE is a tree. This tree is also an enumeration tree, since the nodes $V_i$ generated by this algorithm are obviously columns for the master problem (MP), and by construction of the algorithm (cf. Lines 13 and 19), Condition (ii) of Definition 3.1 is also satisfied. Thus, we only have to show that it is also a complete enumeration tree. Let $(A_{D,\bar{p}}, e_i)$ be an arbitrary column with structure $w_{S,\bar{p}}$ that has negative reduced costs $c_{\bar{p}}$, where $\bar{n} := \sum_{d \in D} a_{d,\bar{p}} \leqslant \ell_{max}$ and $a_{d,\bar{p}} \leqslant a_d^{\max}$ for all $d \in D$. w.l.o.g., let the tasks $d \in D$ with $a_{d,\bar{p}} > 0$ be denominated in such a way that for the sequence of tasks in plan $\bar{p}$ we have $w_{(d_1,1),\bar{p}} = 1, w_{(d_2,2),\bar{p}} = 1, \ldots, w_{(d_{\bar{n}},\bar{n}),\bar{p}} = 1$ and $w_{s,\bar{p}} = 0$ for all other $s \in S_o = D \times \{1, \ldots, \ell_{\max}\}$.

We will show how BaB – SDE traverses the path from root node $v_r$ to $v_{\bar{p}}$ and thus prove the claim. Starting at $v_r$, $r = 0$, with an empty column $A_{\cdot,r} = ((0)_{d \in D}, e_i)$, and $w_{s,r} = 0$ for all $s \in S_o$, the algorithm iterates over all tasks in the order given by tuple $\tilde{\mathcal{D}}$. Thus, and since $a_{d_1,r} = 0 < a_d^{\max}$, it will build column $v_{p_1}$ at task $d_1$ where only $w_{(d_1,1),p_1} = 1$ in sequence $w_{S_o,p_1}$. At some later point, Algorithm BaB – SDE takes index $p_1$ corresponding to node $v_{p_1}$ from $Q$, iterates again over all $d \in D$, until $d_2$ is reached. Then also $w_{(d_2,2),p_2}$ will be set to 1, and this way node $v_{p_2}$ is built. This continues until node $v_{\bar{p}}$ has been generated. By Theorem 3.8, we have for all ascendants $v_{\hat{p}}$ of the chosen node $v_{\bar{p}}$ that $\bar{c}_{\hat{p}} - \operatorname{tmg}(\hat{p}, \ell, \tilde{D}, \bar{\pi}; F, C) < c_{\bar{p}} < 0$. Due to the choice of binary operation $\oplus$, we also know that $\phi(\hat{p}) < \phi(\bar{p}) \leqslant \ell_{\max}$, which then implies that neither the ascendant nodes $v_{\hat{p}}$ nor the chosen node $v_{\bar{p}}$ are pruned in Lines 25 – 28 of BaB – SDE. Having arbitrarily chosen column $A_{\cdot,\bar{p}}$ with structure $w_{S,\bar{p}}$, this proves the claim. □

We have now shown a very generic version of the branch-and-bound column search, namely BaB – SDE. In order to demonstrate the broad applicability of branch-and-bound column search, we discuss in Appendix A.3 a selection of problem classes (sequence dependent and sequence independent) for which it can be used to solve the pricing subproblem.

## 3.4 The Top-Level Column Generation Algorithm

In this section, we introduce the top-level algorithm that describes the interaction between a master problem (MP) and the branch-and-bound column search as solution algorithm of the pricing problem. We also discuss certain implementation techniques, to improve the performance of the column generation approach. The upper-level procedure that controls the master and the subproblem is presented in Algorithm 2.

The input of Algorithm 2 is a problem instance of master problem (MP), cf. Section 2.1. The corresponding pricing subproblem is solved by building monotonically increasing enumeration trees as shown in Section 3.3. Thus, the coefficient bounds $a_d^{\min}$ and $a_d^{\max}$ as well as the parameters search depth $\ell$ and search breadth $\beta$ with their respective step lengths $\varsigma_\ell$ and $\varsigma_\beta$ together with the maximum search length $\ell_{\max}$ are also passed to Algorithm 2. The initial values of $\ell$ and $\beta$ as well as the parameters $\varsigma_\ell$, $\varsigma_\beta$ and $\ell_{\max}$ should be chosen beforehand, depending on the application at hand.

At the initialization, the acceptance threshold $\theta$ is set to zero. A feasible initial solution to (MP) is either already given or computed by an application-specific starting heuristic and is then stored in $\tilde{\mathcal{A}}$ and the auxiliary set gen. The set $\tilde{\mathcal{A}}$, with $\tilde{\mathcal{A}} \subseteq \mathcal{A} \subseteq \bar{\mathcal{A}}$, contains all columns generated in the overall procedure, whereas gen $\subseteq \tilde{\mathcal{A}}$ includes only those columns that are enumerated within the current call of branch-and-bound column search. The enumeration process starts by computing an optimal solution $x^*_{\text{LP}}$ to the current restricted master problem (RMP) with columns $\tilde{P}$ (cf. line 7). Concomitantly, the solution $\bar{\pi} \in \mathbb{R}^{|D|+|I|}$ to the dual problem (RDP) is computed. Then the tasks $d \in D$ are sorted (by primal cost, dual cost, reduced cost or by another application-specific criterion) and the best tasks are stored in the determined order in tuple $\tilde{\mathcal{D}}$.

After that, the actual column generation takes place, using either Algorithm BaB – SDE. By passing only a subset $\tilde{\mathcal{D}}$ of tasks and by limiting the maximum number of tasks in a plan via parameter $\ell$, the tree $T_i = (V_i, E_i, \oplus_i)$, $i \in I$, is not built completely. However, the tree $T_i$ of course becomes deeper the higher $\ell$ is set in the course of the algorithm. Also, the larger the cut-off parameter $\beta$ is chosen, the wider the tree becomes. The regulation of the search breadth alone is already known through methods such as *beam search* (cf. Sabuncuoglu and Bayiz (1999)). With the branch-and-bound column search, however, both the search width and the search depth are regulated at the same time. The parameter $\theta$ is chosen adaptively to control the number of new columns in the master problem by adding only those columns $A_{\cdot,\tilde{p}}$, $\tilde{p} \in P$ that fulfil $\bar{c}_{\tilde{p}} < \theta$. In addition, the enumeration tree below a node $v_{\tilde{p}}$ with $\tilde{p} \in P$ is pruned if $\bar{c}_{\tilde{p}} - \operatorname{tmg}(\tilde{p}, \ell, \tilde{D}, \bar{\pi}; F, C) \geqslant \theta$.

**Algorithm 2:** Enumerative Column Generation Scheme

```
Input  : An instance $(A, b, c, D, I, P)$ of master problem (MP), coefficients bounds $a_d^{\min}$ and $a_d^{\max}$
         for the columns, an initial search depth $\ell$ and breadth $\beta$, step lengths $\varsigma_\ell$ and $\varsigma_\beta$ and a
         maximal search depth $\ell_{\max}$.
Output: An optimal solution to the LP relaxation of (MP) and one feasible integer solution of
         this problem instance
1  Initialize:
2      $\theta \leftarrow 0$
3      $\tilde{A}, \tilde{P} \leftarrow$ Apply starting heuristic to (MP) to generate initial columns with corresp. index set
4      gen $\leftarrow \tilde{A}$
5  while $\ell < \ell_{\max} + 1$ or $\beta < |D| + 1$ do
6      while $|\text{gen}| > 0$ or $\theta < 0$ do
7          $x^*_{\text{LP}} \leftarrow$ Solution of (RMP) with $\tilde{P}$
8          $\bar{\pi} \leftarrow$ Optimal dual solution of (RDP)
9          $\tilde{\mathcal{D}} \leftarrow$ The tuple arising from sorting tasks $d \in D$ and cutting after $\beta$-th task
10         for $i \in I$ do
11             $T_i$, gen $\leftarrow$ BaB – SDE($\tilde{\mathcal{D}}$, $(a_d^{\min}, a_d^{\max})_{d \in D}$, $i$, $\ell$, $\bar{\pi}$, $\theta$)
12             $\tilde{A} \leftarrow \tilde{A} \cup$ gen
13             Update index set $\tilde{P}$ of $\tilde{A}$
14             // Reduce acceptance threshold if too many columns were generated
15             Update $\theta$ depending on the number of generated columns in gen
16         end
17     end
18     $\ell \leftarrow \min\{\ell + \varsigma_\ell, \ell_{\max}\}$ and $\beta \leftarrow \min\{\beta + \varsigma_\beta, |D|\}$
19 end
20 $x^* \leftarrow$ Solution of (MP) with column indices $\tilde{P}$
21 return Optimal solution $x^*_{\text{LP}}$ to the LP relaxation of (MP), feasible solution $x^*$ to (MP)
```

If gen $= \varnothing$ and the acceptance threshold has been set to $\theta = 0$ in a given iteration of the inner while loop, then all columns with negative reduced costs up to the given depth and breadth have been enumerated. At this point, the search space of the tree is expanded. To this end, the parameters $\ell$ and $\beta$ are increased by step lengths $\varsigma_\ell$ and $\varsigma_\beta$, respectively. This is done until the maximal search depth $\ell_{\max}$ and the maximal search breadth $|D|$ have been reached. The algorithm terminates with a complete enumeration tree (cf. Theorem A.5 and Theorem 3.10), such that the solution $x^*_{\text{LP}}$ produced by Algorithm 2 is indeed LP-optimal.

The above approach is intended to ensure fast convergence of the solution process for the LP relaxation of (MP). To summarize, its main ideas are (i) pricing on an initially small, but dynamically increasing search space by adapting search breadth $\beta$ and search depth $\ell$, (ii) controlling the number of accepted columns on the basis of a dynamically adapted threshold $\theta$ and (iii) variation in the way the search space in the enumeration tree is scanned by changing the search criterion. Krumke et al. (2002) formulated these ideas and developed a specialized scheme for solving vehicle routing problems based on these principles, which they call *dynamic pricing control*. Its advantages are the following: (i) At the beginning of the procedure, the optimal solutions of the dual program (RDP) are updated frequently. (ii) For the same reason, the effort to find new columns is initially low when the dual variables are not yet in good shape, i.e. oscillating still a lot (see e.g. Lübbecke and Desrosiers (2005)). (iii) We retrieve a feasible integer solution early in the solution process and update it regularly. (iv) At the final stage of the procedure, when the dual information has become more reliable, the whole enumeration tree is traversed and thus the procedure is guaranteed to find LP optimal solutions in the end. In the present work, we have generalized their approach for vehicle routing to a generic framework which can now be applied to a much larger variety of problem classes.

# 4 Lexicographic Optimization

We now extend the generic column generation approach from the previous section to the case of multi-objective optimization problems. To this end, we use the principle of *lexicographic optimization* for solving problems with multiple, possibly conflicting optimization goals (see Zykina (2004) for details). We will discuss how branch-and-bound column search can be combined with this method. Further, we will show how bounds on reduced costs for pruning the enumeration tree can be calculated across the different levels of the lexicographic optimization approach. Finally, we modify Algorithm 2 such that it is suitable for solving lexicographic problems efficiently.

## 4.1 Basic Principle of the Lexicographic Optimization Approach

In a multi-objective optimization problem, different optimization goals are to be considered, each with a corresponding objective function vector $\sigma_k \in \mathbb{R}^n$, $k \in \{1, \dots, K\}$. A common approach to balance these goals is to weight the objective vectors with parameters $\omega_k \in \mathbb{R}^n$, $k \in \{1, \dots, K\}$ and then aggregate them to obtain one single objective function. This is called the *blended approach* or *weighted-sum method*, and a generic model for this approach could be formulated as follows (cf. Ehrgott (2005)):

$$
\begin{aligned}
\text{(PB)} \quad \min_{\xi} \quad & \sum_{k=1}^{K} \omega_k \sigma_k^\top \xi \\
\text{s.t.} \quad & \xi \in X.
\end{aligned}
$$

Here $\xi \in \mathbb{R}^n$ are the variables and $X \subseteq \mathbb{R}^n$ represents the feasible set. The blended approach is particularly suitable if the components of the objective function are measured in the same unit or if it is possible to convert them into one another. However, if this is not the case, lexicographic optimization is often better suited for modelling the respective multi-criteria optimization problem. In this approach, the individual objectives are optimized strictly in a predetermined, so-called *lexicographic* or *hierarchical order* (cf. Rentmeesters et al. (1996)). If we assume that the descending order of priority of the objectives $\sigma_1, \dots, \sigma_K$ is given via their index, the lexicographic optimization approach creates the following chain of optimization problems:

$$
\begin{aligned}
\text{(PL}_1) \quad \min_{\xi} \quad & \sigma_1^\top \xi \\
\text{s.t.} \quad & \xi \in X
\end{aligned}
\quad \xrightarrow{\alpha_1 \geqslant 1} \quad
\begin{aligned}
\text{(PL}_2) \quad \min_{\xi} \quad & \sigma_2^\top \xi \\
\text{s.t.} \quad & \xi \in X \\
& \sigma_1^\top \xi \leqslant \alpha_1 Z_1
\end{aligned}
\quad \xrightarrow{\alpha_2, \dots, \alpha_{m-1} \geqslant 1} \quad
\begin{aligned}
\text{(PL}_K) \quad \min_{\xi} \quad & \sigma_K^\top \xi \\
\text{s.t.} \quad & \xi \in X \\
& \sigma_k^\top \xi \leqslant \alpha_k Z_k \ \ \forall k \in L,
\end{aligned}
$$

with $L \coloneqq \{1, \dots, K-1\}$. At first we solve optimization problem (PL$_1$) with objective function $\sigma_1$. We store its optimal objective value $Z_1$ and solve the second optimization problem (PL$_2$) with objective function vector $\sigma_2$ and so on. At each lexicographic level $k \in \{2, \dots, K\}$, an additional constraint $\sigma_{k-1}^\top \xi \leqslant \alpha_{k-1} Z_{k-1}$ with $\alpha_{k-1} \geqslant 1$ is added to problem (PL$_k$) to ensure that the objective value of the higher level deteriorates at most by a factor of $\alpha_{k-1}$. In this way, it is also possible to weight the lexicographic levels against each other by calibrating the parameter $\alpha_{k-1}$, $k \in \{2, \dots, K\}$. A solution $\xi^* \in X$ is called *lexicographically optimal* (in case of minimization) for levels $k = 1, \dots, K$ if $\sigma_k^\top \xi^* \leqslant \alpha_k \sigma_k^\top \xi$, $k = 1, \dots, K-1$, and $\sigma_K^\top \xi^* \leqslant \sigma_K^\top \xi$ for all $\xi \in X$. This is a common extension to the definition of lexicographical optimality by Isermann (1982).

A main advantage of the lexicographic optimization approach compared to other methods of multi-objective optimization is that the results are easier to interpret, because the different levels in the lexicographic approach have been optimized separately. Thus, in many applications it is desirable to define a direct prioritization for the different optimization goals. Furthermore, with the lexicographic approach it is possible to avoid the parametrized weighting of optimization goals in the objective function, which can lead to numerical problems, because the parameters often vary greatly in magnitude to reflect the prioritization (see e.g. Schewe et al. (2020) in the context of penalizing violations of feasibility in penalty alternating direction methods).

## 4.2 Combination of Lexicographic Optimization and the Enumerative Approach

We now show how to use column generation via branch-and-bound column search to solve lexicographic optimization problems efficiently. The two main ideas are reusing columns between different lexicographic levels and specific bounds we derive for pruning the enumeration tree in a given level.

The master problem introduced in Section 2.1 can be seen as the first level of a lexicographic optimization approach. In general, for the $k$-th lexicographic level, the master problem can be formulated as

$$
\begin{aligned}
(\mathrm{MP}_k) \quad \min_{x} \quad & \sum_{p\in\tilde{P}} c^k_p x_p \\
\text{s.t.} \quad & \sum_{p\in\tilde{P}} a_{d,p} x_p \geqslant b_d && \forall d \in D \\
& \sum_{p\in\tilde{P}} a_{i,p} x_p = b_i && \forall i \in I \\
& \sum_{p\in\tilde{P}} c^l_p x_p \leqslant \alpha_l Z_l && \forall l \in L \qquad (10) \\
& x_p \in \mathbb{Z}_+ && \forall p \in P,
\end{aligned}
$$

with index sets $D$ and $I$, the constraint matrix $A$ and a column set $P$ as in (MP). The column costs for a column $p \in P$ in the current level $k \in \{1,\dots,K\}$ are given by $c^k_p \in \mathbb{R}$, and for the prior levels by $c^l_p \in \mathbb{R}$ with $l \in L := \{1,\dots,k-1\}$.

For the purpose of column generation, we again consider the restricted master problem (RMP$_k$), which contains only a subset of columns $\tilde{P} \subseteq P$. For the dual problem of the linear relaxation of (RMP$_k$), we have

$$
\begin{aligned}
(\mathrm{RDP}_k) \qquad \max_{\pi,\mu} \quad & \sum_{d\in D} b_d \pi_d + \sum_{i\in I} b_i \pi_i - \sum_{l\in L} \alpha_l Z_l \mu_l \\
\text{s.t.} \quad & \sum_{d\in D} a_{d,p} \pi_d + \sum_{i\in I} a_{i,p} \pi_i - \sum_{l\in L} c^l_p \mu_l \leqslant c^k_p && \forall p \in \tilde{P} \\
& \pi_d, \mu_l \geqslant 0 && \forall d \in D, \forall l \in L,
\end{aligned}
$$

where the additional dual variables $\mu_l$ with $l \in L$ are for the objective constraints (10). The optimal solution $(\bar{\pi}_D, \bar{\pi}_I, \bar{\mu}_L)$ to (RDP$_k$) is then used to calculate the reduced costs $\bar{c}^k_p$ for new columns $A_{\cdot,p}$, $p \in P\backslash\tilde{P}$, as follows:

$$
\begin{aligned}
\bar{c}^k_p &:= c^k_p - \sum_{d\in D} a_{d,p}\bar{\pi}_d - \sum_{i\in I} a_{i,p}\bar{\pi}_i + \sum_{l\in L} c^l_p \bar{\mu}_l \\
&\overset{(6)}{=} \underbrace{f_k\left(A_{D,p}, A_{I,p}\right)}_{\text{primal costs}} - \underbrace{\sum_{d\in D} a_{d,p}\bar{\pi}_d}_{\substack{\text{dual costs by} \\ \text{cumulating dual values of tasks}}} - \underbrace{\sum_{i\in I:\, a_{i,p}=1} \bar{\pi}_i}_{\substack{\text{dual costs through} \\ \text{allocation to agent}}} + \underbrace{\sum_{l\in L} f_l\left(A_{D,p}, A_{I,p}\right)\bar{\mu}_l}_{\substack{\text{dual costs due to} \\ \text{lexicographic optimization}}}, \qquad (11)
\end{aligned}
$$

where $f_k, f_l \colon \mathbb{Z}^{|D|}_+ \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}_+$ with $l \in L = \{1,\dots,k-1\}$ and $k \in \{2,\dots,K\}$ are primal cost functions as described in (6).

A decisive benefit of combining lexicographic optimization with branch-and-bound column search is that we can use the synergy effects resulting from the fact that the generated columns can be used for all levels of the approach. This means that columns generated in different levels of the lexicographic approach are stored in one common pool of columns and thus also contribute to the progress of the solution process in the other lexicographic levels. This is because the calculation of the reduced costs (11) in the $k$-th level takes into account not only the dual costs of this level, but also the dual information of all $k-1$ previous levels. To the best of our knowledge, there is no prior published work that combines enumerative column generation with lexicographic optimization in this way.

Despite the benefits arising from lexicographic optimization, it will be necessary to make the enumeration process efficient for all lexicographic levels. Therefore, we will prune the resulting enumeration trees by determining lower bounds for the reduced costs of all levels. As the generic total maximum gain can be used independently of the concrete application, objective functions are also not explicitly necessary for the calculation of the generic total maximum gain. (cf. Definition 3.5). Thus, there is no need to adapt it for lexicographic optimization. Since it is easy to see that the generic total maximal gain can also be used in its original form of Definition 3.5 to calculate lower bounds on the reduced costs for lexicographic optimization, we will not provide explicit proofs here. Thus, we only provide a generalization of Definition 3.7. That is, we introduce the total maximal gain in an arbitrary lexicographic level $k$ by incorporating the dual information from the lexicographic levels $1,\dots,k-1$.

**Definition 4.1** (Lexicographic Total Maximal Gain)**.** *Let $T^k_i = (V^k_i, E^k_i, \circ^k_i)$ with agent $i \in I$ be a monotonically increasing enumeration tree for the generation of new columns of master problem (MP$_k$) in the k-th lexicographic level. For lexicographic levels $l \in L \cup \{k\}$ with $L = \{1,\dots,k-1\}$, let $F^l = (F^l_d)_{d\in D}$ with $F^l_d \colon \mathbb{Z} \times \bigcup_{i\in I} V^l_i \to \mathbb{R}_+$*

*further be a family of functions and* $C^l = (C^l_p)_{p\in P}$ *with* $C^l_p \in \mathbb{R}_+$ *a family of constants which together fulfil* (9). *Let* $(\bar\pi_d, \bar\mu_l)$ *for* $(d,l) \in \tilde D \times L$ *be optimal solutions to the restricted dual problem* ($RDP_k$)*, and let* $a_d^{\max} := \max\{a_{d,p} \mid p \in P\}$ *be an upper bound for the column entries* $a_{d,p}$ *for all* $d \in \tilde D$, $p \in P$*, where* $\tilde D \subseteq D$. *Then the* lexikographic total maximal gain *of a node* $v_p \in V_i^k$ *in lexicographic level* $k$ *with depth* $\phi(p)$, $p \in P$*, for a specific search depth* $\ell$ *is defined as follows:*

$$\operatorname{tmg}_k\left(p,\ell,\tilde D,\bar\pi,\bar\mu;(F^l)_{l\in L\cup\{k\}},(C^l)_{l\in L\cup\{k\}}\right) :=$$
$$\max_{\chi\in\mathbb{Z}^{\tilde D}}\left\{\sum_{d\in\tilde D}\left(\bar\pi_d\chi_d - F_d^k(\chi_d,v_p) - \sum_{l\in L}\bar\mu_l F_d^l(\chi_d,v_p)\right) - C_p^k - \sum_{l\in L}\bar\mu_l C_p^l \;\middle|\; 0 \leqslant \chi_d \leqslant (a_{d_{\max}} - a_{d,p}),\, d\in\tilde D,\, \sum_{d\in\tilde D}\chi_d \leqslant \ell - \phi(p)\right\}.$$

The lexicographic total maximal gain can be interpreted as the maximum value by which the reduced costs at a specific node $v_p \in V_i^k$, $p \in \bar P$, $i \in I$, in the $k$-th lexicographic level can still decrease by adding at most $\ell - \phi(p)$ tasks $d$ from a specific subset $\tilde D$. This definition allows us to determine a lower bound on the reduced costs of all descendant nodes of $v_p$ in the enumeration tree $T_i^k$ for agent $i \in I$ in lexicographic level $k \in K$ which have a maximum node depth of $\phi(\bar p) \leqslant \ell$ and which arise by solely adding tasks from subset $\tilde D$.

**Theorem 4.2.** *Let* $T_i^k = (V_i^k, E_i^k, \circ_i^k)$ *be a monotonically increasing enumeration tree in the k-th lexicographic level for some agent* $i \in I$*, and let* $v_p, v_{\bar p} \in V_i$ *be any two nodes with* $v_p \leqslant_{T_i} v_{\bar p}$ *and* $\phi(\bar p) \leqslant \ell$*, where* $a_{d,p} = a_{d,\bar p}$ *for all* $d \in D\backslash\tilde D$ *(i.e. node* $v_{\bar p}$ *only has additional tasks from subset* $\tilde D$*). For lexicographic levels* $l \in L \cup \{k\}$ *with* $L = \{1,\dots,k-1\}$*, let* $F^l = (F_d^l)_{d\in D}$ *be a family of functions and* $C^l = (C_d^l)_{d\in D}$ *a family of constants providing lower bounds for the cost* $c_p^l$ *of a plan* $p \in P$*, given by specific functions* $f_l$*, as in* (9). *Then a lower bound for the reduced costs* $\bar c_{\bar p}^k$ *of node* $v_{\bar p} \in V_i^k$ *is given by*

$$\bar c_{\bar p}^k \;\geqslant\; \bar c_p^k \;-\; \operatorname{tmg}_k(p,\ell,\tilde D,\bar\pi,\bar\mu;(F^l)_{l\in L\cup\{k\}},(C^l)_{l\in L\cup\{k\}}).$$

*Proof.* Let $\bar c_{\bar p}^k$ and $c_p^k$ be the reduced costs of two columns $p, \bar p \in P$ in the $k$-th lexicographic level such that the properties of the claim are fulfilled. Then it is possible to bound the reduced costs $\bar c_{\bar p}^k$ from below in the following way:

$$\begin{aligned}
\bar c_{\bar p}^k &= c_{\bar p}^k - \sum_{d\in D} a_{d,\bar p}\bar\pi_d - \sum_{i\in I} a_{i,\bar p}\bar\pi_i + \sum_{l\in L} c_{\bar p}^l\bar\mu_l \\
&\overset{a_{i,\bar p}=a_{i,p}}{=} f_k\left(\left((a_{d,\bar p}-a_{d,p})+a_{d,p}\right)_{d\in D}, A_{I,p}, w_{S,\bar p}\right) - \sum_{d\in D}(a_{d,\bar p}-a_{d,p})\bar\pi_d - \sum_{d\in D} a_{d,p}\bar\pi_d - \sum_{i\in I} a_{i,p}\bar\pi_i \\
&\qquad + \sum_{l\in L}\bar\mu_l f_l\left(\left((a_{d,\bar p}-a_{d,p})+a_{d,p}\right)_{d\in D}, A_{I,p}, w_{S,\bar p}\right) \\
&\overset{(9)}{\geqslant} \sum_{d\in D}\left(F_d^k(a_{d,\bar p}-a_{d,p},v_p)\right) + C_p^k - \sum_{d\in D}(a_{d,\bar p}-a_{d,p})\bar\pi_d + \sum_{l\in L}\bar\mu_l\left(\sum_{d\in D}\left(F_d^l(a_{d,\bar p}-a_{d,p},v_p)\right)+C_p^l\right) \\
&\qquad \underbrace{+ f_k\left(A_{D,p},A_{I,p},w_{S,p}\right) - \sum_{d\in D} a_{d,p}\bar\pi_d - \sum_{i\in I} a_{i,p}\bar\pi_i + \sum_{l\in L}\bar\mu_l f_l\left(A_{D,p},A_{I,p},w_{S,p}\right)}_{\overset{(6)}{=}\bar c_p^k} \\
&= \bar c_p^k + \sum_{d\in\tilde D}\left(F_d^k(a_{d,\bar p}-a_{d,p},v_p)\right) + C_p^k - \sum_{d\in\tilde D}(a_{d,\bar p}-a_{d,p})\bar\pi_d + \sum_{l\in L}\bar\mu_l\left(\sum_{d\in\tilde D}F_d^l(a_{d,\bar p}-a_{d,p},v_p)+C_p^l\right) \\
&\geqslant \bar c_p^k - \max_{\chi\in\mathbb{Z}^{\tilde D}}\left\{\sum_{d\in\tilde D}\left(\bar\pi_d\chi_d - F_d^k\left(\chi_d,v_p\right) - \sum_{l\in L}F_d^l(\chi_d,v_p)\bar\mu_l\right) - C_p^k - \sum_{l\in L}\bar\mu_l C_p^l \;\middle|\; 0\leqslant\chi_d\leqslant(a_d^{\max}-a_{d,p}), d\in\tilde D, \sum_{d\in\tilde D}\chi_d\leqslant\ell-\phi(p)\right\} \\
&= \bar c_p - \operatorname{tmg}_k(p,\ell,\tilde D,\bar\pi,\bar\mu;(F^l)_{l\in L\cup\{k\}},(C^l)_{l\in L\cup\{k\}}).
\end{aligned}$$

The second inequality holds, because we no longer consider a specific node $v_{\bar{p}}$, but the node with minimal reduced costs. □

The above result allows us to construct an extension of Algorithm 2 that computes a lexicographically optimal solution for the LP relaxation of the multi-objective optimization problem ($\text{MP}_k$).

**Algorithm 3:** Lexicographic Column Generation Scheme

```
Input  : An instance of master problems ($\text{MP}_k$) for $k \in \{1,\dots,K\}$, coefficient bounds $a_d^{\min}$ and
         $a_d^{\max}$ for the columns, an initial search depth $\ell$ and breadth $\beta$, step lengths $\varsigma_\ell$ and $\varsigma_\beta$
         and a maximal search depth $\ell_{\max}$.
Output: A lexicographically optimal solution to the LP relaxation of ($\text{MP}_k$) for $k \in \{1,\dots,K\}$
         and a feasible integer solution of this problem instance.
1  Initialize:
2      $\theta \leftarrow 0$
3      $\tilde{\mathcal{D}}, \tilde{P} \leftarrow$ Apply starting heuristic to ($\text{MP}_1$) to generate initial columns with corresponding
         index set
4      gen $\leftarrow \tilde{\mathcal{A}}$.
5  while $\ell < \ell_{\max} + 1$ or $\beta < |D| + 1$ do
       // Execute Column Enumeration for each lexicographic level
6      for $k = 1,\dots,K$ do
7          while |gen| > 0 or $\theta < 0$ do
8              $x^*_{k,\text{LP}}$ := Solution of ($\text{RMP}_k$) with $\tilde{P}$
9              $\bar{\pi}, \bar{\mu} \leftarrow$ Optimal dual solution of ($\text{RDP}_k$)
10             $\tilde{D}, \tilde{\mathcal{D}} \leftarrow$ The tuple arising from sorting tasks $d \in D$ and cutting after $\beta$-th task.
11             for $i \in I$ do
12                 // Depending on application, choose BaB - SDE or BaB - SIDE
13                 $T_i$, gen ←BaB-Column-Search ($\tilde{\mathcal{D}}$, $(a_d^{\min}, a_d^{\max})_{d\in\tilde{D}}$, $i$, $\ell$,$(\bar{\pi},\bar{\mu})$, $\theta$)
14                 $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup$ gen
15                 Update index set $\tilde{P}$ of $\tilde{\mathcal{A}}$
16                 Update $\theta$ depending on the number of generated columns in gen
17             end
18         end
19     end
20     $\ell \leftarrow \min\{\ell + \varsigma_\ell, \ell_{\max}\}$ and $\beta \leftarrow \min\{\beta + \varsigma_\beta, |D|\}$
21 end
22 for $k = 1,\dots,K$ do
23     $x^*_k$ := Solution of ($\text{MP}_k$) with $\tilde{P}$
24 end
25 return Lexicographically optimal solution $x^*_{K,\text{LP}}$ to the LP relaxation of levels ($\text{MP}_k$) with
       $k \in \{1,\dots,K\}$, feasible solution $x^*_K$ to ($\text{MP}_K$).
```

The idea of this algorithm is to solve all lexicographic levels of the optimization problem. This requires an additional for-loop compared to Algorithm 2 which iterates over all lexicographic levels $k \in \{1,\dots,K\}$, see Lines 6–19. In this way, the branch-and-bound column search is performed for all levels $k = 1,\dots,K$ within a specific search depth $\ell$ and search breadth $\beta$. The bounds on the reduced costs in the branch-and-bound column search algorithm are calculated using the (generic) total maximal gain for the respective lexicographic level. The columns generated in each level are stored in one common set $\tilde{\mathcal{A}}$, which means that the columns generated on one lexicographic level are used for solving the restricted master problems of all other levels as well. In each column generation iteration, the LP relaxation ($\text{RMP}_k$), $k = 1,\dots,K$ is solved (cf. Line 8). For $\ell < \ell_{\max}$ and $\beta < |D|$, upper bounds are supplied to the actual optimal solutions, since not all columns in ($\text{RMP}_k$) necessary for an optimal solution are considered at this point in the algorithm. These upper bounds are then used as $Z_k$ for the next level ($\text{RMP}_{k+1}$). When the algorithm reaches the final stage, i.e. $\ell = \ell_{\max}$ and $\beta = |D|$, then the solutions from line 8 are also optimal solutions of the LP relaxations. Due to the fact that in the last lexicographic level all cost functions $f_k$, $k \in \{1,\dots,K-1\}$ are considered, cf. Section 4.1, the solution to the LP relaxation ($\text{RMP}_K$) is lexicographically optimal. In line 23, an integer solution is calculated for all lexicographic levels based on all generated columns. The heuristic solution of one lexicographic level ($\text{MP}_k$) is then used as $Z_k$ (cf. Constraint (10)) for the other levels. Although only an heuristic integer solution of a lexicographic problem is calculated this way, it has been shown in the application (cf. Section 5.4) that in many cases a lexicographic optimal solution was found.

# 5 Real-World Benchmarks

We demonstrate the efficiency of branch-and-bound column search at the hand of two real-world applications. The first one is the cutting stock problem with sequence-dependent cut losses (SDCSP). Here we focus on branch-and-bound column search as an alternative solution algorithm for the pricing problem compared to ordinary column generation via an MIP subproblem. In this application problem, we also consider only the total maximal gain tmg with application-specific functions $F$ and $C$ and not the completely generic gtmg. Second, we study a very generic version of the parallel machine scheduling problem (PMSP) to show the efficient coupling of branch-and-bound column search with lexicographic optimization. We will use PMSP to model and solve a problem arising in the optimization of intra-hospital transports, using real-world data from two different German hospitals. Here we will also compare the two pruning methods – total maximal gain and generic total maximal gain. In both applications, we observe significant improvements in solution quality compared to standard solution approaches.

Our implementation uses the Python API of Gurobi 9.1.1 (see Gurobi Optimization, Inc. (2021)) for the solution of all LPs, IPs and MIPs arising in this computational study. As far as possible we continue to use the notation from the previous sections and concretize the generic algorithmic framework devised before to problem-specific solution approaches. Their derivation takes the form of a tutorial for the application of branch-and-bound column search in practice.

## 5.1 The Cutting Stock Problem with Sequence-Dependent Cut Losses

In the *cutting stock problem with sequence-dependent cut losses* (*SDCSP*), a given set of items $D := \{1, \ldots, |D|\}$ is to be cut off from an unbounded number of larger stock items. Each item $d \in D$ has a length $l_d \in \mathbb{R}_+$, and we need to satisfy a given demand $b_d \in \mathbb{Z}_+$. Each stock item has a length of $L \in \mathbb{R}_+$. In the one-dimensional case, sequence-dependent cut losses can occur, for example, if the items are not all cut at the same angle from the stock item. For more information on the problem see Lewis et al. (2011) and Garraffa et al. (2016).

In this application, the restricted master problem ($\text{RMP}_{\text{CS}}$), and its corresponding dual problem ($\text{RDP}_{\text{CS}}$) have the following form:

$$\begin{aligned} (\text{RMP}_{\text{CS}}) \quad \min_{x} \quad & \sum_{p \in \tilde{P}} c_p x_p & & (12) \\ \text{s.t.} \quad & \sum_{p \in \tilde{P}} a_{d,p} \; x_p \geqslant b_d & \forall d \in D \quad & (13) \\ & x_p \geqslant 0 & \forall p \in \tilde{P}, & \end{aligned} \qquad \begin{aligned} (\text{RDP}_{\text{CS}}) \quad \max_{\pi} \quad & \sum_{d \in D} b_d \pi_d & \\ \text{s.t.} \quad & \sum_{d \in D} a_{d,p} \; \pi_d \leqslant c_p & \forall p \in \tilde{P} \\ & \pi_d \geqslant 0 & \forall d \in D. \end{aligned}$$

The cost of a pattern $p \in P$ is given by $c_p := (M - (L - l_p))$, where $l_p \in \mathbb{R}_+$ is the length of cutting pattern $p \in P$, $M$ is a sufficiently large number, and $a_{d,p} \in \mathbb{Z}_+$ is the number of times item $d \in D$ occurs in pattern $p \in P$. Constraint (13) ensures that the demand of all items $d \in D$ is fulfilled, and objective function (12) first minimizes the number of patterns used and then maximizes the sum of the leftovers of stock items (we use a formulation with weighted objectives as opposed to a lexicographic optimization here in order to obtain results comparable with those of the solution approach chosen in Garraffa et al. (2016)).

The sequence of items within a pattern is modelled via structure variables $w_{S,p} \in \mathbb{Z}_+$. We define $S := D^- \times D^+$ with $D^- := D \cup \{t^-\}$ and $D^+ := D \cup \{t^+\}$, where $t^-, t^+$ are artificial items with lengths $l_{t^-} := l_{t^+} := 0$ that enable the modelling of cut losses at the start and the end of each pattern. The cut losses which arise when placing two items next to each other are given by $c_{d_1,d_2} \in \mathbb{R}_+$ for all $(d_1, d_2) \in S$. Then the length $l_p$ of a pattern $p \in P$ is defined as the length of all items in this pattern together with all corresponding cut losses, i.e. $l_p := \sum_{(d_1,d_2) \in S} c_{d_1,d_2} w_{(d_1,d_2),p} + \sum_{d \in D} l_d a_{d,p}$, where $w_{(d_1,d_2),p} > 0$ if item sequence $(d_1, d_2) \in S$ is part of pattern $p$ and $w_{(d_1,d_2),p} = 0$ otherwise. The total cost of all used patterns is then calculated in objective function (12), where the individual pattern costs are given via $f^{\text{CS}} \colon \mathbb{Z}^{|D|} \times \mathbb{R}^{|S|} \to \mathbb{R}$, $(A_{D,p}, w_{S,p}) \mapsto c_p$.

As a first step in the overall solution approach, we want to solve the LP-relaxed master problem that contains all columns in $P \supset \tilde{P}$. The columns that are not yet part of ($\text{RMP}_{\text{CS}}$), i.e. $p \in P \backslash \tilde{P}$, and that are necessary for an LP-optimal solution, still have to be generated. To this end, we calculate the reduced costs of these columns via $\bar{c}_p := c_p - \sum_{d \in D} a_{d,p} \bar{\pi}_d$, with $\bar{\pi}_d$, $d \in D$, as the optimal solution to ($\text{RDP}_{\text{CS}}$). When solving the problem via branch-and-bound column search, an enumeration tree $T = (V, E, \oplus)$ is built as described in Section 3.1. In this application, the enumeration tree contains cutting patterns from a box $\bar{\mathcal{A}} \coloneqq \{A_{D,p} \in \mathbb{Z}^{|D|} \mid a_d^{\min} \leqslant a_{d,p} \leqslant a_d^{\max},\ \forall d \in D, p \in \bar{P}\}$, represented as nodes $v_p \coloneqq (A_{D,p}, w_{S,p}) \in V$ with $p \in \bar{P} \supset P$.

To cope with the large number of possible columns, an efficient pruning of the enumeration tree as derived in Section 3.2 is necessary. For SDCSP, the idea is to exclude certain patterns $p \in \bar{P}$ in the enumeration process based on their reduced costs $\bar{c}_p$ and their total length $l_p$. For this purpose, we define a family of functions $F^{\mathrm{CS}} = (F_d^{\mathrm{CS}})_{d\in D}$ to estimate the primal costs from below that arise when an item $d \in D$ is added to pattern $p \in \bar{P}$ with $F_d^{\mathrm{CS}} \colon \mathbb{Z} \times V \to \mathbb{R}$, given via $(\chi_d, (A_{D,p}, w_{S,p})) \mapsto \left(l_d + \min\left\{c_{\hat{d},d} \,\middle|\, \hat{d} \in \hat{D}_p\right\}\right) \cdot \chi_d$, where

$$\hat{D}_p \coloneqq \begin{cases} \left\{d \in D \,\middle|\, w_{(d,t^+),p} = 1 \vee a_{d,p} < a_d^{\max}\right\}, & \text{if } A_{D,p} \neq 0 \\ D^-, & \text{otherwise.} \end{cases}$$

If pattern $p$ contains at least one item, then $\hat{D}_p$ consists of the last item in $p$ and of all items that could still be added to $p$. Otherwise, if pattern $p$ is empty, then $\hat{D}_p$ consists of all items in $D^-$. The function $F_d^{\mathrm{CS}}$ bounds the cut loss from below that occurs before item $d$ and adds its length $l_d$. To represent the cut loss after the last item in $p$, we also define a family of constants $C^{\mathrm{CS}} = (C_p^{\mathrm{CS}})_{p\in\bar{P}}$ with $C_p^{\mathrm{CS}} \coloneqq -\sum_{d\in D} c_{d,t^+} w_{(d,t^+),p}$. Analogously as in (9), we can now bound the primal costs of an arbitrary descendent $v_{\bar{p}}$ of $v_p$ from below as follows:

$$\begin{aligned} f^{\mathrm{CS}}(v_{\bar{p}}) &= M - \left(L - \sum_{(d_1,d_2)\in S} c_{d_1,d_2} w_{(d_1,d_2),\bar{p}} - \sum_{d\in D} l_d a_{d,\bar{p}}\right) \\ &= M - \Bigg(L - \sum_{(d_1,d_2)\in S} c_{d_1,d_2}(w_{(d_1,d_2),\bar{p}} - w_{(d_1,d_2),p}) - \sum_{(d_1,d_2)\in S} c_{d_1,d_2} w_{(d_1,d_2),p} \\ &\qquad\qquad - \sum_{d\in D} l_d(a_{d,\bar{p}} - a_{d,p}) - \sum_{d\in D} l_d a_{d,p}\Bigg) \\ &= f^{\mathrm{CS}}(v_p) + \sum_{d\in D} l_d(a_{d,\bar{p}} - a_{d,p}) + \sum_{(d_1,d_2)\in S} c_{d_1,d_2}(w_{(d_1,d_2),\bar{p}} - w_{(d_1,d_2),p}) \\ &\geqslant f^{\mathrm{CS}}(v_p) + \sum_{d\in D^+} \left(l_d + \min\left\{c_{\hat{d},d} \,\middle|\, \hat{d} \in \hat{D}_p\right\}\right)(a_{d,\bar{p}} - a_{d,p}) - \sum_{d\in D} c_{d,t^+} w_{(d,t^+),p} \\ &= f^{\mathrm{CS}}(v_p) + \sum_{d\in D} F_d^{\mathrm{CS}}(a_{d,\bar{p}} - a_{d,p}) + C_p^{\mathrm{CS}}. \end{aligned}$$

By Theorem 3.8, we can prune all descendant nodes $v_{\bar{p}}$ of $v_p$ in the enumeration tree if

$$\begin{aligned} &\bar{c}_p - \mathrm{tmg}^{\mathrm{cut}}(p, \ell, \tilde{D}, \bar{\pi}; F^{\mathrm{CS}}, C^{\mathrm{CS}}) \geqslant 0 && (14) \\ \Leftrightarrow\quad &\bar{c}_p - \max\left\{\sum_{d\in\tilde{D}} \left(\bar{\pi}_d \chi_d - F_d^{\mathrm{CS}}(\chi_d, v_p)\right) + C_p^{\mathrm{CS}} \,\middle|\, 0 \leqslant \chi_d \leqslant a_d^{\max} - a_{d,p}, d \in \tilde{D},\ \sum_{d\in\tilde{D}} \chi_d \leqslant \ell - \phi(p)\right\} \geqslant 0, \end{aligned}$$

because then there are no more patterns with negative reduced costs in the subtree below node $v_p$.

In this application, items are cut out of a set of stock items that have all the same length $L$. The patterns become progressively longer the deeper they lie in enumeration tree $T = (V, E, \oplus)$, because at each edge we append one new item to the previous pattern via the binary operation $\oplus$. Thus, it is possible to prune the enumeration tree below an already generated pattern $p \in P$ by using a bound on the length of the patterns. More precisely, we know that all descendant patterns $\bar{p} \in \bar{P}$ of $p \in P$ for which we have $l_{\bar{p}} > L$ are infeasible. Since this is the only restriction on the feasibility of cutting patterns, this is equivalent to the statement that all feasible patterns $\bar{p} \in P$ fulfil

$$\begin{aligned} & l_{\bar{p}} = \sum_{(d_1,d_2)\in S} c_{d_1,d_2} w_{(d_1,d_2),\bar{p}} + \sum_{d\in D} l_d a_{d,\bar{p}} \leqslant L \\ \Leftrightarrow\quad & l_p - \sum_{d\in D} c_{d,t^+} w_{(d,t^+),p} + \sum_{(d_1,d_2)\in S} c_{d_1,d_2}(w_{(d_1,d_2),\bar{p}} - w_{(d_1,d_2),p}) + \sum_{d\in D} l_d(a_{d,\bar{p}} - a_{d,p}) \leqslant L \\ \Leftrightarrow\quad & \sum_{d\in D} l_d(a_{d,\bar{p}} - a_{d,p}) \leqslant \underbrace{L - l_p + \sum_{d\in D} c_{d,t^+} w_{(d,t^+),p}}_{\text{already known at node } v_p} - \underbrace{\sum_{(d_1,d_2)\in S} c_{d_1,d_2}(w_{(d_1,d_2),\bar{p}} - w_{(d_1,d_2),p})}_{(*)} \eqqcolon \hat{L}_p. \qquad (15) \end{aligned}$$

The first part of the right-hand side of inequality (15) is already known at node $v_p$ in the enumeration tree, and the second part $(*)$ can be bounded from below by $\min\{c_{d_p,d} \mid d \in D\} + \min\{c_{d,t^+} \mid d \in D\}$, where $d_p \in D$ with $w_{(d_p,t^+),p} = 1$. If we define

$$\bar{L}_p := L - l_p + \sum_{d\in D} c_{d,t^+} w_{(d,t^+),p} - \min\left\{c_{d_p,d} \,\middle|\, d \in D\right\} - \min\left\{c_{d,t^+} \,\middle|\, d \in D\right\}, \tag{16}$$

where again $d_p \in D$, with $w_{(d_p,t^+),p} = 1$, we have $\hat{L}_p \leqslant \bar{L}_p$, from which we obtain

$$\left\{\chi \in \mathbb{Z}^{|D|} \,\middle|\, \sum_{d\in D} l_d\chi_d \leqslant \hat{L}_p, 0 \leqslant \chi_d \leqslant a_d^{\max}, d \in D\right\} \subseteq \left\{\chi \in \mathbb{Z}^{|D|} \,\middle|\, \sum_{d\in D} l_d\chi_d \leqslant \bar{L}_p, 0 \leqslant \chi_d \leqslant a_d^{\max}, d \in D\right\}.$$

This means that all cutting patterns $\bar{p} \in \bar{P}$ with $v_p \leqslant_T v_{\bar{p}}$ can be pruned via (16) if we have

$$l_d > \bar{L}_p \quad \forall d \in \{d \in D \mid a_{d,p} < a_d^{\max}\}. \tag{17}$$

Note that it is also possible to combine the total maximal gain and the above lower bound to prune all descendant nodes $v_{\bar{p}}$ of $v_p$ if

$$\bar{c}_p - \max\left\{\sum_{d\in D}\left(\bar{\pi}_d - F^{\mathrm{CS}}(\chi_d, v_p)\right)\chi_d + C_p^{\mathrm{CS}} \,\middle|\, \sum_{d\in D} l_d\chi_d \leqslant \bar{L}_p, a_{d,p} \leqslant \chi_d \leqslant a_d^{\max}, d \in D\right\} \geqslant 0. \tag{18}$$

In order to compute the bound in (18), it is necessary to solve an auxiliary *knapsack problem*. This combined bound (18) is stronger than the two previously presented bounds in (14) and (17), but it is also more expensive w.r.t. to computation time. Indeed, since this knapsack problem proved to be too computationally intensive in our tests, we decided to use a weaker bound which is nevertheless stronger than the other two bounds in (14) and (17) individually and which is not much harder to compute in practice. The details of the calculation of this bound can be found in Appendix A.5.

## 5.2 Computational Results for the (SDCSP)

We now present a computational study for the sequence-dependent cutting stock problem using real-world data sets by Garraffa et al. (2016), stemming from the so-called *truss cutting problem* (*TCP*). It originates from the roofing industry, where trapezoidal profiles of the same width have to be cut from wooden boards to minimize waste. Lewis et al. (2011) show that the (TCP) is actually a special case of the (2D-CSP). However, it can also be formulated as a one dimensional (CSP), see Garraffa et al. (2016). All computations in this section have been executed on a computing cluster using compute nodes with Intel Xeon E3-1240 v6 3.7 GHz processors and 32 GB RAM, using 4 cores. For more details, seeRegionales Rechenzentrum Erlangen (2022).

In the following, we compare the quality of the solutions obtained via six different methods for solving this (SDCSP). The first, very naive method, labelled *SSH* in Table 1, is a simple heuristic that was only intended to generate feasible initial solutions. The idea of this heuristic is to fill the patterns with the required items one after the other, as they are ordered in the input data set. As soon as a pattern has exceeded the stock item length, a new stock item is opened. The method *VC* is our reimplementation of the column generation heuristic presented in Garraffa et al. (2016). Note that we could not reproduce their computational results exactly, as the authors did not describe all details of the approach. From the next method onwards, all algorithms are exact solution algorithms w.r.t. the LP relaxation of the master problem. In *MIP-Sub*, a classical column generation approach was implemented, i.e. the subproblem was solved via an MIP solver, and *SSH* is used as a starting heuristic (see Appendix A.4 for the precise MIP model). A pure branch-and-bound column search proved to be computationally too time-consuming because of the relatively high number of non-zeroes in the required columns. However, solving the subproblem with branch-and-bound column search only up to a certain maximal search depth and switching to an MIP solver beyond this depth proved to be a very effective approach overall. This procedure, called *Hybrid*, also uses *SSH* as a starting heuristic. Finally, we also tested the MIP-based subproblem and the hybrid column generation together with *VC* as a starting heuristic, which gives rise to the methods *VC+MIP* and *VC+Hybrid* respectively.

We evaluate these methods on the 1200 problem random instances from Garraffa et al. (2016), which are divided into 5 classes of 240 instances each. These classes contain instances with a fixed number of

|  | *SSH* |  | *VC* |  | *MIP-Sub* |  | *Hybrid* |  | *VC+MIP* |  | *VC+Hybrid* |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | Obj | Opt | Obj | Opt | Obj | Opt | Obj | Opt | Obj | Opt | Obj | Opt |
| 100 | 11035 | 13 | 10650 | 62 | 10298 | 201 | 10261 | 230 | 10265 | 228 | **10254** | **239** |
| 200 | 21537 | 94 | 20607 | 51 | 19933 | 219 | **19916** | **236** | 19919 | 233 | 19917 | 235 |
| 300 | 32914 | 4 | 31537 | 113 | 30496 | 211 | 30472 | 233 | 30475 | 232 | **30468** | **239** |
| 400 | 44117 | 3 | 41921 | 43 | 40849 | 217 | 40831 | 235 | 40834 | 232 | **40829** | **237** |
| 500 | 55458 | 5 | 53017 | 112 | 51399 | 217 | 51377 | 234 | 51382 | 233 | **51376** | **239** |
| $\sum$ | 165061 | 119 | 157732 | 381 | 152975 | 1065 | 152857 | 1164 | 152875 | 1158 | **152844** | **1189.0** |
| % |  | 9.9 |  | 31.8 |  | 88.8 |  | 97.0 |  | 96.5 |  | **99.1** |

Table 1: Comparison of objective function values for the solution methods *SSH*, *VC*, *MIP-Sub*, *Hybrid*, *VC+MIP* and *VC+Hybrid* for the (SDCSP) on 1200 real-world instances of the (TCP).

items to be cut out, where the number of items increases between the classes from 100 to 500. In the columns named *Obj*, we compare how many stock items were needed in the solutions produced by the different methods, summed over all instances of a class.

The columns *Opt* indicate the number of instances that were solved to optimality w.r.t. the integer formulation of the master problem. We consider here the solutions of the integer master problem $x^*$, which is calculated based on the columns generated for the optimal solution of the LP relaxation $x^*_{\text{LP}}$ (cf. the output of Algorithm 2). If the number of used stock items in the integer solution $x^*$ equals the number of used stock items in the LP solution, or if the difference of those two values is at most 1, we know that the corresponding instance has been solved to optimality.

First, it becomes obvious that the simple heuristic *SSH* produces considerably worse solutions than the much more elaborate scheme *VC*. The generic heuristic *Hybrid*, which is based on the branch-and-bound column search, delivers about 3.2 % better results regarding the objective function than *VC*, which is a heurstic specialised for the SDCSP. While *MIP-Sub* is able to solve 88.8% of the instances to optimality, the *Hybrid* scheme is even optimal for 97.0% of all instances. With regard to the objective function, this still means better results of approx. 0.1%. By using *VC* as a starting heuristic instead of *SSH*, the MIP-based pricing improves to 96.5% optimally solved instances, while the hybrid approach even solves 99.1% of instances to optimality. This demonstrates the advantage of branch-and-bound column search, which can be explained by a significantly higher number of useful columns that are generated early on than obtained with the MIP-based solution of the subproblem. Moreover, it seems that the generated columns over subsequent iterations can also be combined better with each other. A lack of possibility to combine the columns produced is a problem that occurs often with conventional column generation. In contrast, branch-and-bound column search allows to compute optimal or near-optimal integer solutions even without the provision of a good initial solution and without the need to solve the (integer) master problem via an explicit branching framework on non-integer variables.

In terms of runtimes, the two heuristics *SSH* and *VC* perform particularly well. Among the LP-exact approaches, the two methods using MIP-based pricing (*MIP-Sub* and *VC+MIP*) are faster than the methods using branch-and-bound column search. However, as we saw in Table 1, the additional solution time comes with a significantly higher number of instances solved to optimality, independent of the starting heuristic that was used. As the solution time for all methods considered were sufficiently short in the given application of this industrial planning problem, the overall cost savings are the crucial key figure. Therefore *VC+hybrid* is clearly the preferable solution algorithm. Furthermore, we tested all six approaches on the 16 real-world instances of the joinery cutting problem from Garraffa et al. (2016), with largely comparable results.

### 5.3 Problem Class – Parallel Machine Scheduling

As a second application of branch-and-bound column search, we consider the *parallel machine scheduling problem* (*PMSP*). The literature on scheduling is quite extensive. A recent survey by Allahverdi (2015) shows hundreds of papers dealing with problems of scheduling parallel machines. In the (*PMSP*), jobs $d \in D$ must be allocated to machines $i \in I$. Each job must be executed once on some machine without interruption, and each machine can only perform one job at a time. The starting time of the execution of a job $w_d \in \mathbb{R}$ should lie within a given time window $(a_d, b_d) \in \mathbb{R}^2$ with $a_d \leqslant b_d$ for all jobs $d \in D$. Every job $d \in D$ has a certain machine-independent processing time $p_d \in \mathbb{R}$, and between the execution of two jobs $d_1, d_2 \in D$ on a machine $i \in I$ there are specific set-up times $c_{d_1,d_2} \in \mathbb{R}$. Furthermore, before the first job of a schedule can be started, certain job-specific times must be calculated for the start-up of the machine, and there are also certain job-specific times for the shut-down of a machine after the execution of the last job of a schedule.

In the solution of this problem, we consider several optimization goals in a lexicographic fashion. The first and most highly prioritized goal is to minimize delays in the jobs $d \in D$ to be executed, calculated as $\max\{w_d - b_d, 0\}$. On the second level, we try to reduce the total set-up times of the machines $c_{d_1,d_2} \in \mathbb{R}$ between two jobs $d_1, d_2 \in D$. Finally, on the last lexicographic level we aim for a balanced utilization of the machines, i.e. we minimize the maximal number of jobs executed by any single machine $i \in I$. The restricted master problem for the first lexicographic level and the corresponding dual problem can be formulated as

$$
\begin{aligned}
(\mathrm{RMP}^1_{\mathrm{MS}})\quad \min_x \quad & \sum_{p\in\tilde{P}} c^1_p x_p \\
\text{s.t.} \quad & \sum_{p\in\tilde{P}} a_{d,p} x_p \geqslant 1 \quad && \forall d \in D \\
& \sum_{p\in\tilde{P}} a_{i,p} x_p = 1 && \forall i \in I \\
& x_p \geqslant 0 && \forall p \in \tilde{P},
\end{aligned}
\qquad
\begin{aligned}
(\mathrm{DP}^1_{\mathrm{MS}})\quad \max_\pi \quad & \sum_{d\in D} \pi_d - \sum_{i\in I} \pi_i \\
\text{s.t.} \quad & \sum_{d\in D} a_{d,p}\pi_d - \sum_{i\in I} a_{i,p}\pi_i \leqslant c^1_p \quad && \forall p \in \tilde{P} \\
& \pi_d \geqslant 0, && \forall d \in D \\
& \pi_i \in \mathbb{R} && \forall i \in I,
\end{aligned}
$$

with $a_{d,p} = 1$ if job $d \in D$ is part of machine schedule $p \in P$, and $a_{d,p} = 0$ otherwise. Further, $a_{i,p} = 1$ if machine $i \in I$ executes schedule $p \in P$, and $a_{i,p} = 0$ otherwise. The costs $c^1_p$ arising due to delays in a schedule $p \in P$ are calculated by the function $f^{\mathrm{MS1}} \colon \{0,1\}^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|}$, i.e. $c^1_p := f^{\mathrm{MS1}}(A_{D,p}, A_{I,p}, w_{S,p}) = \sum_{d\in D} \delta_d \cdot \max\{w_{d,p} - b_d, 0\} a_{d,p}$. The set $S$ is defined as $S := S_1 \cup S_2$, where $S_1 := D$ and $w_{d,p} \in \mathbb{R}$ is the starting time of job $d \in D$ in schedule $p$, and where $S_2 := D^- \times D^+$ models the sequence of jobs in machine schedule $p \in P$. Here $D^- := D \cup \{t^-\}$ is the set of all jobs $d \in D$ together with an artificial job $t^-$ that represents the initial state of a machine, and $D^+ := D \cup \{t^+\}$ is the set of all jobs extended by an artificial job $t^+$ that represents the final state of a machine after the execution of schedule $p$. These two artificial jobs help to model the job-specific start-up times $c_{t^-,d}$ and the job-specific shutdown times $c_{d,t^+}$, $d \in D$, of a machine within a schedule $p \in P$. The structure variables $w_{(d_1,d_2),p}$ take a value of 1 if the job sequence $(d_1, d_2) \in S_2$ is contained in schedule $p$, otherwise $w_{(d_1,d_2),p} = 0$. Finally, $\delta_d > 0$ is a factor with which possible delays of jobs $d \in D$ can be penalized individually.

Again, we use the optimal solution $\bar{\pi}$ of $(\mathrm{DP}^1_{\mathrm{MS}})$ to calculate the reduced cost of schedules $p \in P \backslash \tilde{P}$ that are not part of the restricted master problem $(\mathrm{RMP}^1_{\mathrm{MS}})$ with $\bar{c}^1_p := c^1_p - \sum_{d\in D} a_{d,p}\bar{\pi}_d - \sum_{i\in I} a_{i,p}\bar{\pi}_i$. In the (PMSP), the nodes $v_p \in V^k_i$ of each enumeration tree $T^k_i = (V^k_i, E^k_i, \oplus^k_i)$, for machine $i \in I$ and lexicographic level $k \in \{1,2,3\}$, correspond to machine schedules from a box $\bar{A} := \{(A_{D,p}, e_i) \in \{0,1\}^{|D|} \times \{0,1\}^{|I|} \mid 0 = a^{\min}_d \leqslant a_{d,p} \leqslant a^{\max}_d = 1, \forall d \in D, p \in \bar{P}\}$.

As already described in the derivation of Algorithm BaB – SDE, the idea of our solution approach is to start with an empty machine schedule at the root node $v_r$ and then to build child nodes by successively appending jobs, i.e. the additional job in the child node is executed right after the previously last job of the schedule at the parent node. For each $v_p$, let $d^{\mathrm{last}}_p := \operatorname{argmax}\{w_{d,p} \mid d \in D\}$ be the job with the latest starting time in the corresponding schedule $p$. Thus, for two schedules $p, \bar{p} \in \bar{P}$ with $v_p \leqslant_{T^k_i} v_{\bar{p}}$, we have $a_{d,\bar{p}} \geqslant a_{d,p}$ for all $d \in D$ (cf. Definition 3.3). For the starting times, we have $w_{d,\bar{p}} \geqslant w_{d^{\mathrm{last}}_p,p}$ for all $d \in D$ with $a_{d,\bar{p}} > a_{d,p}$.

In the first lexicographic level, the enumeration trees are pruned via a family of functions $F^{\mathrm{MS1}} = (F^{\mathrm{MS1}}_d)_{d\in D}$ where $F^{\mathrm{MS1}}_d \colon \{0,1\} \times \bigcup_{i\in I} V_i \to \mathbb{R}$. These functions bound the primal costs from below that arise when a new job $d \in D$ is appended to an existing machine schedule $p \in \bar{P}$. We define each function $F^{\mathrm{MS1}}_d$ via

$$(\chi_d, (A_{D,p}, A_{I,p}, w_{S,p})) \mapsto \delta_d \cdot \max\left\{w_{d^{\mathrm{last}}_p,p} + p_{d^{\mathrm{last}}_p} + s_{d^{\mathrm{last}}_p,d} - b_d, 0\right\} \chi_d, \ \forall d \in D. \tag{19}$$

For all pairs of nodes $v_{\bar{p}}, v_p \in V^1_i$ with $v_p \leqslant_{T^1_i} v_{\bar{p}}$, the family of functions $F^{\mathrm{MS1}}$ fulfils (9):

$$
\begin{aligned}
f^{\mathrm{MS1}}(v_{\bar{p}}) &= \sum_{d\in D} \delta_d \cdot \max\left\{w_{d,\bar{p}} - b_d, 0\right\} a_{d,\bar{p}} \\
&\overset{v_p \leqslant_{T^1_i} v_{\bar{p}}}{=} \sum_{d\in D} \delta_d \cdot \max\left\{w_{d,p} - b_d, 0\right\} a_{d,p} + \sum_{d\in D} \delta_d \cdot \max\left\{w_{d,\bar{p}} - b_d, 0\right\} (a_{d,\bar{p}} - a_{d,p}) \\
&\geqslant f^{\mathrm{MS1}}(A_{D,p}, A_{I,p}, w_{S,p}) + \sum_{d\in D} \delta_d \cdot \max\left\{w_{d^{\mathrm{last}}_p,p} + p_{d^{\mathrm{last}}_p} + s_{d^{\mathrm{last}}_p,d} - b_d, 0\right\} (a_{d,\bar{p}} - a_{d,p}) \\
&= f^{\mathrm{MS1}}(A_{D,p}, A_{I,p}, w_{S,p}) + \sum_{d\in D} F^{\mathrm{MS1}}_d(a_{d,\bar{p}} - a_{d,p}, v_p).
\end{aligned}
$$

This inequality on the primal costs of node $v_{\bar{p}}$ holds, because we have assumed in the definition of $F^{\text{MS1}}$ in (19) that all new jobs $d \in D$ with $a_{d,\bar{p}} > a_{d,p} = 0$ in schedule $\bar{p}$ can directly be appended after the last job $d_p^{\text{last}}$ of schedule $p$ without any other job in between. While this is correct for the (direct) child nodes of $v_p$ in an enumeration tree, it obviously underestimates the primal cost for all further descendants. Thus, it is possible to prune the enumeration tree in the first lexicographic level of the machine scheduling problem at a node $v_p$, $p \in \bar{P}$, if we have

$$\begin{aligned} &\bar{c}_p^1 - \text{tmg}^{\text{MS1}}(p, \ell, \tilde{D}, \bar{\pi}, F^{\text{MS1}}) \geqslant 0 \\ \Leftrightarrow\quad &\bar{c}_p^1 - \max_{\chi \in \mathbb{Z}^{\tilde{D}}} \left\{ \sum_{d \in D} \bar{\pi}_d \chi_d - F_d^{\text{MS1}}(\chi_d, v_p) \;\middle|\; 0 \leqslant \chi_d \leqslant 1 - a_{d,p}, d \in \tilde{D}, \sum_{d \in \tilde{D}} \chi_d \leqslant \ell - \phi(p) \right\} \geqslant 0. \end{aligned}$$

The restricted master problem of the second lexicographic level and the corresponding dual problem can be formulated as

$$\begin{aligned} (\text{RMP}^2_{\text{MS}})\quad \min\ & \sum_{p \in \tilde{P}} c_p^2 x_p \\ \text{s.t.}\ & \sum_{p \in \tilde{P}} a_{d,p} x_p \geqslant 1 \quad \forall d \in D \\ & \sum_{p \in \tilde{P}} a_{i,p} x_p = 1 \quad \forall i \in I \\ & \sum_{p \in \tilde{P}} c_p^1 x_p \leqslant \alpha_1 Z_1 \qquad (20) \\ & x_p \geqslant 0 \qquad \forall p \in \tilde{P}, \end{aligned}$$

$$\begin{aligned} (\text{DP}^2_{\text{MS}})\quad \max\ & \sum_{d \in D} \pi_d - \sum_{i \in I} \pi_i - \alpha_1 Z_1 \mu_1 \\ \text{s.t.}\ & \sum_{d \in D} a_{d,p} \pi_d + \sum_{i \in I} a_{i,p} \pi_i - c_p^1 \mu_1 \leqslant c_p^2 \quad \forall p \in \tilde{P} \\ & \pi_d, \mu_1 \geqslant 0 \qquad \forall d \in D. \end{aligned}$$

The additional constraint (20) ensures that in $(\text{RMP}^2_{\text{MS}})$ the objective function value of the first lexicographic level deteriorates by at most a factor of $\alpha_1$ compared to the first level objective function value $Z_1$. The costs $c_p^2$ are calculated by $f^{\text{MS2}}\colon \{0,1\}^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}$, i.e. $f^{\text{MS2}}(A_{D,p}, A_{I,p}, w_{S,p}) = \sum_{(d_1,d_2) \in S_2} c_{d_1,d_2} w_{(d_1,d_2),p}$ for all $p \in \bar{P}$. That is, the costs correspond to the sum of the set-up times of all jobs in a machine schedule $p$. For the reduced costs of this level, we have $\bar{c}_p^2 := c_p^2 - \sum_{d \in D} a_{d,p} \bar{\pi}_d - \sum_{i \in I} a_{i,p} \bar{\pi}_i + c_p^1 \bar{\mu}_1, \quad \forall p \in \bar{P} \backslash \tilde{P}$, with $(\bar{\pi}, \bar{\mu}_1)$ as the optimal solution to $(\text{DP}^2_{\text{MS}})$. Now we define a family of functions and a family of constants to bound the primal costs and reduced costs of nodes $v_{\bar{p}}$ from below, where $v_p \leqslant_{T_i^2} v_{\bar{p}}$. The family of bounding functions $F^{\text{MS2}} = (F_d^{\text{MS2}})_{d \in D}$ with $F_d^{\text{MS2}}\colon \{0,1\} \times \bigcup_{i \in I} V_i^2 \to \mathbb{R}$ is defined via $(\chi_d, (A_{D,p}, A_{I,p}, w_{S,p})) \mapsto \min\left\{ c_{\hat{d},d} \;\middle|\; \hat{d} \in \hat{D}_p \right\} \chi_d$, $\forall d \in D$, where

$$\hat{D}_p := \begin{cases} \left\{ d_p^{\text{last}} \right\} \cup \left\{ d \in D \;\middle|\; a_{d,p} = 0 \right\}, & \text{if } A_{D,p} \neq 0 \\ D^-, & \text{otherwise.} \end{cases}$$

If schedule $p$ contains at least one job, then $\hat{D}_p$ is the set that consists of the last item in $p$ as well as all jobs that could still be appended to $p$. Otherwise, if pattern $p$ is empty, then $\hat{D}_p$ equals $D^-$.

The family of constants $C^{\text{MS2}} = (C_p^{\text{MS2}})_{p \in \bar{P}}$ is defined as

$$C_p^{\text{MS2}} := \min\{ c_{\hat{d},t^+} \mid \hat{d} \in \hat{D}_p \} - c_{d_p^{\text{last}},t^+} \leqslant c_{d_{\bar{p}}^{\text{last}},t^+} - c_{d_p^{\text{last}},t^+},$$

which bounds the difference of the shut-down times between schedule $\bar{p}$ and its ascendant $p$ from below. If we have two nodes $v_{\bar{p}}$, $v_p$ in the enumeration tree with $v_p \leqslant_{T_i^2} v_{\bar{p}}$, the family of functions $F^{\text{MS2}}$ and the family of constants $C^{\text{MS2}}$ fulfils inequality (9):

$$\begin{aligned} f^{\text{MS2}}(v_{\bar{p}}) &= \sum_{(d_1,d_2) \in S_2} c_{d_1,d_2} w_{(d_1,d_2),\bar{p}} \\ &\overset{v_p \leqslant_{T_i^2} v_{\bar{p}}}{=} \sum_{(d_1,d_2) \in S_2} c_{d_1,d_2} w_{(d_1,d_2),p} + \sum_{(d_1,d_2) \in S_2} c_{d_1,d_2} (w_{(d_1,d_2),\bar{p}} - w_{(d_1,d_2),p}) \\ &\geqslant f^{\text{MS2}}(v_p) + \sum_{d \in D} \min\left\{ c_{\hat{d},d} \;\middle|\; \hat{d} \in \hat{D}_p \right\} (a_{d,\bar{p}} - a_{d,p}) + \min\left\{ c_{\hat{d},t^+} \;\middle|\; \hat{d} \in \hat{D}_p \right\} - c_{d_p^{\text{last}},t^+} \\ &= f^{\text{MS2}}(v_p) + \sum_{d \in D} F_d^{\text{MS2}}(a_{d,\bar{p}} - a_{d,p}) + C_p^{\text{MS2}}. \end{aligned}$$

This inequality is valid, because we have defined the approximation functions $F_d^{\text{MS2}}$ such that every job in machine schedule $\bar{p}$ that is not already part of schedule $p$ contributes with the minimum set-up time w.r.t. the remaining possible preceding jobs $\hat{d} \in \hat{D}_p$ at node $v_p \in V_i^2$. Thus, we can prune the enumeration tree for this lexicographic level at node $v_p$, $p \in \bar{P}$, if we have

$$\begin{aligned}
&\bar{c}_p^2 - \text{tmg}^{\text{MS2}}(p, \ell, \tilde{D}, \bar{\pi}, \bar{\mu}_1; F^{\text{MS1}}, F^{\text{MS2}}, C^{\text{MS2}}) \geqslant 0 \\
\Leftrightarrow \quad & \bar{c}_p^2 - \max_{\chi \in \mathbb{Z}^D} \left\{ \sum_{d \in \tilde{D}} \bar{\pi}_d \chi_d - \sum_{d \in \tilde{D}} F_d^{\text{MS2}}(\chi_d, v_p) - C_p^{\text{MS2}} - \bar{\mu}_1 \sum_{d \in \tilde{D}} F_d^{\text{MS1}}(\chi_d, v_p) \right. \\
& \qquad \left. \middle| \; 0 \leqslant \chi_d \leqslant 1 - a_{d,p},\, d \in \tilde{D},\, \sum_{d \in \tilde{D}} \chi_d \leqslant \ell - \phi(p) \right\} \geqslant 0.
\end{aligned}$$

Finally, the restricted LP relaxation of the master problem on the third lexicographic level and the corresponding dual problem are as follows:

$$\begin{aligned}
(\text{RMP}_{\text{MS}}^3) \quad \min \quad & L \\
\text{s.t.} \quad & \sum_{p \in \tilde{P}} a_{d,p} x_p \geqslant 1 && \forall d \in D \\
& \sum_{p \in \tilde{P}} a_{i,p} x_p = 1 && \forall i \in I \\
& \sum_{p \in \tilde{P}} c_p^1 x_p \leqslant \alpha_1 Z_1 \\
& \sum_{p \in \tilde{P}} c_p^2 x_p \leqslant \alpha_2 Z_2 && (21) \\
& \sum_{p \in \tilde{P}} c_p^3 x_p \leqslant L \quad \forall i \in I && (22) \\
& x_p \geqslant 0 && \forall p \in \tilde{P},
\end{aligned}$$

$$\begin{aligned}
(\text{DLP}_{\text{MS}}^3) \quad \max \quad & \sum_{d \in D} \pi_d - \sum_{i \in I} \pi_i - \alpha_1 Z_1 \mu_1 - \alpha_2 Z_2 \mu_2 \\
\text{s.t.} \quad & \sum_{d \in D} a_{d,p} \pi_d + \sum_{i \in I} a_{i,p} \pi_i \\
& \quad - c_p^1 \mu_1 - c_p^2 \mu_2 - \sum_{i \in I} a_{i,p} c_p^3 \lambda_i \leqslant 0 && \forall p \in \tilde{P} \\
& \sum_{i \in I} \lambda_i \leqslant 1 \\
& \pi_d, \lambda_i, \mu_1, \mu_2 \geqslant 0 && \forall d \in D, \forall i \in I,
\end{aligned}$$

with an additional constraint (21) in $(\text{RMP}_{\text{MS}}^3)$ to ensure that the objective function value of the second lexicographic level $Z_2$ deteriorates at most by a factor of $\alpha_2$. The costs $c_p^3$ of a schedule $p$ on the third level are calculated with function $f^{\text{MS3}}\colon \mathbb{Z}^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|} \to \mathbb{R}$, via $(\chi_d, e_i, x_S) \mapsto \sum_{d \in D} \chi_d$, which counts the number of jobs $d$ in machine schedule $p$. The overall goal of the third lexicographic level is to minimize the maximum number of jobs allocated to any single machine. To this end, we further need constraints (22) and a dedicated variable $L$ in $(\text{RMP}_{\text{MS}}^3)$ to model the maximum number of executed tasks of all chosen machine schedules $p \in \tilde{P}$. In the objective function, $L$ is minimized in order to find a subset of machine schedules with an even distribution of jobs.

For the reduced costs on the third lexicographic level, we have

$$\bar{c}_p^3 \coloneqq \bar{\lambda}_i c_p^3 - \sum_{d \in D} a_{d,p} \bar{\pi}_d - \sum_{i \in I} a_{i,p} \bar{\pi}_i + \bar{\mu}_1 c_p^1 + \bar{\mu}_2 c_p^2.$$

Since the costs $c_p^3$ of a schedule on the third lexicographic level do not depend on structural variables $w_{S,p}$ and thus can be calculated with small computational effort, we can directly specify the lower bound in the third lexicographic step as

$$\begin{aligned}
&\bar{c}_p^3 - \text{tmg}^{\text{MS3}}(p, l, \tilde{D}, \bar{\pi}, \bar{\mu}, \bar{\lambda}; F^{\text{MS1}}, F^{\text{MS2}}, C^{\text{MS2}}) \\
= \; & \bar{c}_p^3 - \max_{\chi \in \mathbb{Z}^D} \left\{ \sum_{d \in \tilde{D}} \bar{\pi}_d \chi_d - \bar{\lambda}_i \sum_{d \in \tilde{D}} \chi_d - \bar{\mu}_1 \sum_{d \in \tilde{D}} F_d^{\text{MS1}}(\chi_d, v_p) - \bar{\mu}_2 \left( \sum_{d \in \tilde{D}^+} F_d^{\text{MS2}}(\chi_d, v_p) + C_p^{\text{MS2}} \right) \right. \\
& \qquad \left. \middle| \; 0 \leqslant \chi_d \leqslant 1 - a_{d,p},\, d \in \tilde{D},\, \sum_{d \in \tilde{D}} \chi_d \leqslant \ell - \phi(p) \right\}.
\end{aligned}$$

The algorithm for the calculation of the total maximal gain for all three lexicographic levels is described in detail in Appendix A.7.

## 5.4 Computational Results for the (PMSP)

We now demonstrate the performance of branch-and-bound column search for the parallel machine scheduling problem (PMSP). To this end, we use real-world data from an intra-logistic transportation problem arising in hospitals. The data we used for our evaluation was kindly provided by our industrial partner OrgaCard Siemantel & Alt GmbH, a software provider for logistics planning in medical facilities.

In the transportation problem of interest, intra-hospital transports must be allocated optimally to transport employees. This application can be modelled as a (PMSP), where the transporters represent the machines and the transports stand for the jobs to be executed. The processing times correspond to the durations for the execution of a transport (i.e. to bring the patient or material from the initial location to the target location), and the set-up times arise from the travelling time of the employees from the target location of their last job to the initial location of their next job. The sequence dependency of the set-up times reflects these lead paths and the corresponding travel times between the transports.

The optimization goals which are important for a hospital operator correspond to the lexicographic levels of the multi-objective (PMSP) described in Section 5.3.

When solving the intra-hospital transport optimization problem the required computation time is also a crucial factor. Given the medical necessity of a fast execution of certain patient transports, such as medical emergencies, it is required that newly incoming orders are allocated by the system at very short notice (within 60 seconds in the hospitals we consider). Small time lags must always be anticipated between the emergence of a transport and the start of the calculation of the optimization system as well as between the moment when the optimization is finished and the time when the notification of the transport order arrives at the employee. Thus, we demand that all results are calculated within a time limit of 55 seconds.

In total, we evaluated 699 instances from two different hospitals, which are customers of our industrial partner OrgaCard. The instances were divided into 3 classes according to the number of transports they were containing: small instances have $< 5$ transports, medium instances $< 10$ transports and large instances $\geqslant 10$ transports. All computations in this section have been executed on a personal computer with Intel Core i7-9750H 2.6GHz processors and 32 GB RAM, using 6 cores.

### 5.4.1 Comparison of Column Generation with Branch-and-Bound Column Search and Conventional Column Generation

Typically, column generation (CG) for the LP relaxation of the master problem only finds fractional optimal solutions. Nevertheless, it is possible to obtain an integer solution afterwards by solving the original integer master problem restricted to the columns generated for the solution of its LP relaxation (cf. solution $x^*$ in Algorithm 2). This approach is also referred to as restricted master heuristic, cf. Joncour et al. (2010). From now on, we call the integer solution determined that way the *IP solution* and the solution to the LP relaxation $x^*_{\mathrm{LP}}$ is denoted with the *LP solution*. The corresponding objective function values are called *IP solution value* and *LP solution value*. If the LP-Solution is found by Algorithm 2 within the time limit of 55 seconds, it is LP-optimal, and of course it also forms a lower bound to the IP-Solution.

In Table 2 we compare the LP solution obtained by branch-and-bound column search, calculated as in Algorithm 2, with an LP solution computed by conventional column generation, i.e. the subproblem is solved by an MIP solver (see Appendix A.6 for the precise MIP formulation). To this end, we calculate for each instance the relative difference of all three objective functions as follows:

$$\text{LP difference} := \frac{\text{LP solution value by conventional CG} - \text{LP solution value by BaB-SDE CG}}{\text{LP solution value by conventional CG}}.$$

The first column of Table 2 lists the three instance classes considered, and the second column contains the number of instances in the corresponding class. Columns 3 to 5 show the arithmetic mean of the LP difference across all instances for objective functions 1 to 3, and the last two columns show the number of instances for which the corresponding method could find a provably optimal LP solution, i.e. the algorithm found the optimal solution within the time limit of 55 seconds. Finally, the last row summarizes the results for all instance classes and provides the arithmetic mean of the LP difference and the number of provably optimal solutions across all 645 instances.

We see that for the class of small instances, the arithmetic mean of the LP differences is zero, which means that both methods perform equally well here. Furthermore, the number of optimal LP solutions for the small instances is the same for the two solution approaches. For 28 instances of the medium class, the conventional column generation did not terminate within the time limit of 55 seconds. With the Branch-and-Bound Column Search, this was only the case for 3 instances. This is also accompanied by a slight improvement with regard to objective functions 1 and 2. The benefit of the branch-and-bound column search becomes even more apparent as the size of the instances increases: it achieved

| Comparison of the LP solution for conventional CG and CG with BaB-SDE | | | | | | |
|---|---|---|---|---|---|---|
| Inst. Class | Inst. no. | Objective 1 | Objective 2 | Objective 3 | Opt. conv. | Opt. BaB |
| small | 313 | 0.00% | 0.00% | 0.00% | 313 | 313 |
| medium | 166 | 0.37% | 0.18% | -0.02% | 138 | **163** |
| large | 220 | 8.41% | 19.02% | 1.73% | 27 | **34** |
| $\sum$ | 699 | 2.73% | 6.03% | 0.54% | 478 | **510** |

Table 2: Comparison of objective function values of the LP solution for conventional column generation and column generation via branch-and-bound column search.

significantly better LP solutions for the large instances. Furthermore, it also found optimal LP solutions for 7 more instances than conventional column generation within the time limit.

Tables 3 - 5 show the comparison of the integer solutions (IP solution), for the conventional column generation and the column generation with branch-and-bound column search applying the generic total maximal gain (cf. Definition 3.5) respective the total maximal gain that uses application-specific functions *F* and *C* (cf. Definition 3.7). The first two columns of the tables list the instance classes and the number of instances that the classes contain. In columns 3 to 5, the absolute difference (conventional CG - CG with BaB-SDE|gtmg, conventional CG - CG with BaB-SDE|tmg and CG with BaB-SDE|gtmg - CG with BaB-SDE|tmg) and in columns 6 to 9 the relative difference. The relative improvement was calculated by relating the absolute improvement to the difference between the above mentioned minuend and a primal lower bound, i.e. conventional CG - primal lb or CG with BaB-SDE/gtmg - primal lb. This makes it easier to quantify the potential for improvement.

| Comparison of the IP solution for CG with BaB-SDE using gtmg and conventional CG | | | | | | | |
|---|---|---|---|---|---|---|---|
| Instance | | Absolute difference | | | Relative difference | | |
| Class | No. | Object. 1 | Object. 2 | Object. 3 | Object. 1 | Object. 2 | Object. 3 |
| small | 313 | 0.00 | 0.00 | 0.00 | 0.00% | 0.00% | 0.00% |
| medium | 166 | 12.30 | 5.90 | 0.04 | 0.96% | 1.22% | 1.51% |
| large | 220 | 256.27 | 249.85 | 0.03 | 7.02% | 13.57% | 1.52% |
| $\sum$ | 699 | 83.58 | 80.04 | 0.02 | 2.44% | 4.56% | 0.83% |

Table 3: Comparison of objective function values for conventional column generation and column generation with branch-and-bound column search using the generic total maximal gain for pruning the enumeration trees.

In Table 3 we see that for the small instances there is no difference between conventional column generation and column generation with branch-and-bound column search and the generic tmg. For the medium instances, slight improvements could be achieved with the generic pruning approach. This effect was then enhanced for the large, more difficult-to-solve instances. This result shows that the BaB column search can compete with other pricing methods for column generation even in a completely generic approach.

The evaluation of the objective functions in Table 4 shows that for the small and medium instances there is not much difference between the application of the generic total maximal gain and the total max-

| Comparison of the IP solution for CG with BaB-SDE using tmg and CG with BaB-SDE using gtmg | | | | | | | |
|---|---|---|---|---|---|---|---|
| Instance | | Absolute difference | | | Relative difference | | |
| Class | No. | Object. 1 | Object. 2 | Object. 3 | Object. 1 | Object. 2 | Object. 3 |
| small | 313 | 0.00 | 0.00 | 0.00 | 0.00% | 0.00% | 0.00% |
| medium | 166 | 0.05 | 0.05 | 0.00 | 0.01% | 0.01% | 0.00% |
| large | 220 | 692.68 | 501.86 | 0.00 | 9.16% | 18.57% | 2.12% |
| $\sum$ | 699 | 218.02 | 157.97 | 0.00 | 2.89% | 5.85% | 0.67% |

Table 4: Comparison of objective function values for column generation with branch-and-bound column search using the application specific total maximal gain and using the generic total maximal gain for pruning the enumeration trees.

| Comparison of the IP solution for CG with BaB-SDE using tmg and conventional CG | | | | | | | |
|---|---|---|---|---|---|---|---|
| Instance | | Absolute difference | | | Relative difference | | |
| Class | No. | Object. 1 | Object. 2 | Object. 3 | Object. 1 | Object. 2 | Object. 3 |
| small | 313 | 0.00 | 0.00 | 0.00 | 0.00% | 0.00% | 0.00% |
| medium | 166 | 12.35 | 5.96 | 0.04 | 0.97% | 1.23% | 1.51% |
| large | 220 | 948.95 | 751.71 | 0.03 | 15.86% | 31.14% | 3.35% |
| $\sum$ | 699 | 301.60 | 238.01 | 0.02 | 7.94% | 10.09% | 1.10% |

Table 5: Comparison of objective function values for conventional column generation and column generation with branch-and-bound column search using the generic total maximal gain for pruning the enumeration trees.

imal gain with application specific functions $F$ and $C$ for pruning the enumeration trees. Approximately the same results are obtained for both classes of instances. For the large instances, the advantage of using application-specific information when pruning the enumeration trees becomes obvious, as considerably better results could be achieved here.

Table 5 shows that for the class of small instances, there is not yet a difference in the IP solutions computed by the two approaches, conventional CG and CG with BaB-SDE using the total maximal gain with application-specific functions $F$ and $C$. However, column generation with BaB-SDE performs slightly better for the medium-sized instances. The crucial instances are once again those of the large class. There, the branch-and-bound column search with the application-specific pruning achieves significant improvements compared to conventional column generation. In conclusion, it seems that branch-and-bound column search generates not only more columns but also columns that are easier to combine in an integer solution. This allows the corresponding restricted integer master problem to find significantly better IP solutions, especially for large instances. Furthermore, the results demonstrate that the branch-and-bound column search can achieve good results even with the generic pruning scheme. However, as one would expect, the results can be improved even further if application-specific information is included in the calculation of the bounds.

Table 6 shows a comparison of the solution times for the three column generation methods under consideration. The first two columns again contain information on the instance classes. Columns 3 to 6 show the solution times of conventional CG, columns 7 to 10 the runtimes of CG with BaB-SDE using the generic total maximal gain and columns 9 to 11 the solution times of CG with BaB-SDE using the application-specific pruning scheme. The arithmetic mean, the median and the geometric mean of the computation times are presented for all three methods and for each instance class. Column generation

| Solution time comparison | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | | Conventional CG | | | CG with BaB-SDE\|gtmg | | | CG with BaB-SDE\|tmg | | |
| Class | No. | arithm | median | geom | arithm | median | geom | arithm | median | geom |
| small | 313 | 1.14 | 1.06 | 1.03 | 0.28 | 0.23 | 0.26 | 0.25 | 0.19 | 0.22 |
| medium | 166 | 12.98 | 3.69 | 5.58 | 10.44 | 1.82 | 3.01 | 4.80 | 1.40 | 1.96 |
| large | 220 | 50.82 | 55.12 | 47.79 | 51.80 | 55.22 | 49.06 | 49.46 | 55.30 | 43.18 |
| $\sum$ | 699 | 19.59 | 2.47 | 5.14 | 18.91 | 0.84 | 2.40 | **16.82** | **0.81** | **1.95** |

Table 6: Comparison of solution times for conventional column generation and the variants of column generation with branch-and-bound column search.

with branch-and-bound column search using the application-specific pruning scheme performs better than the other two methods for all considered performance metrics, except for a slight disadvantage in the median of the solution times of the large instances. The column generation approach with the generic pruning scheme performs better than conventional column generation for the small- and medium-sized instances. For the large instances, the solution times for conventional column generation are somewhat shorter, but essentially the runtimes are very similar. The last line of the table shows that the column generation with BaB-SDE and the application-specific pruning has the shortest solution times w.r.t. all three average values when taking all instances together. Furthermore, BaB-SDE clearly outperforms conventional column generation and also has noticeably better results than the branch-and-bound column search with generic pruning on the instances that reach the time limit (cf. Table 5), which makes it the preferable approach in the online scheduling application at hand.

### 5.4.2 Comparison of Column Generation with Branch-and-Bound Column Search and Solving a MIP Models

In the previous section, we saw that using branch-and-bound column search has significant advantages when solving instances of the (PMSP) compared to conventional column generation. Next, we compare the integer solution obtained with our restricted master heuristic (cf. Algorithm 3), which uses branch-and-bound column search to solve the pricing problem, with the solution computed by a MIP solver. To do this, we modelled the three lexicographic levels as integrated MIPs and had them solved sequentially by an MIP solver. We will call the objective latter integer solution the *IM solution*.

In Table 7, we compare the objective function values of the two considered solution methods in all three objective functions. After the information on the instance classes in the first two columns, columns 3 to 5 list the arithmetic mean of the absolute difference (IM solution − IP solution) between the two objective values w.r.t. to the instances of the corresponding class. Columns 6 to 8 then show the average of the relative difference again calculated with the help of a primal lower bound as in Table 5.

| Objective value comparison for integrated model and CG via BaB-SDE | | | | | | | |
|---|---|---|---|---|---|---|---|
| Instance | | Absolute difference | | | Relative difference | | |
| Class | No. | Object. 1 | Object. 2 | Object. 3 | Object. 1 | Object. 2 | Object. 3 |
| small | 313 | 0.00 | 0.00 | 0.00 | 0.00% | 0.00% | 0.00% |
| medium | 166 | 0.19 | 0.10 | 0.01 | 0.02% | 0.01% | 0.00% |
| large | 220 | 208.45 | 205.35 | 0.05 | 5.16% | 10.73% | 6.06% |
| $\sum$ | 699 | 65.65 | 64.60 | 0.02 | 1.63% | 3.14% | 1.81% |

Table 7: Comparison of objective function values between the integrated model solved by an MIP solver and column generation with BaB-SDE.

For the small instances, both solution approaches again found equally good solutions. Since the time limit was not reached for any instance in this class, the integer solutions calculated by the MIP solver are provably optimal. This implies that column generation with branch-and-bound column search has indeed also found integer optimal solutions for all instances of this class. Some of the instances in class medium could neither be solved to optimality by the integrated model nor by column generation with BaB-SDE. Further, the two methods achieved similarly good results w.r.t. to solution quality here. However, for the large instances, the column generation with BaB-SDE achieved notably better results than the MIP solver for all three objective functions, which makes it the better solution method overall in the considered real-world setting of a solution time limit of 55 seconds.

### 5.4.3 Comparison of Column Generation with Branch-and-Bound Column Search and the Heuristic Algorithm of the Industry Partner

Before our collaboration, our industrial partner OrgaCard used to employ its own heuristic solution algorithm to optimize the intra-hospital transport planning problem arising at its customer hospitals. We finally compare our branch-and-bound column search approach with the solution (*OC solution*) obtained by OrgaCard's heuristic (*OC heuristic*) in Table 8. For the instances described in the first two columns, columns 3 to 5 show the arithmetic mean of the absolute differences (OC solution − IP solution) of all three objective functions, and the last three columns again contain the arithmetic mean of the corresponding relative differences. It becomes apparent that BaB-SDE allows for considerable improvements over the OC heuristic in all three instance classes with regard to the two higher prioritized

| Objective value comparison for OC heuristic and CG via BaB-SDE | | | | | | | |
|---|---|---|---|---|---|---|---|
| Instance | | Absolute difference | | | Relative difference | | |
| Class | No. | Object. 1 | Object. 2 | Object. 3 | Object. 1 | Object. 2 | Object. 3 |
| small | 284 | 24.52 | 48.77 | -0.14 | 8.10% | 13.60% | -15.66% |
| medium | 145 | 81.05 | 163.21 | -0.36 | 21.36% | 16.40% | -27.13% |
| large | 216 | 962.76 | 830.06 | 0.00 | 13.47% | 22.44% | -2.20% |
| $\sum$ | 645 | 354.84 | 339.01 | -0.14 | 12.90% | 13.03% | -13.68% |

Table 8: Comparison of the objective function values between solutions determined by the OC heuristic and column generation with BaB-SDE.

lexicographic levels (see Object. 1 and Object. 2). At the same time, concerning the third optimization goal column generation with BaB-SDE only has slightly worse results than the OC heuristic. However, this is due to the lexicographic optimization approach, which strictly prioritizes the first two optimization goals as the weighting parameters $\alpha_1$ and $\alpha_2$ were set to 1, cf. constraints (20) and (21). If required, a more balanced distribution of improvements between the three optimization goals can be achieved via another configuration of these parameters.

In summary, the column generation algorithm with branch-and-bound column search clearly achieves better results for this application than the heuristic used by our industrial partner so far. This means that branch-and-bound column search not only performs better than other generic optimization algorithms, but that it also outperforms a heuristic designed specifically for this application on real-world instances of the problem. This has led our industrial partner OrgaCard to switch to column generation with branch-and-bound column search as the preferred solution algorithm in the transport planning software system they sell to their customer hospitals.

## 6 Conclusions

In this work, we have generalized and formalized an enumerative approach by Krumke et al. (2002) for solving the pricing subproblem in column generation to what we call branch-and-bound column search. This method explores the search space by building enumeration trees containing columns that could potentially lead to a better solution within the process and thus are candidates to enter the constraint matrix of the master problem. We have presented a variant of this method Especially, we showed that it correctly solves the subproblem in a column generation approach. Furthermore, we derived an integration of the branch-and-bound column search into a column generation framework using dynamic pricing control (cf. Krumke et al. (2002)) in order to dynamically adapt the search effort of the procedure to the stability of the dual variables. Thus, in the final phase of the process, when the dual information is most reliable, the entire enumeration tree is traversed to ensure that LP-optimal solutions for the master problem are found.

The major advantage of branch-and-bound column search compared to conventional pricing approaches is that an adaptively chosen number of new columns with negative reduced costs is generated and added to the master problem, not only one. This can reduce the number of alternations between the master and the subproblem significantly. Further, the larger number of feasible columns in the integer master problem makes it much more likely to find optimal integer solutions, which renders the implementation of a dedicated branch-and-price framework superfluous in many cases. We have also developed two methods for computing exact bounds to prune the arising enumeration trees. One pruning method is completely generic and does not need to consider the explicit application at all. The other bound includes information about the application under consideration, but is applicable to many use cases due to its general calculation approach. Both pruning methodes yield considerable reductions in solution time without sacrificing the LP-optimality of the overall procedure. Moreover, we combined this column generation approach with lexicographic optimization in such a way that the pricing problem is solved by our branch-and-bound column search to re-use the columns found on one lexicographic level in the other levels as well, which yields a major speed-up for this kind of multi-criteria optimization problems.

In our computational results, we found that branch-and-bound column search not only performs significantly better than conventional column generation for both pruning schemes in the two considered applications, but also achieves better results in a real-time optimization setting than either a standard MIP solver on an integrated MIP formulation or an application-specific heuristic by our industrial partner. Overall, this shows that our approach is very well suited for use in practical optimization tasks.

## References

Agarwal, Y., Mathur, K., and Salkin, H. M. (1989). A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Bard, J. F. and Rojanasoonthon, S. (2006). A branch-and-price algorithm for parallel machine scheduling with time windows and job priorities. *Naval Research Logistics (NRL)*, 53(1):24–44.

Barnhart, C., Hane, C. A., and Vance, P. H. (1996). Integer multicommodity flow problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 58–71. Springer.

Borndörfer, R., Grötschel, M., and Pfetsch, M. E. (2007). A column-generation approach to line planning in public transport. *Transportation Science*, 41(1):123–132.

Byong-Hun, A. and Jae-Ho, H. (1990). Single facility multi-class job scheduling. *Computers & Operations Research*, 17(3):265–272.

Bärmann, A., Martin, A., Müller, A., and Weninger, D. (2022). A column generation approach for the lexicographic optimization of intra-hospital transports.

Capone, A., Carello, G., Filippini, I., Gualandi, S., and Malucelli, F. (2010). Solving a resource allocation problem in wireless mesh networks: A comparison between a CP-based and a classical column generation. *Networks: An International Journal*, 55(3):221–233.

Cattrysse, D. G., Salomon, M., and Van Wassenhove, L. N. (1994). A set partitioning heuristic for the generalized assignment problem. *European Journal of Operational Research*, 72(1):167–174.

Ceselli, A. and Righini, G. (2006). A branch-and-price algorithm for the multilevel generalized assignment problem. *Operations research*, 54(6):1172–1184.

Ceselli, A., Righini, G., and Salani, M. (2009). A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, 43(1):56–69.

Chen, S. and Shen, Y. (2013). An improved column generation algorithm for crew scheduling problems. *Journal of Information and Computational Science*, 10(1):175–183.

Chen, Z.-L. and Powell, W. B. (1999). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1):78–94.

Chvátal, V. (1983). *Linear Programming*. W. H. Freeman and Company.

Cintra, G. F., Miyazawa, F. K., Wakabayashi, Y., and Xavier, E. C. (2008). Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191(1):61–85.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410.

De Carvalho, J. V. (1998). Exact solution of cutting stock problems using column generation and branch-and-bound. *International Transactions in Operational Research*, 5(1):35–44.

De Carvalho, J. V. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659.

Dell'Amico, M., Maffioli, F., and Värbrand, P. (1995). On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354.

Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13.

Ehrgott, M. (2005). *Multicriteria Optimization*, volume 491. Springer Science & Business Media.

Engineer, F. G., Nemhauser, G. L., and Savelsbergh, M. W. (2011). Dynamic programming-based column generation on time-expanded networks: Application to the dial-a-flight problem. *INFORMS Journal on Computing*, 23(1):105–119.

Fahle, T., Junker, U., Karisch, S. E., Kohl, N., Sellmann, M., and Vaaben, B. (2002). Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81.

Friese, P. and Rambau, J. (2006). Online-optimization of multi-elevator transport systems with reoptimization algorithms based on set-partitioning models. *Discrete Applied Mathematics*, 154(13):1908–1931.

Garraffa, M., Salassa, F., Vancroonenburg, W., Vanden Berghe, G., and Wauters, T. (2016). The one-dimensional cutting stock problem with sequence-dependent cut losses. *International Transactions in Operational Research*, 23(1-2):5–24.

Gendron, B., Lebbah, H., and Pesant, G. (2005). Improving the cooperation between the master problem and the subproblem in constraint programming based column generation. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 217–227. Springer.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Grötschel, M., Borndörfer, R., and Löbel, A. (2003). Duty scheduling in public transit. In *Mathematics—Key Technology for the Future*, pages 653–674. Springer.

Gualandi, S. and Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100.

Gurobi Optimization, Inc. (2021). Gurobi optimizer reference manual. `https://www.gurobi.com/`.

Hiller, B., Klug, T., and Tuchscherer, A. (2014). An exact reoptimization algorithm for the scheduling of elevator groups. *Flexible Services and Manufacturing Journal*, 26(4):585–608.

Holmberg, K. and Yuan, D. (2003). A multicommodity network-flow problem with side constraints on paths solved by column generation. *INFORMS Journal on Computing*, 15(1):42–57.

Hooker, J. N. and van Hoeve, W.-J. (2018). Constraint programming and operations research. *Constraints*, 23(2):172–195.

Isermann, H. (1982). Linear lexicographic optimization. *OR Spektrum*, 4:223–228.

Joncour, C., Michel, S., Sadykov, R., Sverdlov, D., and Vanderbeck, F. (2010). Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702.

Junker, U., Karisch, S. E., Kohl, N., Vaaben, B., Fahle, T., and Sellmann, M. (1999). A framework for constraint programming based column generation. In *International Conference on Principles and Practice of Constraint Programming*, pages 261–274. Springer.

Krumke, S. O., Rambau, J., and Torres, L. M. (2002). Real-time dispatching of guided and unguided automobile service units with soft time windows. In *European Symposium on Algorithms*, pages 637–648. Springer.

Leus, R. and Kowalczyk, D. (2016). Improving column generation methods or sheduling problems using zdd and stabilization. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 99–103. IEEE.

Lewis, R., Song, X., Dowsland, K., and Thompson, J. (2011). An investigation into two bin packing problems with ordering and orientation implications. *European Journal of Operational Research*, 213(1):52–65.

Lopes, M. J. P. and de Carvalho, J. V. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European journal of operational research*, 176(3):1508–1527.

Lübbecke, M. E. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53(6):1007–1023.

Mehrotra, A. and Trick, M. A. (1996). A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354.

Morrison, D. R., Sewell, E. C., and Jacobson, S. H. (2016). Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams. *INFORMS Journal on Computing*, 28(1):67–82.

Park, J. S., Lim, B. H., and Lee, Y. (1998). A lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem. *Management Science*, 44(12-part-2):S271–S282.

Psaraftis, H. N. (1980). A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, 28(6):1347–1359.

Regionales Rechenzentrum Erlangen (2022). Woodcrest Cluster. `https://hpc.fau.de/systems-services/systems-documentation-instructions/clusters/woody-cluster/`.

Rentmeesters, M. J., Tsai, W. K., and Lin, K.-J. (1996). A theory of lexicographic multi-criteria optimization. In *Proceedings of ICECCS'96: 2nd IEEE International Conference on Engineering of Complex Computer Systems (held jointly with 6th CSESAW and 4th IEEE RTAW)*, pages 76–79. IEEE.

Sabuncuoglu, I. and Bayiz, M. (1999). Job shop scheduling with beam search. *European Journal of Operational Research*, 118(2):390–412.

Sadykov, R. and Vanderbeck, F. (2013). Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255.

Savelsbergh, M. (1997). A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841.

Schewe, L., Schmidt, M., and Weninger, D. (2020). A decomposition heuristic for mixed-integer supply chain problems. *Operations Research Letters*, 48(3):225–232.

Tomlin, J. (1966). Minimum-cost multicommodity network flows. *Operations Research*, 14(1):45–51.

Van den Akker, J. M., Hoogeveen, J. A., and van de Velde, S. L. (1999). Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872.

Van Huigenbosch, P., van de Klundert, J., and Wormer, L. (2011). ANWB automates and improves service personnel dispatching. *Interfaces*, 41(2):123–134.

Vanderbeck, F. (1999). Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594.

Webster, S. and Baker, K. R. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, 43(4):692–703.

Westphal, S. and Krumke, S. O. (2008). Pruning in column generation for service vehicle dispatching. *Annals of Operations Research*, 159(1):355–371.

Yunes, T., Moura, A., and de Souza, C. (1999). Exact solutions for real world crew scheduling problems. In *INFORMS Fall 1999 Meeting*.

Yunes, T. H., Moura, A. V., and De Souza, C. C. (2005). Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, 39(2):273–288.

Zykina, A. V. (2004). A lexicographic optimization algorithm. *Automation and Remote Control*, 65(3):363–368.

## Conflict of interest

In addition to his affiliation with the Friedrich-Alexander-Universität Erlangen-Nürnberg, the author Alexander Müller is also an employee of the industry partner OrgaCard Siemantel & Alt GmbH. The authors have no relevant financial or non-financial interests to disclose.

# A Appendix

In the appendix, we give some of the details omitted in the main part of the article. This entails a formulation of a generic subproblem as a MIP, the branch-and-bound column search in the sequence-independent case, indicative further use cases for branch-and-bound column search as well as some of the MIP formulations and bounding routines we used for the computational experiments in Section 5.

## A.1 Classical Solution of the Subproblem as an MIP

In Section 3, we looked at the branch-and-bound column search as a generic solution method to the pricing problem for master problems with the generalized assignment structure. A frequently chosen generic solution method for column generation is to formulate the pricing problem as an MIP. A generic mixed-integer programming model for the pricing subproblem corresponding to (RMP) can be formulated as follows:

$$\begin{aligned}
\text{(SP)} \qquad & \min_y \quad f(y_D, y_I, y_S) - \sum_{d\in D} y_d \bar{\pi}_d - \sum_{i\in I} y_i \bar{\pi}_i \\
& \text{s.t.} \quad B \begin{pmatrix} y_D \\ y_I \\ y_S \end{pmatrix} \geqslant \tilde{b} && (23) \\
& \qquad\quad y_D \in \mathbb{Z}^{|D|},\, y_I \in \{0,1\}^{|I|},\, y_S \in \mathbb{R}^{|S|},
\end{aligned}$$

where $B \in \mathbb{R}^{\tilde{m}\times(|D|+|I|+|S|)}$, with $\tilde{m} < \infty$, and index sets $D$, $I$ and $S$ as above. The Constraints (23) given by matrix $B$ and right-hand side $\tilde{b}$ are modelled in such a way that (SP) computes only feasible plans $p \in P$ with respect to the application of interest. It must therefore be possible to deduce from variables $y \in \mathbb{Z}^{|D|} \times \{0,1\}^{|I|} \times \mathbb{R}^{|S|}$ the corresponding column $A_{\cdot,p}$ of the constraint matrix of master problem (MP). For this transformation to work, $y$ is subdivided into variables $y_D$, $y_I$, and $y_S$ as follows. We have *task assignment variables* $y_D \in \mathbb{Z}^{|D|}$ that define how often each task $d \in D$ is executed in the solution of (SP). These variables $y_D$ correspond to the coefficients $A_{D,p} \in \mathbb{Z}^{|D|}$ of the constraint matrix $A$ in (MP) for the newly generated plan $p \in P\backslash\tilde{P}$. Further, the *agent allocation variables* $y_I \in \{0,1\}^{|I|}$ specify the agent to whom the plan constructed in (SP) is assigned to. They correspond to the coefficients $A_{I,p} \in \{0,1\}^{|I|}$ of $A$. Finally, the *structure variables* $y_S \in \mathbb{R}^{|S|}$ describe a particular structure (dependent on the considered application) of the plan constructed in (SP), such as a sequence, an execution time, or other structural properties of a plan. The variables $y_S$ correspond to the structural arguments $w_{S,p}$ used in cost function $f$ for the columns of (MP).

The values of $y_D$, $y_I$ and $y_S$ in a solution to (SP) allows to calculate the primal cost $c_p$ for the newly generated plan $p \in P\backslash\tilde{P}$ using an application specific cost function $f$. Altogether, this all required information to create a new column for the restricted master problem. Moreover, the objective function of (SP) guarantees that an optimal plan $p \in P$ has minimum reduced cost $\bar{c}_p$ with respect to a given optimal dual solution $(\bar{\pi}_d, \bar{\pi}_i)$ with $d \in D$ and $i \in I$.

**Example A.1** (MIP Subproblem)**.** *We continue Example 2.1 in order to show how the subproblem of a vehicle routing problem can be modelled. Mathematically, the pricing problem takes the form of a* prize-collecting travelling salesman problem (PCTSP)*, i.e. we need to find a tour for one of the agents ("salesmen") $i \in I$ that visits a chosen subset of customers $D^* \subseteq D$ of minimal cost (travel costs minus gain per visited customer). A possible formulation as an MIP is as follows:*

$$\begin{aligned}
(PCTSP) \quad & \min_y \sum_{d\in D} c_d y_d + \sum_{i\in I} c_i y_i + \sum_{s\in S} c_s y_s - \sum_{d\in D} y_d \pi_d - \sum_{i\in I} y_i \pi_i \\
\text{s.t.} \quad & y_d = \sum_{s\in\delta^-(\{d\})} y_s && d \in D && (24) \\
& y_d = \sum_{s\in\delta^+(\{d\})} y_s && d \in D && (25) \\
& \sum_{d\in D} y_{(t,d)} = \sum_{d\in D} y_{(d,t)} = 1 && && (26) \\
& \sum_{s\in\delta(V)} y_s \leqslant |V| - 1 && \varnothing \neq V \subseteq D && (27)
\end{aligned}$$

$$\sum_{i \in I} y_i = 1 \tag{28}$$

$$y_d, y_i, y_s \in \{0,1\} \qquad d \in D. \tag{29}$$

*The structure of a column can be modelled via the index set $S := D \cup \{t\} \times D \cup \{t\}$, where $t$ is the depot. In this particular problem, the structure is given by the order of the customers visited in the corresponding tour. The variables $y_s = y_{(d_1,d_2)}$ for $(d_1, d_2) \in S$ take a value of 1 if customer $d_1$ is followed by customer $d_2 \in D$ on the chosen tour, and 0 otherwise. The task assignment variables $y_d$ with $d \in D$ have value $y_d = 1$ if the customer $d$ is visited in the constructed tour, and $y_d = 0$ otherwise. Finally, the agent assignment variables indicate which salesman performs the constructed tour as they take the value $y_i = 1$ if salesman $i \in I$ executes the tour, and $y_i = 0$ if not. Furthermore, the following subsets of $S$ are used in (PCTSP): $\delta^+(X) := \{(d_i, d_j) \in S \mid d_i \in X\}$, $\delta^-(X) := \{(d_j, d_i) \in S \mid d_i \in X\}$ and $\delta(X) := \delta^+(X) \cap \delta^-(X)$ for any $X \subseteq D \cup \{t\}$.*

*The objective function minimizes the reduced cost of the columns to be generated, which are tours in this application. The assignment constraints (24)–(25), formulated above as in Dell'Amico et al. (1995), require that each visited customer have exactly one incoming edge and one outgoing edge. Constraint (26) ensures that the depot is left and entered exactly once. Inequality (27) is a modification of the well-known subtour elimination constraint (cf. e.g. Dantzig et al. (1954)) that in our case prevents unwanted subcycles within a tour of the salesman, i.e. the walk over the chosen customers excluding the depot is always a path. Finally, Constraint (28) requires that the tour is assigned to exactly one salesman $i \in I$.*

Obviously, there are only $|I|$ different possibilities which values variables $y_I \in \{0,1\}^{|I|}$, or finally the partial column $A_{I,p} \in \{0,1\}^{|I|}$, can take for $p \in P$. This is because they are all unit vectors in $\mathbb{R}^{|I|}$ due to the assumption that $\sum_{i \in I} a_{i,p} = 1$ for all $p \in P$. In summary, a plan can only be given to exactly one agent. Therefore, it is possible to formulate separate subproblems (SP$_i$), that only generate new columns for the respective agent $i \in I$.

## A.2 Sequence-Independent Enumeration Tree (SIDE)

Section 3.3 describes the enumeration procedure if the sequence of tasks within a plan has a crucial impact on the costs of the plan. This is certainly not the case for all applications. The presented algorithm in this section supplements Section 3.3 with the sequence-independent variant of the branch-and-bound column search.

The starting point of Algorithm 4 is the root node $v_r = (A_{D,r}, e_i, w_{S,r})$, where all coefficients are set to $a_{d,r} = a_d^{\min}$ and the label to $\text{LS}_r = 1$. The set $Q$, the so called queue, is needed to keep track of child nodes that have been enumerated and that may have descendants with negative reduced costs. At the beginning, $Q$ contains only the root node, which has depth $\phi(r) = 0$. The initially empty set $\mathcal{C}$ is filled over the course of the procedure with all generated columns $A_{\cdot,p}$ that have reduced costs lower than the acceptance threshold (i.e. $\bar{c}_p < \theta$) and that are feasible plans for the application (i.e. $A_{\cdot,p} \in \mathcal{A}$). As long as the set $Q$ is not empty, starting from a current node $v_p$, child nodes $v_{p_{\text{new}}}$ are created by increasing the coefficients $a_{d,p}$ by 1 according to the binary operation $\oplus_i$ for each task $d \in \tilde{D}$ whose position in tuple $\tilde{\mathcal{D}}$ is greater or equal to $\text{LS}_p$. This is done for all tasks $d \in \tilde{D}$ for which $a_{d,p} < a_d^{\max}$ holds (cf. Line 11). Then the structure $w_{S,p_{\text{new}}}$ of the new node $v_{p_{\text{new}}}$ for $p_{\text{new}} \in \bar{P}$ must be determined (cf. Line 15). The structure of a node and how it is determined strongly depends on the concrete application problem to be solved, for which we will give examples later. Nevertheless, the structure of the new column $p_{\text{new}} \in \bar{P}$ is determined as the cost-minimizing structure $w_{S,p_{\text{new}}}$ which incorporates the tasks assigned in the partial column $A_{D,p_{\text{new}}}$. Our experience has shown that the minimum cost structure in the application context is usually relatively easy or fast to determine and therefore this is not a crucial criterion for the performance of the method.

The label $\text{LS}_p$ for $p \in \bar{P}$ specifies the index in $\tilde{\mathcal{D}}$ of that task $\tilde{d} \in \tilde{D}$ whose coefficient $a_{\tilde{d},p}$ was last increased for node $v_p$ in the course of the algorithm. Therefore, the label of column $p_{\text{new}}$ is now set to the value $\text{LS}_{p_{\text{new}}} = j$ (cf. Line 17), which corresponds to the index of $d$ in the tuple $\tilde{\mathcal{D}}$. Based on node $v_{p_{\text{new}}} = (A_{D,p_{\text{new}}}, e_i, w_{S,p_{\text{new}}})$ and the costs $c_{p_{\text{new}}}$, it is then possible to calculate the reduced costs $\bar{c}_{p_{\text{new}}}$ of the new column $p_{\text{new}} \in \bar{P}$. If node $v_{p_{\text{new}}}$ contains a feasible column for the master problem with sufficiently small reduced costs, i.e. $A_{\cdot,p_{\text{new}}} \in \mathcal{A}$ and $\bar{c}_{p_{\text{new}}} < \theta$, then the column might be part of an optimal solution of the LP-relaxed master problem and is therefore included in $\mathcal{C}$.

If the lower bound fulfils $\bar{c}_{p_{\text{new}}} - \text{tmg}(p_{\text{new}}, \ell, \tilde{D}, \bar{\pi}; F, C) < \theta$, then it is possible that descendants of node $v_{p_{\text{new}}}$ have reduced costs smaller than the acceptance threshold $\theta$. Thus, we cannot prune the enumeration tree at that node, which is why we add the index of node $v_{p_{\text{new}}}$ is to $Q$. Otherwise, we know from Theorem 3.8 that the reduced costs of the descendants of $v_{p_{\text{new}}}$ are greater than or equal to

$\bar{c}_{p_{\text{new}}} - \text{tmg}(p_{\text{new}}, \ell, \tilde{D}, \bar{\pi}; F, C)$ and thus also greater than or equal to the current acceptance threshold $\theta$, such that we can prune the tree at this point. That is, $p_{\text{new}}$ is not added to $Q$ (cf. Lines 25 – 28).

**Algorithm 4:** BaB – SIDE

```
Input : A $|\tilde{D}|$-Tuple $\tilde{\mathcal{D}}$ of tasks $d \in \tilde{D} \subseteq D$, bounds for column entries $(a_d^{\min}, a_d^{\max})$ for all $d \in \tilde{D}$,
        agent $i \in I$, search depth $\ell$, optimal solution $\bar{\pi} \in \mathbb{R}^{|D|+|I|}$ of (RDP), acceptance
        threshold $\theta$.
Output: An enumeration tree $T_i$ and a set of all columns $\mathcal{C}$ with reduced costs smaller than $\theta$
        and a node depth of at most $\ell$
1  Initialize:
2      $r \leftarrow 0$, $a_{d,r} \leftarrow a_d^{\min}\ \forall d \in D$, $\text{LS}_r = 1$
3      $w_{S,r} \leftarrow$ Determine by application context.
4      $v_r \leftarrow (A_{D,r}, e_i, w_{S,r})$
5      $Q \leftarrow \{r\}$, $\mathcal{C} \leftarrow \emptyset$, $V_i \leftarrow \{v_r\}$, $E_i \leftarrow \emptyset$
6  while $Q \neq \emptyset$ do
7      Choose $p \in Q$
8      $Q \leftarrow Q \backslash \{p\}$
9      for $j = \text{LS}_p : |\tilde{D}|$ do
10         $d \leftarrow \tilde{\mathcal{D}}_j$
11         if $a_{d,p} < a_d^{\max}$ then
12             $p_{\text{new}} \leftarrow p + 1$
               // Apply binary operation $v_{p_{\text{new}}} = v_p \oplus_i d$
13             $A_{D,p_{\text{new}}} \leftarrow A_{D,p}$
14             $a_{d,p_{\text{new}}} \leftarrow a_{d,p} + 1$
15             Choose $w_{S,p_{\text{new}}} \in \text{argmin}_{w \in \mathbb{R}^{|S|}} f(A_{D,p_{\text{new}}}, e_i, w)$
16             $v_{p_{\text{new}}} \leftarrow (A_{D,p_{\text{new}}}, e_i, w_{S,p_{\text{new}}})$
17             $\text{LS}_{p_{\text{new}}} \leftarrow j$ // Store the row index that was increased for $v_{p_{\text{new}}}$
               // Add new node $v_{p_{\text{new}}}$ to tree $T_i$
18             $V_i \leftarrow V_i \cup \{v_{p_{\text{new}}}\}$
19             $E_i \leftarrow E_i \cup \{(v_p, v_{p_{\text{new}}})\}$
20             $\bar{c}_{p_{\text{new}}} = f(v_{p_{\text{new}}}) - \sum_{d \in D} a_{d,p_{\text{new}}} \bar{\pi}_d - \bar{\pi}_i$
21             if $\bar{c}_{p_{\text{new}}} < \theta$ and $A_{\cdot,p_{\text{new}}} \in \mathcal{A}$ then // Check for reduced costs/feasibility
22                 $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_{p_{\text{new}}}\}$
23             end
24             if $\phi(p_{\text{new}}) < \ell$ then // Bounding due to reduced costs/node depth
25                 if $\bar{c}_{p_{\text{new}}} - \text{tmg}(p_{\text{new}}, \ell, \tilde{D}, \bar{\pi}; F, C) < \theta$ then
26                     $Q \leftarrow Q \cup \{p_{\text{new}}\}$
27                 end
28             end
29         end
30     end
31 end
32 return $T_i = (V_i, E_i, \oplus_i)$, $C$
```

**Remark A.2.** *Obviously, the largest reasonable value to which search depth $\ell$ can be set is $\ell_{\max} := \sum_{d \in D} a_d^{\max} - a_d^{\min}$. In this case, the algorithm also generates plans that contain the maximum number $a_d^{\max}$ for each task $d \in D$, and thus it is also ensured that all feasible columns for* (MP) *are generated. However, in most applications it is possible and desirable in terms of running time to choose the maximal search depth smaller than $\ell_{\max}$ and nevertheless maintain the exactness of the column generation method. The maximum search breadth is reached when $\tilde{D}$ is chosen to be the set of all tasks $D$.*

A question that arises now is whether any columns are enumerated unnecessarily in Algorithm 4. In other words: does the algorithm ever enumerate the same node several times? For the sequence-independent case, we say that two nodes $v_p, v_{\bar{p}}$ are *equal* if $A_{D,p} = A_{D,\bar{p}}$ and $A_{I,p} = A_{I,\bar{p}}$. Since in Algorithm 4 the structure variable $w_{S,p}$ is set to minimize the cost, for such nodes we have $(A_{D,p}, A_{I,p}) = (A_{D,\bar{p}}, A_{I,\bar{p}}) \Rightarrow c_p = c_{\bar{p}}$. If there are no two nodes generated by Algorithm 4 that are equal, i.e. no node was created twice and therefore every node has one unique path to the root node $v_r \in V_i$, then the graph $T_i = (V_i, E_i, \oplus_i)$ created by Algorithm 4 is indeed a tree.

In order to prove this, we first show the following lemma, which states that when generating the columns in Algorithm 4 by increasing the coefficients $a_{d,p}$ with $d \in D$ the order given by the tuple $\tilde{\mathcal{D}}$ is strictly followed (cf. Figure 3).

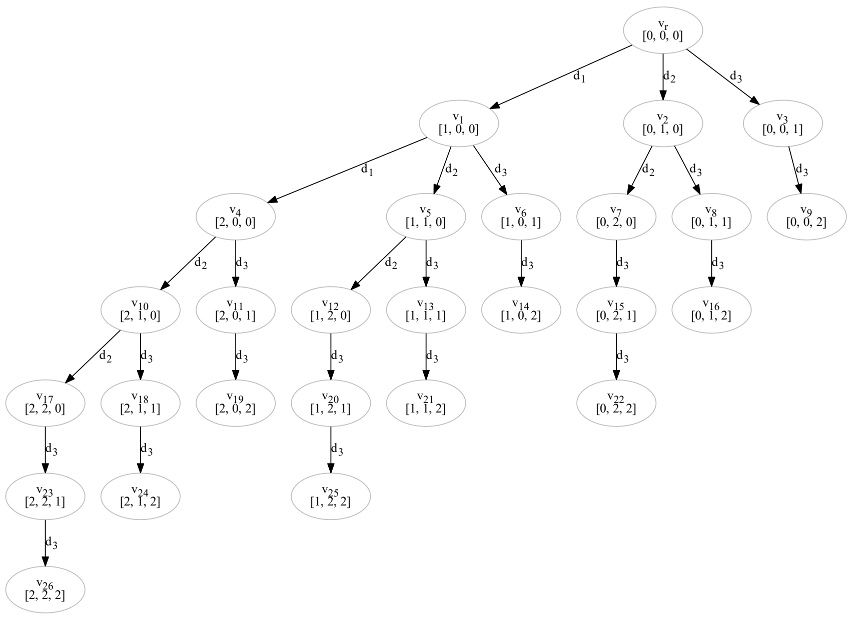

Figure 3: Illustration of an enumeration tree $T_i$ for a specific agent $i \in I$, with $\tilde{\mathcal{D}} = (d_1, d_2, d_3)$, $a_d^{\min} = 0$, $a_d^{\max} = 2$ for all $d \in \tilde{D}$, constructed by the sequence-independent enumeration algorithm Bab – SIDE.

**Lemma A.3.** *Let $T_i = (V_i, E_i, \oplus_i)$, $i \in I$, be a graph that is constructed by Algorithm 4 for a subset of tasks $\tilde{D}$ ordered in a tuple $\tilde{\mathcal{D}}$. Then for all nodes $v_p, v_{\bar{p}} \in V_i$ such that $v_p \leqslant_{T_i} v_{\bar{p}}$ and* $\mathrm{LS}_p = \bar{j}$ *with $\bar{j} \in \{1, \dots, |\tilde{D}|\}$ we have*

$$a_{d,\bar{p}} = a_{d,p} \quad \forall d = \tilde{\mathcal{D}}_j \text{ with } j < \bar{j}.$$

*Proof.* Let $v_p = (A_{D,p}, e_i, w_{S,p})$ be a node in $T_i$ with label $\mathrm{LS}_p = \bar{j}$. For all child nodes $v_{\hat{p}}$ of $v_p$, only tasks $d = \tilde{\mathcal{D}}_{j^+}$ with $j^+ \geqslant \bar{j}$ are attached due to Lines 9 and 10 in Algorithm 4, such that $a_{d,\hat{p}} = a_{d,p}$ applies for all $d = \tilde{\mathcal{D}}_{j^-}$ with $j^- < \bar{j}$.

In addition, label $\mathrm{LS}_{\hat{p}}$ is set for all child nodes $v_{\hat{p}}$ as $\mathrm{LS}_{\hat{p}} = j^+$ with $j^+ \geqslant \bar{j}$ (cf. Line 17), which is why again only tasks $d = \tilde{\mathcal{D}}_{j^+}$ with $j^+ \geqslant \bar{j}$ are attached to the children of these nodes as well. This continues for all descendants of $v_p$, which finishes the proof. □

We can now use this result to prove Lemma A.4.

**Lemma A.4.** *The graph $T_i = (V_i, E_i, \oplus_i)$ constructed by Algorithm 4 is an arborescence, i.e. for the initial node $v_r \in V_i$ in the algorithm and for any two nodes $v_p, v_{\bar{p}} \in V_i$ there are connecting paths $\mathcal{P} := (v_r, \dots, v_p)$, $\bar{\mathcal{P}} := (v_r, \dots, v_{\bar{p}})$ in $T_i$ and we have*

$$v_p = v_{\bar{p}} \Rightarrow \mathcal{P} = \bar{\mathcal{P}},$$

*and every node $v_p$ is visited exactly once via its unique path $\mathcal{P}$ in Algorithm 4.*

*Proof.* Every newly built node $v_{p_{\text{new}}}$ in Algorithm 4 is connected to one previously constructed node $v_p$. Since $v_r$ is the initial node in the algorithm, it follows inductively that every node is connected to $v_r$. So for every node $v_p$ constructed by Algorithm 4, a path from $v_r$ to $v_p$ exists. Let $v_p, v_{\bar{p}} \in V_i$ be nodes with their respective paths $\mathcal{P} = (v_r, \dots, v_p)$ and $\bar{\mathcal{P}} = (v_r, \dots, v_{\bar{p}})$ from the inital node $v_r$. We want to show that if $v_p = v_{\bar{p}}$ then $\mathcal{P} = \bar{\mathcal{P}}$:
Let $l$ be the number of nodes in $\mathcal{P}$ and $\bar{l}$ the number of nodes in $\bar{\mathcal{P}}$. Denote with $\mathcal{P}_k$ the $k$-th node in path $\mathcal{P}$ for $k \in \{1, \dots, l\}$ and with $\bar{\mathcal{P}}_k$ the $k$-th node in $\bar{\mathcal{P}}$ for $k \in \{1, \dots, \bar{l}\}$. Suppose $\mathcal{P} \neq \bar{\mathcal{P}}$, which is equivalent to $(l \neq \bar{l}) \vee (l = \bar{l} \wedge \exists k \in \{1, \dots, l\} : \mathcal{P}_k \neq \bar{\mathcal{P}}_k)$. We distinguish the following two cases:
1.) Case $l \neq \bar{l}$:

If $l \neq \bar{l}$, then the paths $\mathcal{P}$ and $\bar{\mathcal{P}}$ contain a different number of nodes. Since each successor node $\mathcal{P}_{k+1}$ originates from its predecessor $\mathcal{P}_k$ by increasing exactly one coefficient, the sum over all coefficients $\sum_{d \in \tilde{D}} a_{d,p} \neq \sum_{d \in \tilde{D}} a_{d,\bar{p}}$ for $v_p$ and $v_{\bar{p}}$ is different and thus there must also be at least one $d \in \tilde{D}$ such that $a_{d,p} \neq a_{d,\bar{p}}$, from which follows $v_p \neq v_{\bar{p}}$.

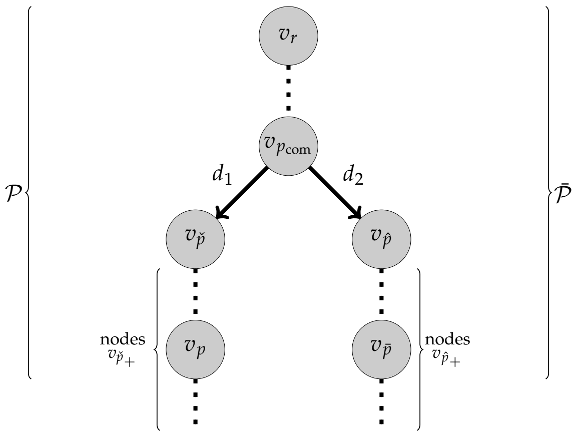

Figure 4: Illustration of the nodes in the second case of the proof of Lemma A.4.

2.) Case $l = \bar{l}$ and $\exists k \in \{1, \dots, l\} : \mathcal{P}_k \neq \bar{\mathcal{P}}_k$:
W.l.o.g., let $k$ be the first index such that $\mathcal{P}_k \neq \bar{\mathcal{P}}_k$, i.e. $\mathcal{P}_q = \bar{\mathcal{P}}_q$ for all $q < k$ with $q, k \in \{1, \dots, l\}$. The nodes $v_{\check{p}} = \mathcal{P}_k$ and $v_{\hat{p}} = \bar{\mathcal{P}}_k$ have then been generated by adding different tasks $d_1, d_2 \in D$ to the last common node $v_{p_{\text{com}}} = \mathcal{P}_{k-1} = \bar{\mathcal{P}}_{k-1}$ (cf. Figure 4). W.l.o.g., we assume $d_1 = \tilde{\mathcal{D}}_{j_1}$ and $d_2 = \tilde{\mathcal{D}}_{j_2}$ with $j_1 < j_2$. Further, let $v_{\check{p}} = v_{p_{\text{com}}} \oplus d_1$, $v_{\hat{p}} = v_{p_{\text{com}}} \oplus d_2$. This means $a_{d_1,\check{p}} > a_{d_1,\hat{p}} = a_{d_1,p_{\text{com}}}$ and $a_{d_2,\hat{p}} > a_{d_2,\check{p}} = a_{d_2,p_{\text{com}}}$. Thus, in Line 17 of Algorithm 4 the labels were set to $\text{LS}_{\check{p}} = j_1$ for node $v_{\check{p}}$ and $\text{LS}_{\hat{p}} = j_2$ for node $v_{\hat{p}}$. Together with Lemma A.3 and since $j_1 < j_2$, this leads to $a_{d_1,\hat{p}_+} = a_{d_1,\hat{p}}$ for all nodes $v_{\hat{p}_+}$ such that $v_{\hat{p}} \leqslant_{T_i} v_{\hat{p}_+}$ and $a_{d_1,\check{p}_+} \geqslant a_{d_1,\check{p}}$ for all nodes $v_{\check{p}_+}$ such that $v_{\check{p}} \leqslant_{T_i} v_{\check{p}_+}$ (cf. Figure 4). As $v_{\check{p}} \leqslant_{T_i} v_p$ and $v_{\hat{p}} \leqslant_{T_i} v_{\bar{p}}$, it follows $a_{d_1,p} \geqslant a_{d_1,\check{p}} > a_{d_1,\hat{p}} = a_{d_1,\bar{p}}$, which leads to $a_{d_1,p} > a_{d_1,\bar{p}}$. Thus, again we have $v_p \neq v_{\bar{p}}$.

Altogether, the only way a node could be visited twice would be if its path was traversed twice in Algorithm 4. But we can rule this out, since each task $d = \tilde{\mathcal{D}}_j$ with $j \in \{LS_p, \dots, |\tilde{D}|\}$ is added only once to the same node (if even possible), cf. Lines 9–30. Thus, it is clear that each node $v_p$ can only be reached via a single path and also that each column is generated exactly once in BaB – SIDE. □

At some point in the column generation procedure, it is necessary to verify that in Algorithm 4 all nodes whose columns have negative reduced costs are actually visited in order to preserve the exactness of the method. Therefore, we show now that Algorithm 4 can be used to enumerate all feasible columns $A_{\cdot,p}$ with $p \in P$ for (MP) for a specific agent $i \in I$, where $a_d^{\min} \leqslant a_{d,p} \leqslant a_d^{\max}$ for all $d \in D$ and where the reduced costs fulfil $\bar{c}_p \leqslant 0$ for a specific setting of input parameters.

**Theorem A.5.** *Consider a master problem* (MP) *for a problem class where the sequence of tasks in plan $p \in P$ has no effect on its costs $c_p$, and let $\bar{\pi} \in \mathbb{R}^{|D|+|I|}$ be an optimal solution to the restricted dual problem* (RDP)*. If the acceptance threshold is $\theta = 0$, the search depth is $\ell = \ell_{\max}$ and the tuple $\tilde{\mathcal{D}}$ contains all tasks $d \in D$, i.e. $\tilde{D} = D$, then Algorithm 4 constructs a complete enumeration tree $T_i = (V_i, E_i, \oplus_i)$ for an agent $i \in I$* (*cf. Definition 3.4*)*.*

*Proof.* We already know by Lemma A.4 that the graph constructed in Algorithm 4 is a tree. The nodes $V_i$ generated by this algorithm are obviously columns for the master problem (MP), and by the construction of the algorithm (cf. Lines 14 and 19), Condition (ii) of Definition 3.1 is also satisfied. Thus, we know that this tree is an enumeration tree. It remains to show that this enumeration tree is also a complete enumeration tree.

Let $(A_{D,\bar{p}}, e_i)$ be an arbitrary column with $a_d^{\min} \leqslant a_{d,\bar{p}} \leqslant a_d^{\max}$ for $d \in D$ that has negative reduced costs $c_{\bar{p}} < 0$ and that fulfils $\sum_{d \in D} a_{d,\bar{p}} - a_d^{\min} \leqslant \ell_{\max}$. We will now show that this arbitrary column corresponds to a node $v_{\bar{p}}$ in a tree $T_i$ by tracing how the path from the initial node $v_r$ to this node $v_{\bar{p}}$ is built by BaB – SIDE. Starting from the initial node $v_r$, the algorithm iterates over all $d \in \tilde{D}$ (w.r.t. the order defined in tuple $\tilde{\mathcal{D}}$, cf. Lines 9 and 10). Since $\text{LS}_r = 1$ and $\tilde{D} = D$, we know that it iterates over all $d \in D$. Therefore, at some point, a child node $v_{p_1}$ will be constructed where the task $d_1 = \tilde{\mathcal{D}}_{j_1}$ with index $j_1 = \min\{j \in \{1, \dots, |D|\} \mid a_{d,\bar{p}} > a_d^{\min}, d = \tilde{\mathcal{D}}_j\}$, has been increased by one (i.e. $a_{d_1,p_1} = a_d^{\min} + 1$). Since the label is still $\text{LS}_1 = j_1$ for node $v_{p_1}$, Algorithm 4 later visits a child node $v_{p_2}$ of $v_{p_1}$ where the coefficient with index $d_1$ again was increased by one, such that $a_{d_1,p_2} = a_{d_1}^{\min} + 2$. This is then repeated until node $v_{p_{k_1}}$ is visited, where $a_{d_1,p_{k_1}} = a_{d_1,\bar{p}}$ and $a_{d,p_{k_1}} = a_d^{\min}$ for all $d \in D \backslash \{d_1\}$.

Up to now, the algorithm has only built new nodes by increasing coefficients corresponding to tasks $d_1$, such that the label is still $\text{LS}_{k_1} = j_1$. Thus, there is also a child of node $v_{p_{k_1}}$ constructed by increasing the coefficient corresponding to the task $d_2 = \tilde{\mathcal{D}}_{j_2}$ by one, where $j_2 > j_1 = \mathrm{LS}_{k_1}$ is the second-smallest index such that $a_{d_2,\bar{p}} > a_d^{\min}$ with $d_2 = \mathcal{D}_{j_2}$. Algorithm 4 follows this procedure for task $d_2$ a number of $a_{d_2,\bar{p}}$ times and arrives at $v_{p_{k_2}}$, where $a_{d_1,p_{k_2}} = a_{d_1,\bar{p}}$, $a_{d_2,p_{k_2}} = a_{d_2,\bar{p}}$ and $a_{d,p_{k_2}} = a_d^{\min}$ for all $d \in D \backslash \{d_1, d_2\}$. Algorithm 4 repeats this process for the remaining indices $d \in D \backslash \{d_1, d_2\}$, where $a_{d,\bar{p}} > a_d^{\min}$ such that it constructs our arbitrarily chosen column $(A_{D,\bar{p}}, e_i)$ in the end.

Finally, we need to make sure that the column $(A_{D,\bar{p}}, e_i)$ has not been pruned in Lines 25 – 28 of Algorithm 4. However, since column $(A_{D,\bar{p}}, e_i)$ has negative reduced cost, we have by Theorem 3.8 that $\bar{c}_{\hat{p}} - \mathrm{tmg}(p, \ell, \tilde{D}, \bar{\pi}; F, C) \leqslant \bar{c}_{\bar{p}} < 0$ for all nodes $v_{\hat{p}}$ such that $v_{\hat{p}} \leqslant_{T_i} v_{\bar{p}}$. Furthermore, we chose $(A_{D,\bar{p}}, e_i)$ such that $\sum_{d \in D} a_{d,\bar{p}} - a_d^{\min} \leqslant \ell_{\max}$. Therefore, we can rule out that the corresponding node $v_{\bar{p}}$ was pruned in Lines 25 – 28, cf. Equation (8).

Having arbitrarily chosen the column $A_{\cdot,\bar{p}} \in \mathcal{A}$, it has been shown that Algorithm 4 with the above defined input parameter setting generates any node $v_p$ with negative reduced costs for which $a_{d,p} \leqslant a_d^{\max}$ holds for all $d \in D$. □

## A.3 Further Applications for Branch-and-Bound Column Search

In Section 3, we presented two variants of our branch-and-bound column search, namely BaB – SIDE and BaB – SDE for the sequence independent and the sequence-dependent case, respectively. Here we discuss a selection of problem classes which can be solved by column generation and where branch-and-bound column search can thus be used for the solution of the pricing problem. In Table 9, several problems are listed where the order of the tasks within a plan has no influence on the costs of the corresponding column, such that we can use BaB – SIDE for the solution of the pricing problem. Table 10 then lists exemplary sequence-dependent problem classes for which BaB – SDE can be applied. Each of the two tables is divided into two sections. The upper section contains the problem classes in which each task can only be added once to a plan ($a_d^{\min} = 0 \wedge a_d^{\max} = 1 \, \forall d \in D$); in the lower section, tasks can also be added several times to a plan ($a_d^{\min} = 0 \wedge a_d^{\max} > 1 \, \forall d \in D$). Moreover, in the second column of the tables, common applications for each case are listed, supplemented by indicative literature references from where the problem originates. The last column provides the problem class of the pricing problem in a column generation approach.

| Case | Applications | Subproblem |
|---|---|---|
| | Sequence Independent Enumeration | |
| $a_d^{\min} = 0 \wedge a_d^{\max} = 1$ $\forall d \in D$ | **Bin packing problem**<br>De Carvalho (1999)<br>Vanderbeck (1999) | Knapsack problem |
| | **Generalized assignment problem**<br>Cattrysse et al. (1994)<br>Savelsbergh (1997)<br>Ceselli and Righini (2006) | Knapsack problem |
| | **Graph colouring problem**<br>Mehrotra and Trick (1996)<br>Gualandi and Malucelli (2012)<br>Morrison et al. (2016) | Maximum weight independent set problem |
| $a_d^{\min} = 0 \wedge a_d^{\max} > 1$ $\forall d \in D$ | **Cutting stock**<br>Gilmore and Gomory (1961)<br>De Carvalho (1998)<br>Vanderbeck (1999) | Knapsack problem |
| | **Generalized multi-assignment problem**<br>Park et al. (1998) (not solved with CG) | (Knapsack problem) |

Table 9: Applications where the branch-and-bound column search algorithm BaB – SIDE can be used as a solution algorithm for the pricing problem.

**Remark A.6.** *Note that* bin packing *and the* cutting stock problem *are very similar. In the literature, there is no generally valid distinction between these two problem classes. Often, the cutting stock problem is seen as a generalization of the bin packing problem where each item must be covered $b_d \in \mathbb{N}$ times for $d \in D$ and a bin may contain multiple copies of the same item (cf. Vanderbeck (1999)). Thus, we adopted this notion in the cutting stock application presented in Section 5.1.*

| Sequence-Dependent Enumeration | | |
|---|---|---|
| $a_d^{\min} = 0 \wedge a_d^{\max} = 1$ | **Vehicle routing problem** | Travelling salesman problem |
| $\forall d \in D$ | Agarwal et al. (1989) | (generic case: knapsack problem) |
| | Desrochers et al. (1992) | |
| | Krumke et al. (2002) | |
| | **Crew scheduling** | Constrained shortest path problem |
| | Desrochers and Soumis (1989) | |
| | Grötschel et al. (2003) | |
| | Chen and Shen (2013) | |
| | **Parallel machine scheduling** | Single machine scheduling problem |
| | Chen and Powell (1999) (not solved with CG) | |
| | Van den Akker et al. (1999), | |
| | Lopes and de Carvalho (2007). | |
| | **Multi-commodity flow problems** | Shortest path problem. |
| | Tomlin (1966) | |
| | Barnhart et al. (1996) | |
| | Holmberg and Yuan (2003) | |
| | **Line planning problem** | Shortest/weighted path problem |
| | Borndörfer et al. (2007) | |
| $a_d^{\min} = 0 \wedge a_d^{\max} > 1$ | **Truss cutting problem/** | Travelling salesman problem |
| $\forall d \in D$ | **CSP with seq.-dep. cutting losses** | |
| | Lewis et al. (2011) (Solved heuristically) | |
| | Garraffa et al. (2016) (Solv. by heuristic CG) | |
| | **Single/parallel machine scheduling** | (Single machine scheduling problem) |
| | **for groups/classes of identical jobs** | |
| | Psaraftis (1980) (not solved with CG) | |
| | Byong-Hun and Jae-Ho (1990) (n.s.w. CG) | |
| | Webster and Baker (1995) (n.s.w. CG) | |

Table 10: Applications where the branch-and-bound column search algorithm BaB – SDE can be used as a solution algorithm for the pricing problem.

The broad, but certainly not exhaustive listing of problem classes in Tables 9 and 10 shows the wide applicability of branch-and-bound column search. As we have seen, branch-and-bound column search works particularly well when the columns that lead to an overall optimal solution are rather sparse, which means that the corresponding nodes have a small depth in the enumeration tree. In this case, the tree can be pruned relatively early using the bound provided by Theorem 3.8. This was the case, for example, with the problem class of the parallel machine scheduling problem in Section 5.3 or more precisely with the concrete use case of assigning transport orders in hospitals to employees from Section 5.4. Favourable transport sequences were found early in the tree here, as long sequences are more likely to lead to both greater delays and uneven utilisation of transporters.

## A.4 MIP Formulation for the Subproblem of Application I

A mixed-integer programming-based formulation of the pricing subproblem for Application I, the sequence-dependent cutting stock problem, can, for instance, be formulated as follows:

$$(\mathrm{SP}_{\mathrm{CS}}) \qquad \min_{x,y} \; c_p - \sum_{d \in D} \pi_d y_d$$

$$\text{s.t.} \quad \sum_{d \in D} x_{t^-,d} = 1 \tag{30}$$

$$\sum_{d \in D} x_{d,t^+} = 1 \tag{31}$$

$$\sum_{d_1 \in E} \sum_{d_2 \in D \setminus E} x_{d_1,d_2} + x_{d_2,d_1} \geqslant 2 \cdot z_E \qquad \forall E \subsetneq D : |E| \geqslant 1 \tag{32}$$

$$M \cdot z_E \geqslant \sum_{d \in E} y_d \qquad \forall E \subsetneq D : |E| \geqslant 1 \tag{33}$$

$$y_d = \sum_{(d_1,d_2) \in \delta^+(d_2)} x_{d_1,d_2} \qquad \forall d \in D \tag{34}$$

$$y_d \geqslant \sum_{(d_1,d_2) \in \delta^-(b)} x_{d_1,d_2} \qquad \forall d_2 \in D \tag{35}$$

$$\sum_{d \in D} l_d y_d + \sum_{(b,d) \in D^- \times D^+} c_{d_1,d_2} x_{d_1,d_2} \leqslant L \tag{36}$$

$$y_d, x_{d_1,d_2} \in \mathbb{Z}_+ \qquad \forall d_1, d_2 \in D \tag{37}$$

$$z_E \in \{0,1\} \qquad \forall E \subsetneq D : |E| \geqslant 1, \tag{38}$$

where the objective function minimizes the reduced costs $\bar{c}_p$. Further, the constraints (30)–(31) guarantee that every pattern possesses exactly one first and one last item. Inequality (32) is a so-called *Subtour Elimination Constraint* (*SEC*). It prevents the solution from containing infeasible subtours, i.e. subtours that are not connected to the source-target-path (path from $t^-$ to $t^+$). The formulation used is based on that of Dantzig et al. (1954). The original formulation of the SEC, i.e. $\sum_{b \in E} \sum_{d \in D \setminus E} x_{b,d} + x_{d,b} \geqslant 2$ for $E \subsetneq D$ and $|E| \geqslant 1$, would stipulate for all subsets $E$ that the number of item pairs $(b,d) \in D \times D$ that enter $E$ together with the number of item pairs $(b,d) \in D \times D$ that leave $E$ equals 2. However, this shall only apply to those subsets in which we have items that are part of the solution pattern, i.e. only for subsets $E$ where $\sum_{d \in E} y_d > 0$. Thus, the auxiliary variable $z_E$ in (32) and the additional constraint (33) are necessary. In practical computations, these two constraints are added in a lazy fashion, i.e. only if the computed solution contains infeasible subtours. Constraint (34) ensures that each item used has a preceding item, and Constraint (35) guarantees that each item has at most one subsequent item. The last inequality (36) make sure that the stock item size $L$ is not exceeded, taking into account the item sizes $l_d$ with $d \in D$ and the sequence-dependent cutting losses $c_{d_1,d_2}$ with $d_1, d_2 \in D^- \times D^+$. Finally, Constraints (37) and (38) define the domains of the variables.

## A.5 Algorithm for the Calculation of a Lower Bound on the Reduced Costs in Application I

Branch-and-bound column search uses lower bounds on the reduced costs of all descendants of a given node to prune the enumeration trees. These lower bounds are calculated with the help of the total maximal gain, see Definition 3.7 and Theorem 3.8. For the cutting stock problem with sequence-dependent cut losses, it is possible to derive an even stronger lower bound, see (18). Unfortunately, this lower bound proved to be too expensive computationally, such that we decided to employ a weaker bound which is easier to compute, but is nevertheless still stronger than the initial bound (14) based on the total maximal gain. Algorithm 5 describes how to calculate this lower bound, which we used for the computational study of the (SDCSP) in Section 5.2.

Algorithm 5 starts with sorting the items $d \in \tilde{D}$ by the reduced costs of the items as well as by the length of the items together with the minimal cutting loss that can arise before them, see Lines 5 and 6. Then it successively adds artificial items to the current pattern $p$, creating also artificial patterns $\hat{p}$, whose reduced costs are lower bounds on the reduced costs of the real descendant patterns of node $v_p$.

More precisely, this is done using the sorted lists SRC and SL. With list SRC, the algorithm successively adds the largest possible reduced cost $\bar{\pi}_{d_{\max}} - l_{d_{\max}}$ of all remaining items to the current reduced cost $\bar{c}_p$ of pattern $p$ (see Line 14). With list SL, we increase the length of pattern $l_{\hat{p}}$ only by the minimal length $l_{d_{\min}}$ and the corresponding smallest cut loss before item $d_{\min}$, see Line 15.

By the for-loops in Lines 8 and 10 as well as the if-statement in Line 13, it is checked which items can still be added to the current pattern $p$ and how often they can be appended to this pattern. Especially, Line 13 determines whether there is any progress possible with respect to the reduced costs (i.e. $\bar{\pi}_{d_{\max}} - l_{d_{\max}} > 0$) and whether it is still possible to obtain feasible patterns by adding further items to current pattern $p$, i.e. $\hat{l}_p + c^+_{\min} + l_{d_{\min}} + c^-_{\min} \leqslant L$.

If the reduced costs of any of these artificial patterns is negative (i.e. $\bar{c}_p - \text{gain} < 0$), the algorithm returns TRUE, which means that we can not prune the enumeration tree at node $v_p$. If no artificial pattern $\hat{p}$ with reduced costs is found, the algorithm returns FALSE, because then we can rule out that there is any pattern with negative reduced costs among the descendants of $v_p$.

**Algorithm 5:** CanSubtreeBePruned – SDCSP

```
Input  : Subset $\tilde{D} \subseteq D$, node $v_p = (A_{D,p}, w_{S,p}) \in V$, cut losses $c_{d_i,d_j}$ with $(d_i,d_j) \in S$, item length
         $l_d$, optimal dual solution $\bar{\pi}$, bounds $(a_d^{\min}, a_d^{\max})$ for $d \in D$ and stock item length $L$
Output: Lower bound on reduced costs for all nodes below the pattern $p$ in enumeration tree $T$.
1  Initialize:
2      $d_p \leftarrow \{d \in D \mid w_{(d,t),p} = 1\}$
3      $l_p \leftarrow \sum_{(d_i,d_j)\in S} c_{d_i,d_j} w_{(d_i,d_j),p} + \sum_{d\in D} l_d a_{d,p}$
4      $l_{\hat{p}} \leftarrow l_p - c_{d_p,t}$
5      SRC $\leftarrow$ sort item indices $d \in \tilde{D}$ by reduced costs $\bar{\pi}_d - l_d$ in list, descending.
6      SL $\leftarrow$ sort item indices $d \in \tilde{D}$ by $\left(l_d + \min\left\{c_{\bar{d},d} \;\middle|\; \bar{d} \in \tilde{D}\right\}\right)$ in list, ascending.
7      gain $\leftarrow 0$
8  for $k \in \{1, \dots, |\tilde{D}|\}$ do
9      $d_{\min} \leftarrow \mathrm{SL}_k$, $d_{\max} \leftarrow \mathrm{SRC}_k$
10     for $j = 0, \dots, a^{\max}_{d_{\min}} - a_{d_{\min},p}$ do
11         $c^{+}_{\min} \leftarrow \min\left\{c_{d_i,d_{\min}} \mid d_i \in D\right\}$
12         $c^{-}_{\min} \leftarrow \min\left\{c_{d_i,t} \mid d_i \in D\right\}$
13         if $\bar{\pi}_{d_{\max}} - l_{d_{\max}} > 0$ and $l_{\hat{p}} + c^{+}_{\min} + l_{d_{\min}} + c^{-}_{\min} \leqslant L$ then
14             gain $\leftarrow$ gain $+ \bar{\pi}_{d_{\max}} - l_{d_{\max}}$
15             $l_{\hat{p}} \leftarrow l_p + c^{+}_{\min} + l_{d_{\min}}$
16             $d_p \leftarrow d_{\min}$
17             if $\bar{c}_p -$ gain $< 0$ then
18                 return TRUE
19             end
20         end
21     end
22 end
23 return FALSE
```

## A.6 MIP Formulations of the Subproblems of Application II

The pricing subproblem of the first lexicographic level of Application II can be formulated as a mixed-integer program in the following fashion:

$$
\begin{aligned}
(\mathrm{SP}^1_{\mathrm{MS}}) \qquad \min_{x,y,z,w} \quad & \sum_{d\in D} \delta_d z_d - \sum_{d\in D} \bar{\pi}_d y_d + \bar{\pi}_i && && (39)\\
\text{s.t.} \quad & y_d = \sum_{\hat{d}\in D^-} x_{\hat{d},d} && \forall d \in D && (40)\\
& y_d = \sum_{\hat{d}\in D^+} x_{d,\hat{d}} && \forall d \in D && (41)\\
& \sum_{d\in D^+} x_{s,d} = 1 && && (42)\\
& \sum_{d\in D^-} x_{d,t} = 1 && && (43)\\
& w_d + p_d + s_{d,\hat{d}} - w_{\hat{d}} \leqslant (1 - x_{d,\hat{d}})\, W_{d,\hat{d}} && \forall (d,\hat{d}) \in S && (44)\\
& a_d y_d \leqslant w_d && \forall d \in D && (45)\\
& w_d \leqslant W\, y_d && \forall d \in D && (46)\\
& z_d \geqslant w_d - b_d && \forall d \in D && (47)\\
& a_s \leqslant w_s \leqslant W && && (48)\\
& x_{\hat{d},d} \in \{0,1\} && \forall (d,\hat{d}) \in S && (49)\\
& y_d \in \{0,1\} && \forall d \in D && (50)\\
& w_d, z_d \geqslant 0 && \forall d \in D, && (51)
\end{aligned}
$$

where the sets $D$, $D^-$, $D^+$ and $S$ are defined as in Section 5.3. The variable $y_d$ takes the value 1 if job $d$

is part of the newly generated machine plan, otherwise $y_d = 0$. Further, the variable $x_{\hat{d},d}$ equals 1 if the pair of jobs $(\hat{d}, d) \in S_2$ is part of the generated plan, otherwise $x_{\hat{d},d} = 0$.

The objective (39) of Subproblem $(\mathrm{SP}^1_{\mathrm{MS}})$ minimizes the reduced cost of the new column, which is derived from the dual problem $(\mathrm{DP}^1_{\mathrm{MS}})$. Further, Constraint (40) ensures that each job $d \in D$ that is part of the machine plan to be generated, i.e. where $y_d = 1$, has exactly one preceding job or is executed first in the job sequence of any machine $i \in I$. In the latter case, we have $\hat{d} = s$. Constraint (41) guarantees that every machine $i \in I$ executes exactly one subsequent job or switches into its final state $t$ after the execution of a job $d \in D^-$. The following equations (42) and (43) ensure that the initial state $s$ and the finale state $t$ are part of the constructed machine plan. In order to take into account the given restrictions on the execution times of the jobs, Constraints (44) to (48) are included into the model. The inequality (44) makes sure that the execution times of two successive jobs $(d, \hat{d}) \in S_2$ are correctly taken into account, also with regard to their processing times $p_d$ and set-up times $s_{d,\hat{d}}$. In addition, this inequality ensures that subtours are eliminated, as it forces the starting times of the jobs to be strictly increasing (cf. Bard and Rojanasoonthon (2006)). This statement is valid, because $p_d > 0$ holds for all $d \in D$ and due to the definition of $S_2$, where only edges $(s, d)$ are part of the set $S$ and no edges in the opposite direction, i.e. $(d, s) \notin S$ for all $d \in D$. According to Constraint (45), a job is never started before the earliest possible starting timestamp $a_d \in \mathbb{R}$. Inequality (46) fixes the starting time of every job $d \in D$ which is not part of the solution to $w_d = 0$. Possible delays of jobs are modelled via Constraint (47). Further, due to Constraint (47), and since $z_d \geqslant 0$ holds for all $d \in D$, it holds that $z_d = \max\{w_d - b_d, 0\}$ with $b_d$ as the latest possible starting time. Thus, we have $z_d = 0$ if no delay occurs in the execution of task $d$. Otherwise, $z_d$ takes the value of the corresponding delay in minutes. In the latter case, this violation of the corresponding time window is penalized in the objective function by the first summand $\sum_{d \in D} \delta_d z_d$. Finally, Constraints (49), (50) and (51) define the domains of the variables.

The subproblem does not solve the assignment part of the overall optimization problem, i.e. it does not decide which job is assigned to which machine. Rather, it only constructs schedules for one single given machine. Thus, it has to be solved separately for each machine $i \in I$.

## A.7 Algorithm for the Calculation of the Total Maximal Gain in Application II

In order to prune the enumeration trees within branch-and-bound column search, it is necessary to have strong lower bounds on the reduced costs. To this end, we compute the total maximal gain (see Definition 3.7) to obtain such a lower bound (cf. Theorem 3.8). Algorithm 6 shows one possible way for the calculation of the total maximal gain in all three lexicographic levels of Application II in Section 5.

Its input corresponds essentially to the parameters determining the total maximal gain in the third lexicographic level. We pass to the algorithm the current lexicographic level $k$, a subset $\tilde{D} \subseteq D$ corresponding to the current search breadth $\beta$, the current search depth $\ell$, the current node $v_p$ of the node set $V_i^k$ of agent $i \in I$ in the current lexicographic level $k$ and an optimal solution $(\bar{\pi}, \bar{\mu}, \bar{\lambda})$ to the current dual problem on the corresponding level $k$. If we call the algorithm on the first or second lexicographic level, the parameters $\bar{\mu}, \bar{\lambda}$ are initialized, but their values are not relevant for the computation of the total maximal gain.

Algorithm 6 is initialized by setting the gain g to zero for all $d \in \tilde{D}$. The total maximal gain tmg is initialized with $-\infty$. Afterwards, we calculate the gain corresponding to the lexicographic level $k$ for all jobs $d \in \tilde{D}$ with the help of the bounding functions $F^{\mathrm{MS1}}$, $F^{\mathrm{MS2}}$ and $F^{\mathrm{MS3}}$ together with constant $C_p^{\mathrm{MS2}}$ as well as the optimal dual solution $(\bar{\pi}, \bar{\mu}, \bar{\lambda})$, see Lines 4–14. By the properties of the bounding functions defined in Section 5.3, the calculated gain g is an upper bound on the value by which the reduced costs decrease when a given job $d$ is added to plan $p$. Then the job indices $d \in \tilde{D}$ are sorted by their respective gain $\mathrm{g}_d$ and stored in SRC in a descending order. Now, the total gains tg are calculated for different numbers of jobs $1, \ldots, (\ell - \sum_{d \in \tilde{D}} a_{d,p})$ that still can be appended to plan $p$ given the search depth $\ell$. For a given number of jobs $\ell_{\mathrm{temp}}$, the largest total gain tg is calculated by accumulating the largest gains $g_{\mathrm{SRC}_1}$ to $g_{\mathrm{SRC}_{\ell_{\mathrm{temp}}}}$ in Lines 18–19. Finally, the maximumof these total gains tg is determined in Line 27, and the algorithm returns this total maximal gain tmg of node $v_p \in V_i^k$ in lexicographic level $k$ for the current search depth $\ell$.

**Algorithm 6:** CalculateTotalMaximalGain – PMSP

```
Input  : Lexicographic level $k$, subset $\tilde{D} \subseteq D$, search depth $\ell$, node $v_p = (A_{D,p}, A_{I,p}, w_{S,p}) \in V_i^k$,
         optimal dual solution $(\bar{\pi}, \bar{\mu}, \bar{\lambda})$
Output: The total maximal gain of node $v_p \in V_i^k$ on lexicographic level $k$ for search depth $\ell$ in an
         enumeration tree $T_i^k$ of the (PMSP).
1  Initialize:
2      $g \leftarrow (0)_{d \in \tilde{D}}$.
3      tmg $\leftarrow -\infty$.
4  for $d \in \tilde{D}$ do
5      if $k = 1$ then
6          $g_d \leftarrow \bar{\pi}_d - F_d^{\mathrm{MS1}}(1, v_p)$.
7      end
8      else if $k = 2$ then
9          $g_d \leftarrow \bar{\pi}_d - F_d^{\mathrm{MS2}}(1, v_p) - \bar{\mu}_1 F_d^{\mathrm{MS1}}(1, v_p)$
10     end
11     else if $k = 3$ then
12         $g_d \leftarrow \bar{\pi}_d - \bar{\lambda}_i - \bar{\mu}_1 F_d^{\mathrm{MS1}}(1, v_p) - \bar{\mu}_2 F_d^{\mathrm{MS2}}(1, v_p)$
13     end
14 end
15 SRC $\leftarrow$ sort job indices $d \in \tilde{D}$ with $a_{d,p} = 0$ by gain $g_d$, descending.
16 for $\ell^{\mathrm{temp}} \in \{1, \dots, (\ell - \sum_{d \in \tilde{D}} a_{d,p})\}$ do
17     tg $\leftarrow 0$.
18     for $j \in \{1, \dots, \ell^{\mathrm{temp}}\}$ do
19         tg $\leftarrow$ tg $+ g_{\mathrm{SRC}_j}$.
20     end
21     if $k = 2$ then
22         tg $\leftarrow$ tg $- C_p^{\mathrm{MS2}}$.
23     end
24     else if $k = 3$ then
25         tg $\leftarrow$ tg $- \bar{\mu}_2 C_p^{\mathrm{MS2}}$.
26     end
27     tmg $\leftarrow \max\{$tg, tmg$\}$.
28 end
29 return tmg
```

For the computational study in Section 5.4, we used a more sophisticated bound introduced by Westphal and Krumke (2008). This bound is indeed computationally more expensive, but it often allows to prune the enumeration trees earlier in the enumeration process, such that the branch-and-bound column search has to enumerate less columns explicitly (see Bärmann et al. (2022) for details).