An exact column-generation approach for the lot-type design problem

Miriam Kießling Sascha Kurz
Jörg Rambau

Lehrstuhl für Wirtschaftsmathematik
Universität Bayreuth
Germany.
Tel.: +49-921-557353, Fax: +49-921-557352
miriam.kiessling@googlemail.com
{sascha.kurz,joerg.rambau}@uni-bayreuth.de

July 7, 2020

Abstract

We consider a fashion discounter distributing its many branches with integral multiples from a set of available lot-types. For the problem of approximating the branch and size dependent demand using those lots we propose a tailored exact column generation approach assisted by fast algorithms for intrinsic subproblems, which turns out to be very efficient on our real-world instances as well as on random instances.

Keywords: p-median; facility location; lot-type design; column generation
Mathematics subjects classification: 90C06; 90C90; 90B99

1 Introduction

Due to small profit margins of most fashion discounters, applying OR methods is mandatory for them. In order to reduce the handling costs and the error proneness in the central warehouse, our business partner orders all products in multiples of so-called *lot-types* from the suppliers and distributes them without any replenishing to its branches. A lot-type specifies a number of pieces of a product for each available size, e.g., lot-type $(1, 2, 2, 1)$ means two pieces of size M and L, one piece of size S and XL, if the sizes are $(\mathrm{S}, \mathrm{M}, \mathrm{L}, \mathrm{XL})$.

We want to solve the following approximation problem: which (integral) multiples of which (integral) lot-types should be supplied to a set of branches in order to meet a (fractional) expected demand as closely as possible? We call this specific demand approximation problem the *lot-type design problem (LDP)* in [12]. In that paper, also a basic model for the LDP was introduced, accompanied by an integer linear programming formulation and a tailored heuristic, which turned out to perform very well for the real-world data of our partner.

The problem that remained unaddressed in [12] is the following: For many practical instances the set of applicable lot-types and thus the number of variables is so large that the ILP formulation

from [12] cannot be solved directly.[1] In this paper, we therefore propose a new algorithm *Augmented Subproblem Grinding (ASG)* based on dynamic model generation by the branch-and-price principle. The main result is twofold, experimentally obtained on a benchmark set of 860 real-world instances from the running production in 2011: First, ASG is able to solve LDPs with millions and billions of lot-types (for details see below) in 10 to 100 seconds to optimality. Second, for smaller instances ASG is faster by a factor of 100 (more for larger instances) than the solution of the statically generated ILP model for LDP with a state-of-the-art MILP solver (in this paper: `cplex 12.9`).

1.1 Related work

We start with a brief overview of related literature from the application side. Apparel supply chains pose many challenges. A generic survey would be beyond the scope of this paper. An overview of some aspects (though not specifically related to the special subject of this paper) can be found in [2]. Let us mention two very difficult partial challenges. The first problem is the estimation of the demand, since many apparel products are never replenished. Therefore, the sales data only yield a right-censored observation of the demand. Moreover, apparel comes in different sizes. A branch of a fashion discounter is supplied by only very few (one, two, or three) pieces per size, and the whole supply must have been sold at the end of the sales-period (12 weeks for our partner), possibly by heavy mark-downs. Thus, it is difficult to know, how and when the sales data should be measured. The second problem is the integrality of the supply in the presence of small fractional expected demands. That means, that even if the fractional expected demand for an item in a size were known, we face the approximation problem of which integral supply is best for which expected demand. Our industrial partner uses a system of ordering pre-packed assortments (lots) of items with only a small variation of possible size-combinations (lot-types). This restricts the approximation problem even further (the LDP). See, e.g., [19] for a discussion and a field study of both aspects combined.

The exact lot-type design problem was introduced for the first time by two of the authors in [12]. Later, it was connected to the sales-process more accurately by modeling the mark-downs explicitly as recourse actions of a two-stage stochastic program in [16]. Since in that paper, the solution of the LDP is an important subroutine, the investigations in this paper have direct consequences in the applicability of the results in [16] on instances with many applicable lot-types.

The LDP is a special case of a more general problem setting: assortment optimization asks for good combinations of items with arbitrary attributes (not only size), and the customer substitutes between them, e.g., buys a red instead of a blue T-shirt. See [11] for various aspects of the problem including demand estimation and heuristic algorithms. In [3] it is shown that the general assortment problem is difficult and approximation algorithms are analyzed.

Let us now turn to related literature from the methodological point of view, mainly mixed integer linear programming (MILP)

Our new method is a specialization of the *branch-and-price* principle for mixed-integer linear programming problems (MILP). In the branch-and-price framework, not all variables are added to an MILP in the beginning. Instead, in each branch-and-bound node of the MILP-solution process, variables are added dynamically by solving the pricing problem until a computational optimality proof can be completed. A standard survey for branch-and-price algorithms for large-scale integer programming problems can be found in [21]. One of the most influential early systematic discussions for the branch-and-price principle for MILP was presented in [5]. In the

[1] E.g. for 12 different sizes, which is reasonable for lingerie or children's clothing, there are 1 159 533 584 different lot-types, if we assume that there should be at most 5 items of each size and that the total number of items in a lot-type should be between 12 and 30.

same spirit, in [32] a (quite sophisticated) branching scheme based on a classification of columns is formulated in a problem-independent way. Our new method by-passes the challenges attacked in that paper and is therefore, in our opinion, easier to implement.

If not only variables but also restrictions are added dynamically, one is concerned with *branch-price-and-cut* algorithms (see [28] for a more recent practical application, [22] for a methodological survey and [29] for a more recent comprehensive treatment). There are two principally different reasons for adding restrictions dynamically. First, they can be inevitable, when there are model restrictions whose support solely consists of variables that are dynamically generated (model restrictions). Such restrictions leading to *row-dependent columns* are discussed, e.g., in [10, 25]. Second, they can be desirable, when they help to close the integrality gap in the master problem (cuts). In the second case, one speaks of *robust* cuts if the cuts do not change the complexity of the pricing problem (see [26]). In contrast to this, *non-robust* cuts need to be observed in the pricing problem and can increase its complexity (see [15]). In the case of the LDP, we are concerned with inevitable and new restrictions that have a significant influence on the pricing problem, as we will see below. In [22, Section 3.2] it is discussed how, in principle, the problem of determining reasonable dual variables for future cuts in the master problem should be attacked. However, this is essentially a problem that until today is more successfully solved for each particular application separately. One of our main results is a solution to this *dual lifting problem* for the LDP.

Another important partial problem is the determination of feasible integer solutions. If there is no tight problem-specific heuristic available, generic primal heuristics can be used, e.g., based on diving down the branch-and-bound tree like in [30]. In our case, a fast and tight heuristics is fortunately known from [12]. In [31] it was demonstrated how machine learning can accelerate the time spent in the pricing problem by learning a dual bound from previous iterations. Since for the LDP the time spent in the pricing problem is not dominant, we have not (yet) tried this idea for the LDP.

The LDP is related to the p-median and the facility location problem: for computational results on large instances of the p-median problem we refer the reader to [4, 7, 17, 27]. Other algorithmic approaches can, e.g., be found in [14, 24, 13, 8, 1, 9, 23]. The additional cardinality constraint of the LDP prevents a direct application of specialized p-median algorithms. Therefore, we follow the concept of the more generic branch-and-price framework.

1.2 Contribution

We present a new algorithmic procedure named *Augmented Subproblem Grinding (ASG)* based on the branch-and-price principle, albeit with a new strategy. The new results are:

- Theoretically, we construct for the LDP a characteristic lifting of the dual variables that yields, in a certain sense, the tightest possible information about promising new columns.
- Algorithmically, based on the previous contribution, we devise a column-generation procedure that generates promising combinations of columns instead of sets of unrelated columns. These combinations avoid the generation of columns whose negative reduced costs can be compensated by an easy modification of the duals of the reduced master problem.
- Practically, ASG uses a (to the best of our knowledge) new branching scheme: at any time there are only two branches, one of which is solved immediately by the black-box MILP-solver, whereas the other excludes all solutions of the solved branch by a single set-covering constraint with dynamically growing support. Thus, there no branch-and-price-and-bound tree must be maintained. Consequently, ASG is easier to implement than ordinary branch-and-price. Moreover, it exploits the employed black-box MILP-solver

and its branching intelligence. This way, ASG will take profit of any improvement in the MILP-solver technology without modifications in its own code.

- Experimentally, the efficiency and effectiveness of ASG is shown based on 860 real-world instances from the daily production of our industry partner from 2011: For the task to find an optimal solution and prove its optimality, ASG reduces the cpu-times to 0.67 % and the numbers of columns to 0.18 % on average compared to solving the full ILP model by a state-of-the-art MILP-solver. Moreover, for the largest 100 instances with 134,596 through 1,198,774,720 applicable lot-types (unsolved prior to this paper), ASG needs less than 30 seconds and 10,000 columns on average. This makes ASG perfectly applicable in practice. We confirm the success by a second randomized benchmark set in order to avoid possible artifacts caused by a hidden structure of our real-world instances.

1.3 Outline of the paper

A formal problem statement is given in Section 2, followed by an ILP model in Section 3. In Section 4 we discuss the theoretical foundation of our algorithm that is presented in Section 5. We show computational results on real-world data and on random data in Section 6, before we conclude with Section 7.

2 Formal problem statement

We consider the distribution of supply for a single product and start with the formal problem statement in the deterministic context before we extend it to a stochastic version.

2.1 The deterministic lot-type design problem (LDP)

Data. Let $\mathcal{B}$ be the set of branches, $\mathcal{S}$ be the set of sizes, and $\mathcal{M} \subset \mathbb{N}$ be an interval of possible multiples. A *lot-type* is a vector $l = (l_s)_{s\in\mathcal{S}} \in \mathbb{N}^{|\mathcal{S}|}$. For given integers $\min_c \leq \max_c$ (the *component bounds*) and $\min_t \leq \max_t$ (the *type-bounds*), a lot-type l is *applicable*, if $\min_c \leq l_s \leq \max_c$ for all $s \in \mathcal{S}$ and $\min_t \leq |l| := \sum_{s\in\mathcal{S}} l_s \leq \max_t$. The set of applicable lot-types is called a *set of lot-types parametrized by* $\min_c$, $\max_c$, $\min_t$, *and* $\max_t$ and is denoted by $\mathcal{L}$. Note, that specifying a parametrized set of lot-types and specifying an explicit list of lot-types differ in terms of the complexity of models. A model that is polynomial in the number of lot-types is polynomially-sized if the input is an explicit list of lot-types whereas it may be exponentially-sized if the input is a parametrized set of lot-types.

The motivation for parametrized sets of lot-types is as follows. The cost reduction induced by ordering apparel in pre-packed lots in the Far East (low wage) results from the reduction of the number of picks in the central warehouse in Europe (high wage). On the one hand, this reduction is only significant if there are enough pieces in the used lot-type. This can be enforced by setting $\min_t$ large enough. On the other hand, a single pick of a lot is only possible if there are not too many pieces in the used lot-type. This can be enforced by setting $\max_t$ small enough. Moreover, if a product is advertised in a newspaper or the like, each branch must have at least one piece in each size by legal regulations. This can be guaranteed by setting $\min_c$ to one. For the sake of a clean presentation at the sales start, it may be desirable to enforce that no single size is extremely over-represented. This can be enforced by setting $\max_c$ small enough. If $\min_c = 0$ and $\max_c = \max_t$, then $\min_c$ and $\max_c$ do not impose any restriction beyond the restrictions imposed by $\min_t$ and $\max_t$. In practice, the lot-type parameters $\min_t$, $\max_t$, $\min_c$, and $\max_c$

depend on the product. For example, it is easily possible to pick a lot containing ten T-shirts whereas it is undesirable to pick a lot of ten winter coats.

There is an upper bound $\overline{I}$ and a lower bound $\underline{I}$ given on the total supply over all branches and sizes. Moreover, there is an upper bound $k \in \mathbb{N}$ on the number of lot-types used. By $d_{b,s} \in \mathbb{Q}_{\geq 0}$ we denote the deterministic demand at branch b in size s.

Decisions. Consider an assignment of a unique lot-type $l(b) \in \mathcal{L}$ and an assignment of a unique multiplicity $m(b) \in \mathcal{M}$ to each branch $b \in \mathcal{B}$. These data specify that $m(b)$ lots of lot-type $l(b)$ are to be delivered to branch b. These decisions induce inventories $I_{b,s}(l,m) := m(b)l(b)_s$ for each branch $b \in \mathcal{B}$ and each size $s \in \mathcal{S}$ and a total inventory of $I(l,m) := \sum_{b\in\mathcal{B}} m(b)|l(b)|$ over all branches and sizes.

Objective. The goal is to find a subset $L \subseteq \mathcal{L}$ of at most k lot-types and assignments $l(b) \in L$ and $m(b) \in \mathcal{M}$ such that the total supply is within the bounds $\left[\underline{I}, \overline{I}\right]$, and the total deviation between supply and demand is minimized. More specifically, the *deviation* $\Delta_{b,s}(l,m)$ of supply and demand for branch b and size s given a decision $l(b)$, $m(b)$ for all $b \in \mathcal{B}$ is defined by $\Delta_{b,s}(l,m) := |d_{b,s} - I_{b,s}(l,m)|$. The *cost* of the decisions $l(b)$, $m(b)$ for branch $b \in \mathcal{B}$ is defined as $c_b(l,m) := \sum_{s\in\mathcal{S}} \Delta_{b,s}(l,m)$. The goal is to minimize the total cost $c(l,m) := \sum_{b\in\mathcal{B}} c_b(l,m)$.

The resulting optimization problem is the *Lot-Type Design Problem (LDP)*. It is strongly related to the LDP in [12]. The difference is that in [12] the set of applicable lot-types was given by an explicit list $\mathcal{L}$.

2.2 The stochastic lot-type design problem (SLDP)

The SLDP differs from the LDP only in the following. We consider a set Ω of *scenarios* (for the success of the product). For each scenario $\omega \in \Omega$ we denote by p^ω its probability and with $d^\omega_{b,s} \in \mathbb{Q}_{\geq 0}$ the demand at branch b in size s in scenario ω for all $b \in \mathcal{B}$ and $s \in \mathcal{S}$. The goal then is to minimize the expected total deviation between supply and demand. The remaining ingredients are the same as for the LDP. In particular, the LDP and the SLDP differ only in the objective function.

We call this single-stage stochastic optimization problem the *Stochastic Lot-Type Design Problem (SLDP)*.

In this paper, we assume for an SLDP that there is a designated *nominal scenario* $\hat{\omega} \in \Omega$ that models a customer demand that is considered "normal". For example, in the data of our industrial partner there are usually three scenarios that stem from the classification of a product as "Renner" (German designation for a product in high demand), "Normal" (= normal demand), and "Penner" (German designation for a product in low demand). In general, if a nominal scenario is not given explicitly in an SLDP, the scenario of expected demands is added to the set of scenarios and defined as the nominal scenario.

Each SLDP with a nominal scenario induces two LDPs with identical constraint sets as the SLDP: The LDP with deterministic demands corresponding the the nominal scenario is called the *nominal LDP* of the SLDP. The LDP where we set the deterministic deviations $\Delta(l,m)$ to the expected deviations $\overline{\Delta}(l,m) := \sum_{\omega\in\Omega} p^\omega \Delta^\omega(l,m)$ with $\Delta^\omega(l,m) := \sum_{b\in\mathcal{B}} \sum_{s\in\mathcal{S}} \Delta^\omega_{b,s}(l,m)$ and $\Delta^\omega_{b,s}(l,m) := |d^\omega_{b,s} - I_{b,s}(l,m)|$ is called the *equivalent LDP* of the SLDP. The equivalent LDP of an SLDP is equivalent to the SLDP since both constraints and objective functions are identical by the linearity of expectation. In other word, for any SLDP the equivalent LDP is a *deterministic equivalent program* [6, Section 2.4] for the SLDP.

In any case, any algorithm for the LDP can be transformed into an algorithm for the SLDP by replacing each computation of $\Delta_{b,s}(l,m)$ in the LDP algorithm by the computation of $\overline{\Delta}_{b,s}(l,m) = \sum_{\omega\in\Omega} p^\omega \Delta^\omega_{b,s}(l,m)$. This incurs an overhead proportional to the number of scenarios.

In contrast to the equivalent LDP, the nominal LDP, in general, has a smaller objective than

the SLDP, since the deviation is non-linear and convex in the demands. We will find out that this difference in the objective values is significant in section 6. We will also evaluate the standard measure for the relevance of the difference in the *optimal solutions* of the SLDP and the nominal LDP: The *value of the stochastic solution (VSS)* [6, Section 4.2]. It turns out that this difference is far less pronounced.

3 Modelling

We must choose a set L of selected lot-types with $|L| \leq k$ and assign to each branch $b \in \mathcal{B}$ a lot-type $l = l(b) \in L$ and a multiplicity $m = m(b) \in \mathcal{M}$.

Remark 1. *One possible attempt to model the (S)LDP is to use variables for the supplies for each size separately. Such a model is given in Appendix A. It has the advantage that its size is polynomial in the input size, even for a parametrized set of lot-types. However, it exhibits an extremely large integrality gap (see Appendix A for details). In this paper, we follow the path of using a tighter model at the expense of exponentially many variables. The model was introduced in [12] and is polynomially-sized if the input is an explicit list of lot-types. It is exponentially sized if the input is a parametrized set of lot-types.*

In order to model the SLDP as an integer linear program we use binary assignment variables $x_{b,l,m}$, i.e, $x_{b,l,m} = 1$ if branch b is supplied with lot-type l in multiplicity m and $x_{b,l,m} = 0$ otherwise, see (6). For the used lot-types we use binary selection variables y_l, i.e., $y_l = 1$ if $l \in L$ and $y_l = 0$ otherwise, see (7). Recall that we utilize $|l| := \sum_{s \in \mathcal{S}} l_s$ for the number of pieces contained in lot-type l.

$$\min \quad \sum_{b \in \mathcal{B}} \sum_{l \in \mathcal{L}} \sum_{m \in \mathcal{M}} c_{b,l,m} \cdot x_{b,l,m} \tag{1}$$

$$s.t. \quad \sum_{l \in \mathcal{L}} \sum_{m \in \mathcal{M}} x_{b,l,m} = 1 \qquad \forall b \in \mathcal{B} \tag{2}$$

$$\sum_{m \in \mathcal{M}} x_{b,l,m} \leq y_l \qquad \forall b \in \mathcal{B}, l \in \mathcal{L} \tag{3}$$

$$\sum_{l \in \mathcal{L}} y_l \leq k \tag{4}$$

$$\underline{I} \leq \sum_{b \in \mathcal{B}} \sum_{l \in \mathcal{L}} \sum_{m \in \mathcal{M}} m \cdot |l| \cdot x_{b,l,m} \leq \overline{I} \tag{5}$$

$$x_{b,l,m} \in \{0,1\} \qquad \forall b \in \mathcal{B}, l \in \mathcal{L}, m \in \mathcal{M} \tag{6}$$

$$y_l \in \{0,1\} \qquad \forall l \in \mathcal{L}, \tag{7}$$

where $c_{b,l,m} = \sum_{\omega \in \Omega} p^\omega \cdot \sum_{s \in \mathcal{S}} \left| d^\omega_{b,s} - m \cdot l_s \right| \geq 0$. With this, the stated objective function models the "costs" of the deviation between supply and demand as motivated in the previous section. The deterministic LDP is also covered by defining Ω as a single-element set. Inequality (2) guarantees that each branch is supplied with exactly one lot-type and one multiplicity. By inequality (3) we ensure that $y_l = 1$ whenever there is a branch b which is supplied with lot-type l in some multiplicity m. It may happen that $y_l = 1$ while no branch is supplied with lot-type l. This is logically no problem at this point. However, later in our algorithm, we will add constraints to exclude this case. By inequality (4) we ensure that at most k lot-types are used, i.e., those with $y_l = 1$. The lower and the upper bound on the total supply is modeled by inequality (5).

The model reflects the correspondence between an SLDP and its equivalent LDP by the fact that it models both at the same time.

Remark 2. *It is easy to generate an SLDP with no feasible solution. For example, if supplying each branch with* $\min_t$ *pieces exceeds the upper bound* $\overline{I}$ *on the total supply, no feasible lot-type design can be found. Even a feasible instance may be ill-posed from a practical point of view: if the (expected) total demand does not satisfy the lower and upper bounds on the total supply, then the theoretically most desirable supply with exactly the demand will be infeasible for the model. Still, there may be feasible solutions missing the demand completely. Just think of the extreme case of strictly positive demands and* $\overline{I} = \min_c = \min_t = 0$*; a delivery of zero is then formally feasible. Moreover, the smaller the difference between the lower and upper bounds on the total supply becomes, the less important is the demand consistency of the supply, which is the original goal. Think again of an extreme case of identical lower and upper bounds and only one allowed lot-type: then, e.g., if the bound is not divisible by the number of branches, there is no feasible solution – a condition that our industrial partner did not intend to impose. In practice, our industrial partner preferred to be notified about such anomalies in the input data rather than to receive an artificial solution. The usual set-up was to use the expected demand plus/minus 10% for the bounds.*

In order to avoid anomalies in the input data, we specify in the following *consistent instances* of the SLDP and the LDP, respectively. Technically, we define a consistent instance in such a way that a natural and easy heuristics (the ALH heuristics, see algorithm 6) always finds a feasible solution (see section 5.3 for details). The concept is to report inconsistent instances at the beginning and let the core algorithm only deal with SLDP instances that are consistent in the following sense:

Definition 1. An SLDP instance is *consistent*, if

(i) it is feasible

(ii) the total nominal demand lies in the cardinality interval, i.e., $\underline{I} \leq \sum_{b\in\mathcal{B}} \sum_{s\in\mathcal{S}} d^{\hat{\omega}}_{b,s} \leq \overline{I}$ (demand consistency)

(iii) the total-cardinality flexibility is at least the lot-type-cardinality flexibility, i.e., $\overline{I} - \underline{I} \geq \max_t - \min_t$ (cardinality consistency)

4 The theoretical foundations

In this subsection, we present the underlying theory in more detail. Since for each SLDP we can consider the equivalent LDP, we will, without loss of generality, formulate all results in terms of the LDP with deterministic demands $d_{b,s}$ to simplify the notation.

By using only a small number of variables, the master problem for the pricing phase (MP) can be restricted to a manageable size, resulting in the restricted master problem (RMP). More specifically: Let $\mathcal{L}'' \subseteq \mathcal{L}$ be the subset of lot-types used in the previously solved RSLDP, which is initially empty. We then consider in the RMP a (small) subset $\mathcal{L}' \subseteq \mathcal{L}$ of the lot-types containing $\mathcal{L}''$. For each branch $b \in \mathcal{B}$ we consider a subset $L_{\mathcal{L}'}(b) = L(b) \subseteq \mathcal{L}'$ of these lot-types and for each $l \in L_{\mathcal{L}'}(b)$ we consider only a subset $M_{\mathcal{M}}(b,l) = M(b,l) \subseteq \mathcal{M}$ of the multiplicities.

For the following, we denote by SLDP the ILP model of the previous section augmented by

- a set covering constraint of the form $\sum_{l\in\mathcal{L}\setminus\mathcal{L}'} y_l + p \geq 1$ with a slack variable p whose use is essentially *prohibited* by a very large cost coefficient P, set to some upper bound for

the optimal SLDP value; $\mathcal{L}'$ is initially empty and will grow throughout the algorithm; this constraint will be used to guarantee that newly generated lot-types need to enter any solution with a total weight of at least one.

- a reverse coupling constraint $y_l \leq \sum_{m\in\mathcal{M},b\in B} x_{b,l,m}$ for each lot-type $l \in \mathcal{L}$; this constraint ensures that a lot-type is only selected if it is assigned to some branch with some multiplicity, which will incur some cost $c_{b,l,m}$.

Moreover, we denote by RSLDP (= restricted SLDP) the SLDP restricted to some smaller set of lot-types, by MP (= master problem) the linear programming relaxation of the SLDP including the set covering constraint, by RMP (= restricted master problem) the MP restricted to some smaller set of columns, and by PP (= pricing problem) the pricing problem to determine whether there exist columns with negative reduced costs.

The restricted master problem (RMP) then reads as follows:

$$\min \quad \sum_{b\in\mathcal{B}} \sum_{l\in L(b)} \sum_{m\in M(b,l)} c_{b,l,m} \cdot x_{b,l,m} + Pp \qquad \text{(duals:)} \tag{8}$$

$$s.t. \quad \sum_{l\in L(b)} \sum_{m\in M(b,l)} x_{b,l,m} = 1 \qquad \forall b \in \mathcal{B} \qquad (\alpha_b) \tag{9}$$

$$-\sum_{m\in M(b,l)} x_{b,l,m} + y_l \geq 0 \qquad \forall b \in \mathcal{B}, l \in L(b) \qquad (\beta_{b,l}) \tag{10}$$

$$-\sum_{l\in\mathcal{L}'} y_l \geq -k \qquad (\gamma) \tag{11}$$

$$\sum_{b\in\mathcal{B}:l\in L(b)} \sum_{m\in M(b,l)} x_{b,l,m} - y_l \geq 0 \qquad \forall l \in \mathcal{L}' \qquad (\delta_l) \tag{12}$$

$$\sum_{l\in\mathcal{L}'\setminus\mathcal{L}''} y_l + p \geq 1 \qquad (\mu) \tag{13}$$

$$\sum_{b\in\mathcal{B}} \sum_{l\in L(b)} \sum_{m\in M(b,l)} m \cdot |l| \cdot x_{b,l,m} \geq \underline{I} \qquad (\phi) \tag{14}$$

$$-\sum_{b\in\mathcal{B}} \sum_{l\in L(b)} \sum_{m\in M(b,l)} m \cdot |l| \cdot x_{b,l,m} \geq -\overline{I} \qquad (\psi) \tag{15}$$

$$x_{b,l,m} \geq 0 \qquad \forall b \in \mathcal{B}, \ \forall l \in L(b), \ \forall m \in M(b,l) \tag{16}$$

$$y_l \geq 0 \qquad \forall l \in \mathcal{L}'. \tag{17}$$

Using the indicated dual variables, the dual restricted master problem (DRMP) is then given by:

$$\max \quad \sum_{b\in\mathcal{B}} \alpha_b - k\gamma + \underline{I}\phi - \overline{I}\psi + \mu \qquad (primals:) \tag{18}$$

$$s.t. \quad \alpha_b - \beta_{b,l} + \delta_l + m|l|\phi - m|l|\psi \leq c_{b,l,m} \qquad \forall b \in \mathcal{B}, \ \forall l \in L(b), \ \forall m \in M(b,l) \qquad (x_{blm}) \tag{19}$$

$$\sum_{b\in\mathcal{B}\,:\,l\in L(b)} \beta_{b,l} - \gamma - \delta_l + \mu \leq 0 \qquad \forall l \in \mathcal{L}' \qquad (y_l) \tag{20}$$

$$\mu \leq P \qquad (p) \tag{21}$$
$$\alpha_b \in \mathbb{R} \qquad \forall b \in \mathcal{B} \tag{22}$$
$$\beta_{b,l} \geq 0 \qquad \forall b \in \mathcal{B}, l \in L(b) \tag{23}$$
$$\delta_l \geq 0 \qquad \forall l \in \mathcal{L}' \tag{24}$$
$$\gamma, \phi, \psi \geq 0 \tag{25}$$
$$\mu \geq 0 \qquad . \tag{26}$$

In the following, we name all constraints by the names of the variables dual to them. That is, the DMP consists of x-, y-, and p-constraints whereas the MP consists of α-, β-, γ-, δ-, μ-, ϕ-, and ψ-constraints.

The pricing problem (PP) is defined by finding those constraints in the dual (DMP) of the unrestricted master problem (MP) that are most violated in the DMP by the current solution of the DRMP.

Note that each feasible solution of the RMP induces a feasible solution of the MP via a *lifting* by zeroes in all missing components. Whenever the addition of variables to the RMP requires the introduction of new constraints (like in our case), any exact formulation of the pricing problem must be based on a *lifting* of the dual variables. The following notions make the formulation of our main result easier.

Definition 2. Let the RMP contain the slack variable p and the variables and constraints for all lot-types $l \in \mathcal{L}'$ and assignments (b, l, m) for all $b \in \mathcal{B}$, $l \in L(b) \subseteq \mathcal{L}'$, and $m \in M(b, l) \subseteq \mathcal{M}$. At times we use for this the short-hand notation $(\mathcal{L}', L, M) := \big\{(b, l, m) : b \in \mathcal{B}, l \in L(b) \subseteq \mathcal{L}', m \in M(b, l)\}$, in which we consider L and M as set-valued functions. Moreover, let (x, y, p) be a feasible solution for the RMP and $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ a feasible solution for the DRMP with identical objective value. In particular, (x, y, p) and $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ are optimal for the RMP and the DRMP, respectively.

The *canonical lifting* of (x, y, p) in the RMP is a complete set of primal variables $\big(\bar{x}, \bar{y}, \bar{p}\big)$ for all $b \in \mathcal{B}$, $l \in \mathcal{L}$, and $m \in \mathcal{M}$ that arises from (x, y, p) by adding zeroes in the missing components. Similarly, the *canonical lifting* of the dual variables $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ in the DRMP is a complete set of dual variables $(\bar{\alpha}, \bar{\beta}, \bar{\gamma}, \bar{\delta}, \bar{\mu}, \bar{\phi}, \bar{\psi})$ for all $b \in \mathcal{B}$, $l \in \mathcal{L}$, and $m \in \mathcal{M}$ that arises from $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ by adding zeroes in the missing components.

A *cost-invariant lifting* of the dual variables $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ in the DRMP is a complete set of dual variables $(\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\delta}, \hat{\mu}, \hat{\phi}, \hat{\psi})$ with the following properties:

(i) The restriction of $(\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\delta}, \hat{\mu}, \hat{\phi}, \hat{\psi})$ to $(\mathcal{L}', L, M)$ equals $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$.

(ii) The objective in the DMP of $(\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\delta}, \hat{\mu}, \hat{\phi}, \hat{\psi})$ equals the objective in the DRMP of $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$.

The *(uncompensated) reduced cost* $\bar{c}_{b,l,m}$ of a variable $x_{b,l,m}$ is defined as its reduced cost with respect to the canonical lifting, i.e.,

$$\bar{c}_{b,l,m} := c_{b,l,m} - \bar{\alpha}_b + \bar{\beta}_{b,l} - \bar{\delta}_l - m|l|(\bar{\phi} - \bar{\psi}). \tag{27}$$

Given a cost-invariant lifting $(\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\delta}, \hat{\mu}, \hat{\phi}, \hat{\psi})$, the *compensated reduced cost* $\hat{c}_{b,l,m}$ of a variable $x_{b,l,m}$ is defined as its reduced cost with respect to the cost-invariant lifting, i.e.,

$$\hat{c}_{b,l,m} := c_{b,l,m} - \hat{\alpha}_b + \hat{\beta}_{b,l} - \hat{\delta}_l - m|l|(\hat{\phi} - \hat{\psi}). \tag{28}$$

We call a cost-invariant lifting *fully compensating* if all x-variables have non-negative compensated reduced costs.

By weak duality, we have the following:

Observation 1. *Assume a cost-invariant lifting of dual variables for the DRMP is feasible for the DMP. Then, the canonical lifting of an optimal solution to the RMP is optimal for the MP. Moreover, for any cost-invariant lifting we have: A cost-invariant lifting satisfies all x-constraints if and only if it is fully compensating.*

The (PP) seeks for new variables with minimal reduced costs in the MP and formally reads as follows for any cost-invariant lifting:

$$\min\Big\{ \min\big\{c_{b,l,m} - \hat{\alpha}_b + \hat{\beta}_{b,l} - \hat{\delta}_l - m|l|(\hat{\phi} - \hat{\psi}) : b \in \mathcal{B}, l \in \mathcal{L}, m \in \mathcal{M}\big\},$$
$$\min\big\{-\sum_{b\in\mathcal{B}} \hat{\beta}_{b,l} + \hat{\gamma} + \hat{\delta}_l - \hat{\mu} \qquad : l \in \mathcal{L}\big\}\Big\}. \tag{29}$$

Our idea is now to use a special cost-invariant lifting and check whether or not it is fully compensating and feasible for the DMP. We use the usual notations $(x)^+ := \max\{x, 0\}$ and $(x)^- := \max\{-x, 0\} = -\min\{x, 0\}$, componentwise.

Definition 3. For an optimal solution $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ of the DRMP we define the *characteristic lifting* of dual variables by

$$\check{\beta}_{b,l} := \begin{cases} \beta_{b,l} & \text{if } l \in L(b), \\ \big(\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big)^- & \text{if } l \in \mathcal{L}' \setminus L(b) \cup \mathcal{L} \setminus \mathcal{L}', \end{cases} \tag{30}$$

$$\check{\delta}_l := \begin{cases} \delta_l & \text{if } l \in \mathcal{L}', \\ \big(\min\limits_{\substack{b\in\mathcal{B}\\ m\in\mathcal{M}}} \bar{c}_{b,l,m}\big)^+ & \text{if } l \in \mathcal{L} \setminus \mathcal{L}'. \end{cases} \tag{31}$$

Moreover, define as an abbreviation

$$\check{c}_l := \begin{cases} -\sum\limits_{\substack{b\in\mathcal{B}:\\ l\in\mathcal{L}'\setminus L(b)}} \check{\beta}_{b,l} & \text{if } l \in \mathcal{L}' \setminus \bigcap_{b\in\mathcal{B}} L(b), \\ -\sum_{b\in\mathcal{B}} \check{\beta}_{b,l} + \check{\delta}_l & \text{if } l \in \mathcal{L} \setminus \mathcal{L}'. \end{cases} \tag{32}$$

The importance of this lifting is demonstrated by the following theorem:

Theorem 1. *Consider a primal-dual pair of optimal solutions (x, y, p) of the RMP and $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ of the DRMP. Then, there is a fully-compensating cost-invariant lifting feasible for the DMP if and only if the characteristic lifting is fully compensating and feasible for the DMP.*

In terms of uncompensated reduced costs this reads: If

$$\bar{c}_{b,l,m} \geq 0 \qquad \forall b \in \mathcal{B}, \; l \in L(b), \; m \in \mathcal{M} \setminus M(b,l), \tag{33}$$

$$-\sum_{\substack{b\in\mathcal{B}:\\ l\in\mathcal{L}'\setminus L(b)}} \big(\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big)^- \geq \sum_{\substack{b\in\mathcal{B}:\\ l\in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu \qquad \forall l \in \mathcal{L}' \setminus \bigcap_{b\in\mathcal{B}} L(b), \tag{34}$$

$$-\sum_{b\in\mathcal{B}} \big(\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big)^- + \big(\min_{\substack{b\in\mathcal{B}\\ m\in\mathcal{M}}} \bar{c}_{b,l,m}\big)^+ \geq -\gamma + \mu \qquad \forall l \in \mathcal{L} \setminus \mathcal{L}', \tag{35}$$

then the canonical lifting $(\bar{x}, \bar{y}, \bar{p})$ of (x, y, p) is optimal for the MP.

Moreover, if any of these inequalities is violated, then no cost-invariant lifting is fully-compensating and feasible for the DMP.

Proof. Consider first the situation in which all inequalities listed in the theorem are satisfied, which is straight-forwardly equivalent to the characteristic lifting being fully compensating and feasible for the DMP. Moreover, by cost-invariance the characteristic lifting has the same objective as (x, y, p) and, therefore, the same objective as $(\bar{x}, \bar{y}, \bar{p})$. Thus, $(\bar{x}, \bar{y}, \bar{p})$ is optimal for the MP.

Consider now any fully compensating cost-invariant lifting $(\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\delta}, \hat{\mu}, \hat{\phi}, \hat{\psi})$ of $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)$ of the DRMP that is feasible for the DMP. We will show that then all conditions of the inequalities listed in the theorem are satisfied, i.e., the characteristic lifting is fully compensating and feasible for the DMP as well. By the definition of a lifting we know that $\hat{\alpha} = \alpha$, $\hat{\gamma} = \gamma$, $\hat{\mu} = \mu$, $\hat{\phi} = \phi$, and $\hat{\psi} = \psi$. Since the lifting is fully compensating, the compensated reduced costs for all x-variables are non-negative. In particular, the first inequality in the theorem is satisfied, since in that case $\hat{c}_{b,l,m} = \bar{c}_{b,l,m}$.

Consider first the case $b \in \mathcal{B}$, $l \in \mathcal{L}' \setminus L(b)$, and $m \in \mathcal{M}$. Now $\hat{\delta}_l = \delta_l$ must hold by the lifting property. We have

$$c_{b,l,m} - \alpha_b + \hat{\beta}_{b,l} - \hat{\delta}_l - m|l|(\phi - \psi) = \bar{c}_{b,l,m} + \hat{\beta}_{b,l} \geq 0. \tag{36}$$

From this, it follows that for all $b \in \mathcal{B}$ that

$$\hat{\beta}_{b,l} \geq \max_{m \in \mathcal{M}} (-\bar{c}_{b,l,m}) = -\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}. \tag{37}$$

Since the given lifting is feasible for the DMP, this implies (using $\hat{\beta}_{b,l} \geq 0$) for all $l \in \mathcal{L}' \setminus L(b)$:

$$\begin{aligned}
0 &\geq \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu \\
&\geq \sum_{\substack{b \in \mathcal{B}: \\ b \in \mathcal{L}' \setminus L(b)}} \hat{\beta}_{b,l} + \sum_{\substack{b \in \mathcal{B}: \\ b \in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu \\
&= \sum_{\substack{b \in \mathcal{B}: \\ b \in \mathcal{L}' \setminus L(b)}} \max\{0, \hat{\beta}_{b,l}\} + \sum_{\substack{b \in \mathcal{B}: \\ b \in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu \\
&\geq \sum_{\substack{b \in \mathcal{B}: \\ b \in \mathcal{L}' \setminus L(b)}} \max\{0, -\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}\} + \sum_{\substack{b \in \mathcal{B}: \\ b \in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu \\
&\geq \sum_{\substack{b \in \mathcal{B}: \\ b \in \mathcal{L}' \setminus L(b)}} \big(\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}\big)^- + \sum_{\substack{b \in \mathcal{B}: \\ b \in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu.
\end{aligned} \tag{38}$$

This is equivalent to the second inequality in the theorem. Consider next the case $b \in \mathcal{B}$, $l \in \mathcal{L} \setminus \mathcal{L}'$, and $m \in \mathcal{M}$. We have

$$c_{b,l,m} - \alpha_b + \hat{\beta}_{b,l} - \hat{\delta}_l - m|l|(\phi - \psi) = \bar{c}_{b,l,m} + \hat{\beta}_{b,l} - \hat{\delta}_l \geq 0. \tag{39}$$

Let (b^*, m^*) be a minimizer in $\min_{b \in \mathcal{B}, m \in \mathcal{M}} \bar{c}_{b,l,m} = \bar{c}_{b^*,l,m^*}$. That is, $\bar{c}_{b^*,l,m^*} \leq \min_{m \in \mathcal{M}} \bar{c}_{b,l,m}$ for all $b \in \mathcal{B}$. We consider the cases $\bar{c}_{b^*,l,m^*} \leq 0$ and $\bar{c}_{b^*,l,m^*} > 0$. In the first case, since the lifting is fully compensating, we have $\hat{\beta}_{b,l} - \hat{\delta}_l \geq -\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}$ for all $b \in \mathcal{B}$, in particular

$$\hat{\beta}_{b^*,l} - \hat{\delta}_l \geq -c_{b^*,l,m^*} \geq 0, \tag{40}$$

$$\hat{\beta}_{b,l} \geq -\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}. \tag{41}$$

We conclude for all $l \in \mathcal{L} \setminus \mathcal{L}'$, using $\hat{\delta}_l \geq 0$, $\hat{\beta}_{b,l} \geq 0$:

$$\begin{aligned}
0 &\geq \sum_{b\in\mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu \\
&\geq \sum_{b\in\mathcal{B}\setminus\{b^*\}} \hat{\beta}_{b,l} + \big(\hat{\beta}_{b^*,l} - \hat{\delta}_l\big) - \gamma + \mu \\
&= \sum_{b\in\mathcal{B}\setminus\{b^*\}} \max\big\{0, \hat{\beta}_{b,l}\big\} + \max\{0, \hat{\beta}_{b^*,l} - \hat{\delta}_l\} - \gamma + \mu \\
&\geq \sum_{b\in\mathcal{B}\setminus\{b^*\}} \max\big\{0, -\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big\} + \max\big\{0, -\min_{m\in\mathcal{M}} \bar{c}_{b^*,l,m}\big\} - \gamma + \mu \\
&= \sum_{b\in\mathcal{B}} \max\big\{0, -\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big\} - \gamma + \mu \\
&= \sum_{b\in\mathcal{B}} \big(\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big)^- - \gamma + \mu. \qquad (42)
\end{aligned}$$

This is equivalent to the third inequality of the theorem in the case $\bar{c}_{b^*,l,m^*} \leq 0$. In the case $\bar{c}_{b^*,l,m^*} > 0$, we use that

$$0 \leq \hat{\delta}_l \leq \min_{\substack{b\in\mathcal{B}\\ m\in\mathcal{M}}} \big(\bar{c}_{b,l,m} + \hat{\beta}_{b,l}\big) = \bar{c}_{b^*,l,m^*} + \hat{\beta}_{b^*,l}, \qquad (43)$$

$$\hat{\beta}_{b,l} \geq -\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}. \qquad (44)$$

Then, again using $\hat{\beta}_{b,l} \geq 0$ and $\min_{m\in\mathcal{M}} \bar{c}_{b,l,m} \geq \bar{c}_{b^*,l,m^*} > 0$, we get:

$$\begin{aligned}
0 &\geq \sum_{b\in\mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu \\
&\geq \sum_{b\in\mathcal{B}} \hat{\beta}_{b,l} - \big(\bar{c}_{b^*,l,m^*} + \hat{\beta}_{b^*,l}\big) - \gamma + \mu \\
&= \sum_{b\in\mathcal{B}\setminus\{b^*\}} \max\big\{0, \hat{\beta}_{b,l}\big\} - \bar{c}_{b^*,l,m^*} - \gamma + \mu \\
&\geq \sum_{b\in\mathcal{B}\setminus\{b^*\}} \max\big\{0, -\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big\} - \bar{c}_{b^*,l,m^*} - \gamma + \mu \\
&= -\big(\bar{c}_{b^*,l,m^*}\big)^+ - \gamma + \mu. \qquad (45)
\end{aligned}$$

This is equivalent to the third inequality of the theorem in the case $\bar{c}_{b^*,l,m^*} > 0$. Together, these two cases imply the third inequality in the theorem. Summarized, the characteristic lifting is feasible as well, as claimed. □

Clearly, we will generate new columns with respect to the compensated reduced costs with respect to the characteristic lifting. Then, only complete sets of columns will be generated whose negative reduced costs cannot be fully compensated by *any* lifting. We call such a variable set *promising*.

The individual $x_{b,l,m}$ with $\bar{c}_{b,l,m} < 0$ for which $l \in L(b)$ and $m \in \mathcal{M} \setminus M(b,l)$ form single-element promising sets of variables because for those we have $\hat{c}_{b,l,m} = \bar{c}_{bl,m}$, i.e., no lifting can compensate their negative uncompensated reduced costs.

Another promising set of variables stems from any fixed $l \in \mathcal{L}' \setminus \bigcap_{b\in\mathcal{B}} L(b)$ for which

$$\check{c}_l = -\sum_{\substack{b\in\mathcal{B}:\\ l\in\mathcal{L}'\setminus L(b)}} \big(\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big)^- < \sum_{\substack{b\in\mathcal{B}:\\ l\in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu. \tag{46}$$

The set contains all $x_{b,l,m}$ with $\min_{m\in\mathcal{M}} \bar{c}_{b,l,m} < 0$.

And finally, we have a promising set of variables from any $l \in \mathcal{L} \setminus \mathcal{L}'$ for which

$$\check{c}_l = -\sum_{b\in\mathcal{B}} \big(\min_{m\in\mathcal{M}} \bar{c}_{b,l,m}\big)^- + \big(\min_{\substack{b\in\mathcal{B}\\ m\in\mathcal{M}}} \bar{c}_{b,l,m}\big)^+ < -\gamma + \mu. \tag{47}$$

The set contains all $x_{b,l,m}$ with $\min_{m\in\mathcal{M}} \bar{c}_{b,l,m} < 0$ plus the new variable y_l.

Note, that the resulting mechanism is very natural, since it essentially enforces a *coordinated* generation of new variables taking the quality of an assignment into account with respect to all branches at the same time.

We complete our consideration by two lower bounds implied by using essentially two non-cost-invariant fully compensating shifts of dual variables (that is no lifting, by the way) that are feasible for the DMP.

The first idea is to shift the α-variables of the DMP so as to compensate all the negative reduced costs of variables $x_{b,l,m}$ with $b \in \mathcal{B}$, $l \in L(b)$, and $m \in \mathcal{M} \setminus M(b,l)$. The result is that for all such (b,l,m) the x-constraints of the DMP are feasible. The second idea is to shift the γ-variable of the DMP so as to compensate the violation of the y-constraints, while the reduced costs of the variables $x_{b,l,m}$ for $b \in \mathcal{B}$, $l \in \mathcal{L} \setminus L(b) \cup \mathcal{L} \setminus \mathcal{L}'$, and $m \in \mathcal{M}$ are fully compensated by the characteristic lifting.

More formally, define the minimal negative reduced costs of any promising variable $x_{b,l,m}$ with $b \in \mathcal{B}$, $l \in L(b)$, and $m \in \mathcal{M} \setminus M(b,l)$ as

$$\bar{c}^*_b := -\big(\min_{\substack{l\in L(b)\\ m\in M(b,l)}} \bar{c}_{b,l,m}\big)^- \le 0. \tag{48}$$

Next, define the maximal violation of the y-constraint in the DMP by the characteristic lifting as

$$\bar{d}^* := -\big(\min_{\substack{l\in\mathcal{L}'\setminus\bigcap_{b\in\mathcal{B}} L(b)\cup\mathcal{L}\setminus\mathcal{L}'\\ m\in M(b,l)}} -\sum_{b\in\mathcal{B}} \check{\beta}_{b,l} + \check{\delta}_l + \gamma - \mu\big)^- \le 0. \tag{49}$$

Next, we define a shift of the characteristic lifting by

$$\tilde{\alpha}_b := \alpha_b + \bar{c}^*_b, \tag{50}$$
$$\tilde{\beta}_{b,l} := \check{\beta}_{b,l}, \tag{51}$$
$$\tilde{\gamma} := \gamma - \bar{d}^* \tag{52}$$
$$\tilde{\delta}_l := \check{\delta}_l, \tag{53}$$
$$\tilde{\mu} := \mu, \tag{54}$$
$$\tilde{\phi} := \phi, \tag{55}$$
$$\tilde{\psi} := \psi. \tag{56}$$

We call this the *lower-bound shift of the characteristic lifting.*

Theorem 2. *The lower-bound shift of the characteristic lifting is feasible for the DMP. Moreover, it changes the objective function value of the characteristic lifting by* $\sum_{b\in\mathcal{B}} \bar{c}_b^* + k\bar{d}^*$. *In other words, if* z^{RMP} *is the optimal value of the RMP and* z^{MP} *is the optimal value of the MP, then we have*

$$z^{MP} \geq z^{CSB} := z^{RMP} + \sum_{b\in\mathcal{B}} \bar{c}_b^* + k\bar{d}^*. \tag{57}$$

Proof. Since $\bar{d}^* \leq 0$ we have that $\tilde{\gamma}$ is non-negative. Note, that there is no non-negativity constraint for α_b. We consider the x-constraints in two steps:

1. The x-constraints with $b \in \mathcal{B}$, $l \in L(b)$, and $m \in \mathcal{M}$ read:

$$\begin{aligned} & \tilde{\alpha}_b - \tilde{\beta}_{b,l} + \tilde{\delta}_l + m|l|(\tilde{\phi} - \tilde{\psi}) \leq c_{b,l,m} \\ \iff & \alpha_b + \bar{c}_b^* - \beta_{b,l} + \delta_l + m|l|(\phi - \psi) \leq c_{b,l,m} && (58) \\ \iff & \bar{c}_{b,l,m} = c_{b,l,m} - \alpha_b + \beta_{b,l} - \delta_l - m|l|(\phi - \psi) \geq \bar{c}_b^*, && (59) \end{aligned}$$

   which holds by the minimality of $\bar{c}_b^*$.

2. The x-constraints for all (b, l, m) with $l \in \mathcal{L}' \setminus L(b) \cup \mathcal{L} \setminus \mathcal{L}'$ are satisfied since the characteristic lifting is fully compensating, and its lower-bound shift relaxes the x-constraints.

Next we consider the y-constraints.

1. The y-constraints with $l \in \bigcap_{b\in\mathcal{B}} L(b)$ are satisfied by the characteristic lifting, since they are identical to those in the DRMP, and there they are satisfied by the lifting property. The shift of γ relaxes the y-constraints, so that also the lower-bound shift satisfies these y-constraints.

2. The y-constraints with $l \in \mathcal{L}' \setminus \bigcap_{b\in\mathcal{B}} L(b) \cup \mathcal{L} \setminus \mathcal{L}'$ read:

$$\begin{aligned} & \sum_{b\in\mathcal{B}} \tilde{\beta}_{b,l} - \tilde{\gamma} - \tilde{\delta}_l + \tilde{\mu} \leq 0, && (60) \\ \iff & \sum_{b\in\mathcal{B}} \check{\beta}_{b,l} - (\gamma - \bar{d}^*) - \check{\delta}_l + \mu \leq 0, && (61) \\ \iff & -\sum_{b\in\mathcal{B}} \check{\beta}_{b,l} - \gamma - \check{\delta}_l + \mu \geq \bar{d}^*, && (62) \end{aligned}$$

   which holds by the minimality of $\bar{d}^*$.

Thus, the lower-bound shift of the characteristic lifting is feasible for the DMP. Plugging the lower-bound shift into the objective function of the DMP yields the lower-bound formula. □

We will refer to this lower bound by the name *characteristic shift bound*. There may be tighter bounds by other shifts. The advantage of the characteristic shift bound is that is is readily available in any algorithm that computes the promising sets of variables.

5 A branch-and-price algorithm for the SLDP

Since the set of applicable lot-types and, thus, the set of binary variables in the stated ILP formulation may become too large for a static ILP solution, a natural approach is to consider

applicable lot-types and the corresponding assignment options dynamically in a branch-and-price algorithm.

In this section we show how special structure can be used to obtain a fast branch-and-price algorithm for practically relevant instances: We typically have $300 \leq |\mathcal{B}| \leq 1600$ and $3 \leq |\mathcal{M}| \leq 7$ while $|\mathcal{L}|$ can be around 10^9, see the example stated in the introduction.

5.1 The top-level algorithm ASG

The idea of our exact branch-and-price algorithm is based on the following practical observations on real-world data:

- The integrality gap of our SLDP model is small.
- Solutions generated by heuristics perform very well (see [12] for SFA).
- An MILP solver can solve instances of the ILP model with few lot-types very quickly.
- There seems to be a "small" set of "good" and a "large" set of "bad" lot-types.
- No mathematical structure of the set of "good" lot-types is known a-priori.

We want to exploit the fast heuristic and the tight ILP model. Instead of using only the LP-solver part of an MILP solver and branch to integrality manually in our own branch-and-price tree, we decided to utilize as much of the modern ILP solver technology as possible in our algorithm by generating and solving a sequence of ILP models handling distinct subproblems with only few lot-types. We call this branch-and-price method *augmented subproblem grinding (ASG)*. The name is motivated by the interpretation that in an iteration we "grind-off" a promising subproblem from the full remaining subproblem, solve it to optimality, and proceed with the remaining subproblem that does not contain the solved promising subproblem anymore. The implementation of this method needs a much smaller effort than a typical branch-and-price implementation with branching, e.g., on the variables of a compact model formulation like the one in appendix A.

With this, the outline of ASG in its simplest version reads as follows.

1. Run the SFA heuristics to obtain an incumbent feasible SLDP solution (x^*, y^*, p^*) that implies an upper bound z^{upper} for the SLDP optimum.
2. Generate a small set of initial columns for the RMP so that the RMP is feasible. Solve the RMP. Enter the pricing phase.
3. The pricing phase: While the PP returns new columns with negative reduced costs, do the following:
   (a) Add (some of) these columns to the RMP and resolve.
   (b) From the PP update z^{lower} using the characteristic shift bound.
   (c) If $z^{lower} \geq z^{upper}$, return (x^*, y^*, p^*) and their cost z^{upper}.

   At the end of this step, we obtain an optimal MP solution which implies an updated lower bound z^{lower} for the SLDP optimum. If $z^{lower} \geq z^{upper}$, return (x^*, y^*, p^*) and their cost z^{upper}. Else enter the cutting phase.

4. The cutting phase: Add all lot-types $\mathcal{L}' \subset \mathcal{L}$ from the RMP to the RSLDP and solve the RSLDP. This is the *promising-subproblem branch* of ASG, which is solved immediately.[2] If the RSLDP solution is cheaper than the current incumbent (x^*, y^*, p^*), update the incumbent (x^*, y^*, p^*) and the upper bound z^{upper}. If $z^{lower} \geq z^{upper}$, return (x^*, y^*, p^*) and their cost z^{upper}. Otherwise, strengthen the set covering constraint in the SLDP to $\sum_{l \in \mathcal{L} \setminus \mathcal{L}'} y_l + p \geq 1$ yielding $p \geq 1$ in the RMP, solve the resulting RMP, and enter the pricing phase. This is the *remaining-subproblem branch* of ASG.

The top-level algorithm for ASG is listed in pseudo code in Algorithm 1. Several subroutines are used that will be explained in the following sections.

Input: A consistent SLDP; $K_1, K_2, K_3, K_4, K_5 \in \mathbb{N}$ (Parameters)
Output: An optimal solution for the SLDP

$z^{upper} \longleftarrow \infty$;
$z^{lower} \longleftarrow -\infty$;
$(\mathcal{L}', \mathsf{ScoreTable}) \longleftarrow$ GenerateLocBestLottypes(K_1, K_2);
$\mathcal{L}'' \longleftarrow \varnothing$;
$\big((\bar{x}^*, \bar{y}^*, \bar{p}^*), z^{upper}\big) \longleftarrow$ ScoreFixAdjust$^+(\mathsf{ScoreTable})$;
$(\mathcal{L}', L, M) \longleftarrow$ InitRMP(K_1, K_2, K_3);
while *true* **do**
  repeat
    `// The pricing phase:`
    $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}) \longleftarrow$ SolveRMP$((\mathcal{L}', L, M); \mathcal{L}'')$;
    $\big((\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}), z^{lower}\big) \longleftarrow$ SolvePP$(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; z^{lower}; K_4, K_5)$;
    if $z^{upper} \leq z^{lower}$ **then**
      return $(\bar{x}^*, \bar{y}^*, \bar{p}^*)$;
    else
      $(\mathcal{L}', L, M) \longleftarrow (\mathcal{L}' \cup \mathcal{L}^{\text{new}}, L \cup L^{\text{new}}, M \cup M^{\text{new}})$;
  until $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) = (\varnothing, \varnothing, \varnothing)$;
  `// The cutting phase:`
  $\big((x^*, y^*, p^*), z^{upper}\big) \longleftarrow$ SolveRSLDP$(\mathcal{L}')$;
  if $z^{upper} \leq z^{lower}$ **then**
    return $(\bar{x}^*, \bar{y}^*, \bar{p}^*)$
  else
    $\mathcal{L}'' \longleftarrow \mathcal{L}'$;

Algorithm 1: Top Level Algorithm ASG for an SLDP

5.2 The subroutine GenerateLocBestLottypes(K, K')

We call an element $(b, l, m^*) \in \mathcal{B} \times \mathcal{L} \times \mathcal{M}$ a *locally optimal multiplicity assignment*, if $c_{b,l,m^*} = \min_{m \in \mathcal{M}} c_{b,l,m}$. (Note that m^* depends on b and l.) Moreover, we call an element $(b, l^*, m^*) \in \mathcal{B} \times \mathcal{L} \times \mathcal{M}$ a *locally optimal lot-type-multiplicity assignment*, if $c_{b,l^*,m^*} = \min_{l \in \mathcal{L}} c_{b,l,m^*}$. For a branch $b \in \mathcal{B}$ and a number $K \in \mathbb{N}$ we call a list of lot-types $\big(l_1(b), \ldots, l_K(b)\big)$ a *list of K locally best lot-types for branch b*, if for all $j = 1, \ldots, K$ there are at most $j - 1$ many $l \in \mathcal{L}$ with $c_{b,l,m^*} < c_{b,l_j,m^*}$.

For parameters $K, K' \in \mathbb{N}$, the subroutine GenerateLocBestLottypes(K, K') generates a subset of lot-types in the following way: first, for each branch $b \in \mathcal{B}$ a list $\big(l_1(b), \ldots, l_K(b)\big)$ of K locally best lot-types is determined. Then, we score the lot-types by adding a score of $-\big(10^{K-j}\big)$

[2] In various applications with tight models, the procedure is stopped at this point because one hopes that the integrality gap is so small that further effort need not be spent (see, e.g., [18]). ASG can be seen as extending this into an exact algorithm finding the optimum ILP solution.

to the jth best lot-type for branch b, $j = 1, \ldots, K$. From the score ranking we pick the K' elements with the smallest scores. The pseudo code can be found in Algorithm 2.

Input: A consistent SLDP and $K, K' \in \mathbb{N}$
Output: A subset of lot-types $\mathcal{L}' \subseteq \mathcal{L}$ and a sorted table ScoreTable of all lot-types with non-zero score
`// store lot-types (keys) with scores (values) in a table, initially empty:`
ScoreTable $\longleftarrow$ ();
for $b \in \mathcal{B}$ **do**
  $(l_1(b), \ldots, l_K(b)) \longleftarrow$ FindLocBestLottypes(b, K);
  for j *from* 1 *to* K **do**
    if ScoreTable *does not yet contain* l_j **then**
      insert l_j into ScoreTable with value 0;
    ScoreTable$(l_j) \longleftarrow$ ScoreTable$(l_j) - 10^{K-j}$
$\mathcal{L}' \longleftarrow$ { the best K' lot-types in ScoreTable };
return $(\mathcal{L}',$ ScoreTable$)$;

Algorithm 2: GenerateLocBestLottypes

Input: A consistent SLDP, $b \in \mathcal{B}$, $K \in \mathbb{N}$
Output: A list LotTypeList $= (l_1, \ldots, l_K)$ of K locally best lot-types for branch b
`// start with an empty sorted list:`
LotTypeList $\longleftarrow$ ();
`// start with an empty partial lot-type:`
$l' \longleftarrow$ ();
`// applicable range of cardinalities for the first size:`
$n_{\min} \longleftarrow \min_t -(|S| - 1) \max_c$;
$n_{\max} \longleftarrow \max_t -(|S| - 1) \min_c$;
RecLocBestLottypes$(b, K, l', n_{\min}, n_{\max};$ LotTypeList$)$;
return LotTypeList;

Algorithm 3: FindLocBestLottypes

Since determining the K locally best lot-types for a branch requires a search in the complete, possibly very large lot-type set $\mathcal{L}$, we rather synthesize lot-types by determining the number of pieces for each size separately in a branch-and-bound algorithm named FindLocBestLottypes(b, K) that recursively extends partial lot-types in a depth-first-search manner. In each branch-and-bound node, the cost for the completion of a partial lot-type is bounded from below by the minimal cost over all possible multiplicities of the cheapest (not necessarily feasible) completions. Each leaf in the branch-and-bound tree yields an applicable lot-type and, together with its locally optimal multiplicity assignment, a cost. We maintain a sorted list of the K best lot-types. As long as the list contains less than K elements, each new (complete) applicable lot-type is added to it. If the list is full, the cost of each new lot-type is compared to the worst in the list. If it is cheaper, then it is exchanged with the current worst solution, and the list is resorted. Moreover, once the list is full, the lower bound of each partial lot-type is compared to the worst in the list. If the lower bound exceeds the cost of the worst element in the list, then the node corresponding to the partial lot-type is pruned. In the following, we explain the procedure more formally.

A *partial lot-type* is a vector $(l'_{s'})_{s' \in \mathcal{S}'} \in \mathbb{N}^{\mathcal{S}'}$ for some $\mathcal{S}' \subseteq \mathcal{S}$. As for complete lot-types, the number of pieces in this partial lot-type is denoted by $|l'| := \sum_{s' \in \mathcal{S}'} l'_{s'}$. A *completion* of a partial lot-type l' is a lot-type l with $l_{s'} = l'_{s'}$ for all $s' \in \mathcal{S}'$. For any partial lot-type l' we consider the two extremal completions $\underline{l'}$ with $(\underline{l'})_{s \in \mathcal{S} \setminus \mathcal{S}'} := (\min_c)_{s \in \mathcal{S} \setminus \mathcal{S}'}$ and $\overline{l'}$ with $(\overline{l'})_{s \in \mathcal{S} \setminus \mathcal{S}'} := (\max_c)_{s \in \mathcal{S} \setminus \mathcal{S}'}$, where we fix the numbers of pieces for missing sizes to the minimal and maximal applicable values, respectively. Consequently, the minimal and maximal total numbers of pieces for the missing

Input: A consistent SLDP, $b \in \mathcal{B}$, $K \in \mathbb{N}$, a partial lot-type l', $n_{\min}, n_{\max} \in \mathbb{N}$, a list $\mathsf{LotTypeList} = (l_1, \ldots, l_K)$ of lot-types found so far
Output: An updated list $\mathsf{LotTypeList}$ of lot-types

if l' *is complete* **then**
  `// lot-type reached - check cost and update lot-type list:`
  if $|\mathsf{LotTypeList}| < K$ *or* $c_{b,l',m^*} < c_{b,l_K,m^*}$ **then**
    insert l' into $\mathsf{LotTypeList}$;
else
  `// list extensions of` l' `sorted by lower bounds:`
  $\mathsf{CardList} \longleftarrow ()$;
  for n *from* $n_{\min}$ *to* $n_{\max}$ **do**
    $l'' \longleftarrow (l', n)$;
    if $\lambda_b(l'') \le c_{b,l_K,m^*}$ **then**
      `// lower bound does not exclude` l''`:`
      insert n into $\mathsf{CardList}$;
  sort $\mathsf{CardList}$ by increasing $\lambda_b(l'')$;
  for n *in* $\mathsf{CardList}$ **do**
    `// process all non-pruned extensions:`
    $l'' \longleftarrow (l', n)$;
    $n'_{\min} \longleftarrow n_{\min} + (\max_c - n)$;
    $n'_{\max} \longleftarrow n_{\max} - (n - \min_c)$;
    RecLocBestLottypes$(b, K, l'', n'_{\min}, n'_{\max}; \mathsf{LotTypeList})$;

Algorithm 4: RecLocBestLottypes

sizes are given by $|\underline{l'}|$ and $|\overline{l'}|$, respectively.

A partial lot-type l' is *applicable* if $\min_c \le l'_{s'} \le \max_c$ for all $s' \in \mathcal{S}'$, and, moreover, $|\overline{l'}| \ge \min_t$, and $|\underline{l'}| \le \max_t$, i.e., there exists a completion to an applicable lot-type. For a given branch b, the *partial cost* of a partial lot-type l' if delivered with multiplicity m is given by $c_{b,l',m} := \sum_{s' \in \mathcal{S}'} |d_{b,s'} - m l'_{s'}|$. A lower bound for the cost $c_{b,l,m}$ of any completion l of l' can be derived as follows: Denote by

$$c_{b,l^*_s,m} := \min\big\{|d_{b,s} - m l_s| : \min_c \le l_s \le \max_c\big\} \tag{63}$$

the minimal partial cost over all single-size partial lot-types l_s for a given branch b and a given multiplicity m. Let l^*_s be a corresponding minimizer. In other words, if branch b receives m lots of some lot-type, then the lot-type component l^*_s for size s incurs the smallest possible cost contribution in size s. With this locally optimal fixing, we obtain a lower bound for any completion l of l' if the multiplicity is m:

$$c_{b,l,m} \ge \lambda_b(l', m) := c_{b,l',m} + \sum_{s \in \mathcal{S} \setminus \mathcal{S}'} c_{b,l^*_s,m}. \tag{64}$$

A lower bound for the cost in branch b of any completion l of l' with any multiplicity is then

$$c_{b,l,m} \ge \lambda_b(l') := \min_{m \in \mathcal{M}} \lambda_b(l', m). \tag{65}$$

Unfortunately, the function $\lambda_b(l', m)$ is not convex in m. Still, the optimization over $m \in \mathcal{M}$ can be done by complete enumeration, since typically $|M| \le 7$.

The depth-first-search branch-and-bound follows in each recursion level the extensions of a partial lot-type by a single new size in the order of increasing lower bounds for the cost of completion. A possible pseudo code is shown in Algorithms 3 and 4.

5.3 The subroutine ScoreFixAdjust$^+$(ScoreTable)

We briefly recall the SFA heuristics from [12], adapted to our context. The name stems from the three main steps *score – fix – adjust*. In the *score* step, all lot-types are scored by how-often they are the locally best, second-best, ..., Kth-best lot-types for a branch. In our set-up, the scoring information is taken from the output ScoreTable of GenerateLocBestLottypes in Algorithm 2. The order of ScoreTable implies a lexicographic order on all k-subsets of lot-types in ScoreTable. In the *fix* step, all k-subsets of lot-types in ScoreTable are traversed in this lexicographic order. To each branch, the best fitting lot-type from this k-subset is assigned in the locally optimal multiplicity. If the lower and upper bounds on the total cardinality of the supply over all branches are satisfied, this yields a feasible solution. If not, the *adjust* step is necessary: For each possible exchange of a multiplicity assignment that makes the total-cardinality violation smaller (no overshooting allowed), the relative cost-increase per cardinality change is computed. Then, the assignment from the *fix* step is adjusted by applying these exchange sequentially in the order of increasing relative cost increases until either the total cardinality is feasible or no feasible exchange is possible. In the first case, a feasible assignment is found and the cost is compared to the current best incumbent. In the second case, the k-subset under consideration is dismissed. This procedure is run until either all k-subsets of lot-types in ScoreTable have been processed or a time limit has been reached. The best found solution is returned. In this paper, we extend the *adjust* step by exchanges of lot-type-multiplicity assignments. This has the advantage that we can find feasible solutions in more instances. An overview in pseudo code can be found in Algorithm 5.

```
   Input: A consistent SLDP, a sorted table ScoreTable of lot-types with non-zero score
   Output: A feasible solution and its objective value ((x*, y*, p*), z^upper) or (−, ∞)
 1 ((x*, y*, p*), z^upper) ⟵ (−, ∞);
 2 if k > 1 then
 3 |  ((x*, y*, p*), z^upper) ⟵ AverageLottypeHeuristics();
 4 repeat
 5 |  for L in k-subsets of lot-types in ScoreTable do
 6 |  |  for b ∈ B do
 7 |  |  |  (b, l_L(b), m_L(b)) ⟵ (b, l*, m*) ;          // locally optimal assignment
 8 |  |  while (b, l_L(b), m_L(b)) violates cardinality constraint do
 9 |  |  |  AdjTable ⟵ {(b, l, m) : (b, l, m) decreases violation of (b, l_L(n), m_L(b))};
10 |  |  |  if AdjTable = ∅ then
11 |  |  |  |  z(L) ⟵ ∞, break;
12 |  |  |  else
13 |  |  |  |  (b*, l*, m*) ⟵ the marginally cheapest element in AdjTable;
14 |  |  |  |  (b*, l_L(b*), m_L(b*)) ⟵ (b*, l*, m*);
15 |  |  z(L) ⟵ cost of assignment (b, l_L(b), m_L(b))_{b∈B};
16 |  |  if z(L) < z^upper then
17 |  |  |  z^upper ⟵ z(L), (x̄*, ȳ*, p̄*) ⟵ variable encoding of (l_L(b), m_L(b))_{b∈B};
18 until time limit reached;
19 return ((x̄*, ȳ*, p̄*), z^upper);
```

Algorithm 5: The subroutine ScoreFixAdjust$^+$.

In order to find a feasible solution for all consistent instances we employ a simple fall-back heuristic called *average lot-type heuristics (ALH)* that guarantees feasibility in all consistent instances with $k > 1$. This is no serious restriction because for $k = 1$ SFA was proven to be fast and optimal in [12]. We consider ALH as part of SFA that is run at the very beginning of SFA.

ALH works as follows: Compute the average demand $\bar{d}_t := \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} d_b$ per branch over all

Input: A consistent SLDP with $k > 1$
Output: A feasible solution and its objective value $\big((x^*, y^*, p^*), z^{upper}\big)$

$\bar{d}_t \longleftarrow \max\{\min_t, \min\{\max_t, \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}}\sum_{s\in\mathcal{S}} d_{b,s}\}\}$;
$\bar{d}_c \longleftarrow \max\{\min_c, \min\{\max_c, \frac{1}{|\mathcal{S}|}\bar{d}_t\}\}$;
$\check{l} \longleftarrow \big(\lfloor \bar{d}_c \rfloor\big)_{s\in\mathcal{S}}$, $\hat{l} \longleftarrow \big(\lceil \bar{d}_c \rceil\big)_{s\in\mathcal{S}}$;
while $|\check{l}| < \min_t$ **do**
  $\check{l}_s \longleftarrow \check{l}_s + 1$ for some $s \in \mathcal{S}$ with $\check{l}_s < \max_c$;
while $|\hat{l}| > \max_t$ **do**
  $\hat{l}_s \longleftarrow \hat{l}_s - 1$ for some $s \in \mathcal{S}$ with $\hat{l}_s > \max_c$;
$\bar{n} \longleftarrow 0$, $D \longleftarrow 0$, $m(b) \longleftarrow 1$;
for $b \in \mathcal{B}$ **do**
  if $\bar{n}|\check{l}| + (|\mathcal{B}| - b - 1)|\hat{l}| < \underline{I}$ **then**
    $l(b) \longleftarrow \hat{l}$;
  else if $\bar{n}|\check{l}| + (|\mathcal{B}| - b - 1)|\check{l}| > \overline{I}$ **then**
    $l(b) \longleftarrow \check{l}$;
  else if $\bar{n} + \frac{|\check{l}|+|\hat{l}|}{2} > D + d_b$ **then**
    $l(b) \longleftarrow \check{l}$;
  else
    $l(b) \longleftarrow \hat{l}$;
  $\bar{n} \longleftarrow \bar{n} + |l(b)|$, $D \longleftarrow D + d_b$;
$(\bar{x}^*, \bar{y}^*, \bar{p}^*) \longleftarrow$ variable encoding of $\big(l(b), m(b)\big)_{b\in\mathcal{B}}$, $z^{upper} \longleftarrow$ cost of (x^*, y^*, p^*);
return $\big((\bar{x}^*, \bar{y}^*, \bar{p}^*), z^{upper}\big)$;

Algorithm 6: The subroutine AverageLottypeHeuristics.

branches with $d_b := \sum_{s\in\mathcal{S}} d_{b,s}$. If this number is smaller than $\min_t$, set it to $\min_t$. If it is larger than $\max_t$, set it to $\max_t$. Both cannot happen since then $\min_t > \max_t$, and the instance is infeasible. Next, compute the average demand per size $\bar{d}_c := \frac{1}{|\mathcal{S}|}\bar{d}_t$. If this number is smaller than $\min_c$ set it to $\min_c$. If it is larger than $\max_c$ set it to $\max_c$. Again, both cannot happen for a feasible instance.

For each $s \in \mathcal{S}$ let

$$\check{l}_s := \max\big\{\lfloor \bar{d}_c \rfloor, \min_c\big\} \text{ and} \tag{66}$$

$$\hat{l}_s := \min\big\{\lceil \bar{d}_c \rceil, \max_c\big\}. \tag{67}$$

While $|\check{l}| < \min_t$, increase an arbitrary component $\check{l}_s < \max_c$ by one. (If no such component exists, the instance is infeasible.) Analogously, we adjust $\hat{l}$.

This defines two applicable lot-types $\check{l}$ and $\hat{l}$. If $|\mathcal{B}|\cdot|\check{l}| > \overline{I}$, then, by construction, $|\mathcal{B}|\cdot|\mathcal{S}|\bar{d}_c > \overline{I}$, or $|\mathcal{B}|\cdot|\mathcal{S}|\min_c > \overline{I}$, or $|\mathcal{B}|\bar{d}_t > \overline{I}$, or $|\mathcal{B}|\min_t > \overline{I}$. All cases prove that the instance is not consistent. Similarly, $|\mathcal{B}| \cdot |\hat{l}| < \underline{I}$ is impossible for consistent instances.

Now assign $\check{l}$ and $\hat{l}$ sequentially to branches. This is possible for all proper instances ($k > 1$). Assume, after the assignment to $b-1$ branches we have assigned n_{b-1} pieces and the aggregated demand over the first $b-1$ branches is D_{b-1}. If $n_{b-1} + |\check{l}| + (|\mathcal{B}| - b - 1)|\hat{l}| < \underline{I}$, we have to assign $\hat{l}$ to branch b. If $n_{b-1} + |\hat{l}| + (|\mathcal{B}| - b - 1)|\check{l}| > \overline{I}$, we have to assign $\check{l}$ to branch b. Otherwise, assign to branch b the lot-type $\check{l}$ if and only if $n_{b-1} + \frac{|\check{l}|+|\hat{l}|}{2} > D_{b-1} + d_b$, i.e., try to approximate the demand aggregated up to branch b as closely as possible by the number of assigned pieces aggregated up to branch b.

Whenever $\overline{I} - \underline{I} \geq |\hat{l}| - |\check{l}|$ this sequential algorithm yields a feasible solution. For example, if $\overline{I} - \underline{I} \geq \max_t - \min_t$ this is satisfied because, by construction, $\max_t - \min_t \geq |\hat{l}| - |\check{l}|$. The

pseudo code for ALH can be found in Algorithm 6. In all of our test instances, SFA found a feasible solution within the time limit so that ALH was not really needed.

5.4 The subroutine InitRMP(K, K', K'')

In this subroutine, we generate an initial set of columns for the RMP. Our choice is to add

1. the x-columns corresponding to the SFA solution and the implied y-columns for all lot-types used in the SFA solution;
2. the y-columns corresponding to the K highest-scored lot-types, and for each of these K lot-types the x-columns for all branches with the locally optimal multiplicities;
3. for each branch b, the x-columns corresponding to all lot-types l that are among the locally best K' lot-types for b and the corresponding locally optimal lot-type-multiplicities, together with the implied y-columns.
4. If for the same lot-type and branch, there are multiple multiplicities in the initial RMP, we add the intermediate multiplicities as well; the resulting interval of multiplicities is then enlarged by K'' multiplicities to the left and the right up to the boundaries of $\mathcal{M}$.

The reason for the first set of columns is that we want the RSLDP to find a solution at least as good as the heuristic. The reason for the second set of columns is to consider some lot-types for *all* branches, namely those that fit well a large number of branches; this should ease the satisfaction of the cardinality constraint on the number of used lot-types. The reason for the third set of columns is that the individually best lot-types for branches are promising for a "spot-repair" of almost complete assignments. The reason for adding the intermediate multiplicities is that this makes some (though not all) of the subproblems convex. The computation times in our test instances (see section 6) did not react too sensitive on the deliberate choices we made here. We tested smaller and larger sets of columns with no consistent improvement in our tests.

5.5 The subroutine SolveRMP$((\mathcal{L}', L, M); \mathcal{L}'')$

In this subroutine, we solve the current RMP containing y-variables corresponding to a subset of lot-types $\mathcal{L}'$ and x-variables for assignment options corresponding to $(\mathcal{L}', L, M)$. The subset of lot-types $\mathcal{L}''$ was considered in the previous RSLDP, i.e., it is missing in the support of the set-covering constraint of the current RMP.

The only important choice to make in this procedure is which LP algorithm we call in a black-box LP solver. We chose to employ, as usual,

1. the primal simplex algorithm in the pricing phase (because adding columns maintains the primal feasibility of the previous optimal basis);
2. the dual simplex algorithm in the cutting phase (because the strengthening of the set-covering constraint maintains the dual feasibility of the previous optimal basis).

The outcome of this subroutine is a set of dual variables together with an objective value $(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP})$ that is optimal for the current DRMP. Note, that the objective function value z^{RMP} of the RMP and the DRMP is, in general, neither an upper nor a lower bound for the optimal SLDP value.

5.6 The subroutine SolvePP$(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; z^{lower}; K, K')$

In this subroutine, the pricing problem (PP) is solved based on the optimal dual variables and the objective value from the previous solving of the RMP. We follow the three types of sets of promising variables from section 4.

Input: Duals/objective $\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}$ optimal for the DRMP with lot-types $\mathcal{L}'$, current lower bound z^{lower}; $K, K' \in \mathbb{N}$
Output: New variables and a new lower bound $\big((\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}), z^{lower}\big)$

$(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \longleftarrow \varnothing$;
for $b \in \mathcal{B}$ **do**
  for $l \in L(b)$ **do**
    if $\min_{m \in \mathcal{M}} \bar{c}_{b,l,m} < 0$ **then**
      `// let` m^* `be a minimizer:`
      $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \longleftarrow (b, l, m^*)$;
if $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \neq \varnothing$ **then**
  return $\big((\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}), z^{lower}\big)$;
for $l \in \mathcal{L}' \setminus \bigcap_{b \in \mathcal{B}} L(b)$ **do**
  if $\check{c}_l < \sum_{\substack{b \in \mathcal{B}: \\ l \in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu$ **then**
    `// let` m^* `be a minimizer for` b `in` $\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}$`:`
    if $|\mathcal{L}^{new}| < K$ *or* $\check{c}_l < \check{c}_{l_K}$ **then**
      $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \longleftarrow (\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \cup \Big\{\big(b, l, m^*\big) : b \in \mathcal{B}, l \in \mathcal{L}' \setminus L(b)\Big\}$;
if $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \neq \varnothing$ **then**
  return $\big((\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}), z^{lower}\big)$;
$(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \longleftarrow (\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \cup$ FindPromLottypes$(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; K')$;
if $z^{CSB} > z^{lower}$ **then**
  $z^{lower} \longleftarrow z^{CSB}$;
return $\big((\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}), z^{lower}\big)$;

Algorithm 7: SolvePP

Input: Duals/objective $\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}$ optimal for the DRMP with lot-types $\mathcal{L}'$, $K \in \mathbb{N}$
Output: New columns $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}})$ for the K most promising new lot-types
`// start with an empty sorted list:`
$(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \longleftarrow ()$;
`// start with an empty partial lot-type:`
$l' \longleftarrow ()$;
`// applicable range of cardinalities for the first size:`
$n_{\min} \longleftarrow \min_t - (|S| - 1) \max_c$;
$n_{\max} \longleftarrow \max_t - (|S| - 1) \min_c$;
RecPromLottypes$(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; K; l', n_{\min}, n_{\max}; (\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}))$;
return $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}})$;

Algorithm 8: FindPromLottypes

First, we check whether there are promising $x_{b,l,m}$ with $\bar{c}_{b,l,m} < 0$ for which $l \in L(b)$ and $m \in \mathcal{M} \setminus M(b, l)$. We add all such variables to the RMP and store for each branch the minimal reduced cost observed.

Next, we check for all $l \in \mathcal{L}' \setminus \bigcap_{b \in \mathcal{B}} L(b)$ whether

$$\check{c}_l = - \sum_{\substack{b \in \mathcal{B}: \\ l \in \mathcal{L}' \setminus L(b)}} \big(\min_{m \in \mathcal{M}} \bar{c}_{b,l,m}\big)^- < \sum_{\substack{b \in \mathcal{B}: \\ l \in L(b)}} \beta_{b,l} - \gamma - \delta_l + \mu. \tag{68}$$

Input: $\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}$ optimal for the DMP with lot-types $\mathcal{L}'$, $K \in \mathbb{N}$, a partial lot-type l', $n_{\min}, n_{\max} \in \mathbb{N}$, a set $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}})$ of columns corresponding to lot-types found so far with $\mathcal{L}' = (l_1, \dots, l_K)$ sorted by sum of negative reduced costs

Output: An updated set $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}})$ of columns

if l' *is complete* **then**
  `// lot-type reached - check cost and update lot-type list:`
  if $l' \in \mathcal{L} \setminus \mathcal{L}'$ **then**
    if $|\mathcal{L}^{new}| < K$ *or* $\check{c}_{l'} < \check{c}_{l_K}$ **then**
      $(\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \longleftarrow (\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}) \cup \{(b, l', m^*) : b \in \mathcal{B} : \bar{c}_{b,l,m^*} < 0\}$;
else
  `// list extensions of` l' `sorted by lower bounds:`
  CardList $\longleftarrow$ ();
  for n *from* $n_{\min}$ *to* $n_{\max}$ **do**
    $l'' \longleftarrow (l', n)$;
    if $\bar{\lambda}(l'') < -\gamma + \mu$ *and* $\bar{\lambda}(l'') < \check{c}_{l_K}$ **then**
      `// lower bound does not exclude` l''`:`
      insert n into CardList;
  sort CardList by increasing $\bar{\lambda}(l'')$;
  for n *in* CardList **do**
    `// process all non-pruned extensions:`
    $l'' \longleftarrow (l', n)$;
    $n'_{\min} \longleftarrow n_{\min} + (\max_c - n)$;
    $n'_{\max} \longleftarrow n_{\max} - (n - \min_c)$;
    RecPromLottypes$(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; K; l', n_{\min}, n_{\max}; (\mathcal{L}^{\text{new}}, L^{\text{new}}, M^{\text{new}}))$;

Algorithm 9: RecPromLottypes

For any l for which this happens we add all $x_{b,l,m}$ with $\min_{m \in \mathcal{M}} \bar{c}_{b,l,m} < 0$ to the RMP. We support a parameter $K \in \mathbb{N}$ to restrict the output to the K column sets that have the smallest $\check{c}_l$.

For the final case, if there are many lot-types, we cannot simply check for all $l \in \mathcal{L} \setminus \mathcal{L}'$ whether

$$\check{c}_l = -\sum_{b \in \mathcal{B}} \big(\min_{m \in \mathcal{M}} \bar{c}_{b,l,m} \big)^- + \big(\min_{\substack{b \in \mathcal{B} \\ m \in \mathcal{M}}} \bar{c}_{b,l,m} \big)^+ < -\gamma + \mu. \tag{69}$$

Therefore, for some $K' \in \mathbb{N}$ we solve the optimization problem to find the (at most) K' lot-types $l \in \mathcal{L} \setminus \mathcal{L}'$ with minimal $\check{c}_l$ among those satisfying (69). Then, we add all corresponding y_l and all $x_{b,l,m}$ with $\min_{m \in \mathcal{M}} \bar{c}_{b,l,m} < 0$ to the RMP.

To solve the optimization problem, we devise a branch-and-bound algorithm extending partial lot-types that is very similar to the generation of the K locally best lot-types in Algorithm 2. The only difference is that instead of optimizing the cost for a single branch, we optimize the sum of negative reduced costs over all branches. This requires a little care for the lower bound.

For a given branch b and a partial lot-type l' with sizes in $\mathcal{S}'$, we use the lower bound $\lambda_b(l', m)$ from subsection 2 for the primal cost of any completion of l' to a complete lot-type when delivered to branch b in multiplicity m. In order to turn this into a bound for the reduced cost of any completion, we subtract the duals from the canonical lifting. This yields:

$$\begin{aligned} \bar{c}_{b,l',m} &\geq \bar{\lambda}_b(l', m) \\ &:= \lambda_b(l', m) - \alpha_b - m \max_{l_{\mathcal{S}'} = l'} |l| (\phi - \psi)^+ + m \min_{l_{\mathcal{S}'} = l'} |l| (\phi - \psi)^-. \end{aligned} \tag{70}$$

Together with

$$\max_{l_{\mathcal{S}'} = l'} |l| = |l'| + \min \Big\{ \big(|\mathcal{S}| - |\mathcal{S}'| \big) \max_c, \max_t - |l'| \Big\}, \tag{71}$$

$$\min_{l_{\mathcal{S}'}=l'} |l| = |l'| + \max\Big\{ \big(|\mathcal{S}| - |\mathcal{S}'|\big) \min_c, \min_t -|l'| \Big\}, \tag{72}$$

we receive a lower bound for the reduced cost for a given branch and a given multiplicity for any completion l of l'. Now we minimize over all multiplicities for each branch separately and take the sum. Thus, for each completion l of l' we have, using the short-hand $m^* := m^*(b,l)$ for minimizing multiplicities:

$$\check{c}_l = \sum_{b\in\mathcal{B}} -\big(\bar{c}_{b,l,m^*}\big)^- + \big(\min_{b\in\mathcal{B}} \bar{c}_{b,l,m^*}\big)^+ \tag{73}$$

$$\geq \sum_{b\in\mathcal{B}} -\big(\bar{\lambda}_b\big(l',m^*\big)\big)^- + \min_{b\in\mathcal{B}}\big(\bar{\lambda}_b\big(l',m^*\big)\big)^+ \tag{74}$$

$$:= \bar{\lambda}(l'). \tag{75}$$

Again, the function $\bar{\lambda}_b(l',m)$ is not convex in m, and we implement the optimization over $m \in \mathcal{M}$ by complete enumeration (recall that $|M| \leq 7$ typically).

From all the newly generated columns, we compute the characteristic shift bound z^{CSB} using Theorem 2 and update the lower bound z^{lower} if $z^{CSB} > z^{lower}$, i.e., the new bound is tighter. Algorithms 7 through 9 list a possible pseudo code for this.

In our test, we used *incomplete pricing*, i.e., if we have found promising sets of variables with $l \in L(b)$, then we stop; if not, we look for promising sets of variables with $l \in \mathcal{L}' \setminus \bigcup_{b\in\mathcal{B}} L(b)$; if we find promising sets of variables there, then again we stop. If then we still have not found a promising set of variables, then we enter the more time-consuming search for new lot-types $l \in \mathcal{L} \setminus \mathcal{L}$. Note, that the characteristic shift bound can only be derived from complete pricing steps.

5.7 The subroutine SolveRSLDP($\mathcal{L}'$)

This subroutine builds an RSLDP with all variables corresponding to all branches $b \in \mathcal{B}$, lot-types $l \in \mathcal{L}'$, and multiplicities $m \in \mathcal{M}$. This RSLDP is then solved by the black-box ILP solver. There is one specialty for our implementation: We used the original model from Section 3 without the augmenting constraints. This means, the RSLDP will reproduce earlier solutions, which causes no harm. The reason for this is simply that our particular solver (`cplex`) could prove the optimality of an old solution in the smaller model more quickly than the infeasibility of the larger model with a cut-off at the current upper bound z^{upper}. Since this can be dependent on the solver software and even the version of it, we cannot make a general statement about which option is to be preferred.

6 Computational results

In this section we compare the algorithm ASG, implemented in C++, with the static solution of the ILP model from Section 3, denoted by STATIC_CPLEX.

The computational environment was as follows. The computer was an iMac (Retina 5K, 27inches, 2019) 3 GHz Intel Core i5 with 32GB 2667 MHz DDR4 RAM running `MacOSX 10.15.5` based on `Darwin Kernel Version 19.5.0`. We used `gcc/g++` from `Apple clang version 11.0.3 (clang-1103.0.32.62)`. As an LP- and ILP-solver we used `IBM ILOG CPLEX 12.9.0.0`, both for STATIC_CPLEX and for the RMP and RSLDP during the run of ASG. Profiling was performed using the `clock()` utility from `time.h`. Timing was consistent to an accuracy of $\sim \pm 5\%$. We performed all test for zero with $\epsilon = 10^{-9}$. We used the defaults for all solver

parameters. Thus, the solutions can not be expected to be more accurate than the solver's largest default accuracy "`mipgap`" of 10^{-4}, which is still far more accurate than the demand data.

For ASG, we used a time limit of $\max\{1.0, \log(|\mathcal{L}|)/10.0\}$ for SFA. Moreover, in the "default" parameter setting of ASG we use $K_1 = 3$ (for the maximal number of locally best lot-types determined for each branch), $K_2 = 3$ (for the maximal number of used lot-types from the score table), $K_3 = 0$ (for the maximal number of additional multiplicities), $K_4 = 10$ (for the maximal number of promising sets of columns for new branches assigned to an already generated lot-type), and $K_5 = 3$ (for the number of promising sets of columns with not yet generated lot-types). In a comparison, we checked the influence of parameter settings representing "fewer columns" ($K_1 = 1$, $K_2 = 3$, $K_3 = 0$, $K_4 = 1$, and $K_5 = 1$) and "more columns" ($K_1 = 10$, $K_2 = 5$, $K_3 = 2$, $K_4 = \infty$, and $K_5 = 10$). Furthermore, we used incomplete pricing to avoid the time-consuming branch-and-bound search for promising new lot-types whenever other promising columns could be generated.

Our tests encompass two substantially different sets of instances of the (S)LDP. First, we used a set of 860 real-world production instances with various instance parameters. All instances have three scenarios for the demands. One of these scenarios represents the nominal scenario, as defined in Section 2.2. For all real-world instances, we solved the SLDP (in terms of the equivalent LDP) and the nominal LDP, both by ASG and STATIC_CPLEX. To illustrate the gap between the SLDP and the nominal LDP we report on the difference in the objective function values and the value of the stochastic solution (VSS) [6, Section 4.2]. Moreover, we compared the run-times and the numbers of used columns for the two competing algorithms. Second, we compare the algorithms ASG and STATIC_CPLEX on a set of 5 groups of 9 random LDP-instances with identical parameters each. The motivation for this second set of instances is to safe-guard against the possibility that the results on real-world data are an artifact of some hidden structure in our special instances. Since the real-world instances exhibit only minor differences in cpu times between solving the nominal LDPs and the SLDPs, we restricted the tests for the random instances to the nominal LDP.

In the 860 real-world instances, the numbers of applicable lot-types range from 9 through 1,198,774,720, the numbers of branches from 1370 to 1686, and the numbers of sizes from 2 to 7. In the static model, the numbers of variables range from 8381 to 9,867,114,720,320, and the numbers of constraints from 3355 to 1,973,183,190,769. About half of the instances have at least 125,038,518 variables and 25,006,257 constraints. Since the larger instances are too large to read the complete static ILP-model into our computer, we used STATIC_CPLEX only for those instances with at most 2000 applicable lot-types, leading to 334 instances with 8381 to 15,344,420 variables and 3355 to 3,070,209 constraints. In this set, there were also 8 obviously infeasible instances for various reasons that can be attributed to errors in the data handling or pathologic historic demand data at the time. We implemented a straight-forward procedure to filter them out automatically. ASG's memory consumption stayed well below 1GB RAM, thanks to an implicit encoding of lot-types via their lexicographic index in our implementation. Note that the wide range of parameters in the daily production data stems from the fact that different ware groups have different numbers of sizes and different restrictions for the numbers of pieces in the lot-types. For example, while winter coats usually come in four sizes with only very few pieces allowed in a lot-type, womens' underwear comes in twelve sizes with many pieces possible in a lot-type. Some of the instances are similar in the optimal objective function value because the demand data has been estimated aggregated over the whole ware group (the sample sizes for estimation are too small otherwise).

In the 5×9 random instances, the parameters in the 5 groups are given in Table 1. The parameter sets are similar to those found in the real data. For these 5 sets of parameters, we generated 9 sets of demand data uniformly at random so that the resulting instances were

consistent.

$\|\mathcal{B}\|$	$\|\mathcal{S}\|$	$\|\mathcal{M}\|$	$\min_c$	$\max_c$	$\min_t$	$\max_t$
10	4	5	0	2	4	8
10	4	5	0	5	3	15
1303	4	5	0	5	3	15
1328	7	5	1	3	7	14
682	12	5	0	5	12	30

$\|\mathcal{L}\|$	k	#vars	#cons
50	3	2550	513
1211	5	61,761	12,123
1211	5	7,890,876	1,579,239
1290	4	8,566,890	1,714,451
1,159,533,584	5	3,955,169,055,024	790,801,904,973

Table 1: The parameters of the $5 \times 9 = 45$ random instances.

Let us first discuss the results on the 860 real-world instances. In those instances that were solved both statically and dynamically we obtained identical optimal values up to the expected numerical accuracy.

metric	average	median	maximum
ELDP – LDP	1638.48	1654.25	5033.04
(ELDP – LDP) / ELDP	66.69 %	66.40 %	200.59 %
VSS = ELDP – SLDP	15.69	6.49	263.34
relative VSS = (ELDP – SLDP) / ELDP	0.36 %	0.15 %	4.51 %

Table 2: Various metrics indicating the differences in the model optimal solutions for the SLDP and the nominal LDP: LDP denotes the optimal objective of the nominal LDP; ELDP denotes the expected cost of the nominal LDP solution over all three scenarios; SLDP denotes the optimal objective of the SLDP; VSS is the value of the stochastic solution

We compared ASG and STATIC_CPLEX for the SLDP (via the equivalent LDP) and the nominal LDP in Table 2. It can be seen that solving the nominal LDP instead of the SLDP significantly underestimates the expected cost. However, the quality of its optimal solution evaluated in the SLDP is almost optimal: the relative value of the stochastic solution (relative VSS) is 0.36 % on average; no instance showed a relative VSS of more than 4.51 %. That means, if underestimation of cost and the rare occurrence of a larger relative VSS are not an issue, one can safely solve the nominal LDP instead of the SLDP. It is a different question whether this pays off, as the next results show.

metric	ASG on nominal LDP			ASG on SLDP		
	default	fewer col's	more col's	default	fewer col's	more col's
median cpu-time (small)	4.00 s	8.08 s	9.68 s	10.90 s	10.79 s	11.54 s
percentage of STATIC_CPLEX	0.67 %	0.81 %	0.81 %	1.13 %	0.96 %	1.02 %
median cpu-time (all)	12.21 s	12.68 s	14.23 s	13.39 s	13.30 s	15.28 s
median cpu-time (large)	22.84 s	25.60 s	33.13 s	26.13 s	32.52 s	29.83 s

Table 3: Cpu-time comparison for various parameter settings for ASG solving the SLDP or the nominal LDP; here, the "small" instances are the ones that were solved by both STATIC_CPLEX and ASG (at most 1820 applicable lot-types), the "large" instances are the 100 largest ones (134,596 through 1,198,774,720 applicable lot-types)

Table 3 compares cpu-times on three parameter settings for ASG, both for solving the nominal LDP and the SLDP. Over all instances in the default setting, solving the nominal LDP was around 10 % faster on average than solving the SLDP. Still, all cpu-times are usable in practice. It is a matter of the user's preferences whether to pay the 10 % extra cpu-time to obtain the better cost predictions of the SLDP. Second, there is no unique fastest parameter setting. Since in no category (LDP or SLDP; small, all, or large instances) "more columns" is the best, it is safe to go with "default" or "fewer columns", where the "default setting" has the edge for the large instances. For us the most important result is that for *all three* investigated and substantially different parameters settings ASG outperforms STATIC_CPLEX by a median-factor of around 100 and solves even the large instances in only mildly longer cpu-times. In other words, our main result does not at all depend on the parameter setting.

metric	ASG on nominal LDP			ASG on SLDP		
	default	fewer col's	more col's	default	fewer col's	more col's
median no. of cols (small)	6461	6404	28553	6846	6634	28676
percentage of STATIC_CPLEX	0.18 %	0.18 %	0.78 %	0.19 %	0.18 %	0.79 %
median no. of cols (all)	7050	6916	29089	7280	7072	29338
median no. of cols (large)	7136	6871	29362	7234	6981	29476

Table 4: Comparison of the numbers of columns for various parameter settings for ASG solving the SLDP or the nominal LDP; here, the "small" instances are the ones that were solved by both STATIC_CPLEX and ASG (at most 1820 applicable lot-types), the "large" instances are the 100 largest ones (134,596 through 1,198,774,720 applicable lot-types)

Table 4 compares the number of generated columns until optimality is proved by ASG. On average over all instances in the "default" setting, the nominal LDP needs 3 % fewer columns than the SLDP. The parameter setting of "more columns" leads to an unnecessary large number of columns, whereas "default" and "few columns" generate similar amounts of columns, making both of them perfectly usable in practice.

Summary: The analysis of the different parameter settings shows that ASG works quickly for all three settings, though "more columns" wastes some memory for unnecessary columns. Moreover, solving the SLDP has the advantage over the nominal LDP of a precise cost prediction at a small cpu-time overhead, but the quality of the SLDP solution is only insignificantly better than the nominal LDP solution.

After this summary of results aggregated over all 860 real-world instances for both the nominal LDP and the SLDP, we have a look at the performances of ASG and STATIC_CPLEX depending

on the number of applicable lot-types, which is, in our opinion, the single-most important parameter for the size of an instance. We restrict ourselves to the solution of the nominal LDP in the following.

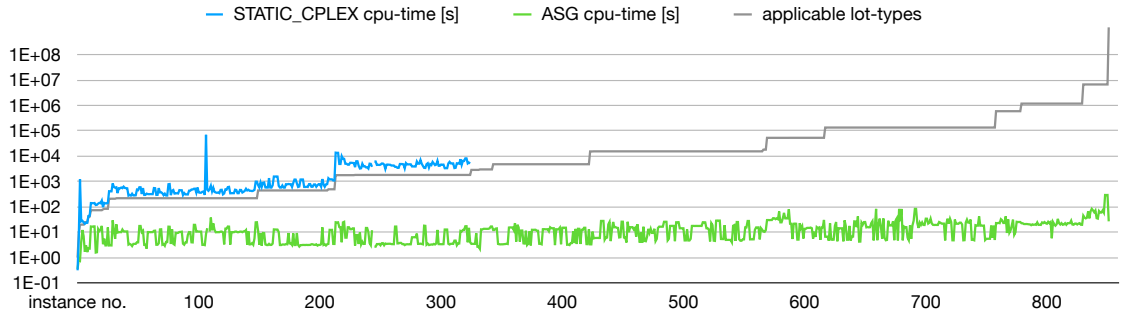

Figure 1: Comparison of cpu times in seconds on 860 real-world instances ordered by the numbers of applicable lot-types (indicated by the grey line); note that we connected all data points by straight lines for better visibility, although we have a discrete number of 860 instances on the category axis

Figure 1 shows the cpu times for STATIC_CPLEX and the ASG solution. The instances have been weakly ordered from left to right by their numbers of applicable lot-types. The numbers of applicable lot-types is shown by the grey graph in the chart. The scale for the time axis in seconds is logarithmic. We see that ASG is consistently two to three orders of magnitude faster than STATIC_CPLEX. It is striking that the cpu time of ASG is essentially stable around 10 seconds, usually never exceeding 100 seconds over the whole range of instances with only very few instances over 100 seconds. In contrast to this, the cpu times of STATIC_CPLEX grow more or less proportional to the numbers of applicable lot-types. The visible outlier for the cpu time of STATIC_CPLEX is instance 107 – we will see below that it exhibits an exceptionally large integrality gap of around 10 %. ASG is not equally affected by this problem, most probably because the optimality gap of SFA is zero for this instance, as we can see below as well.

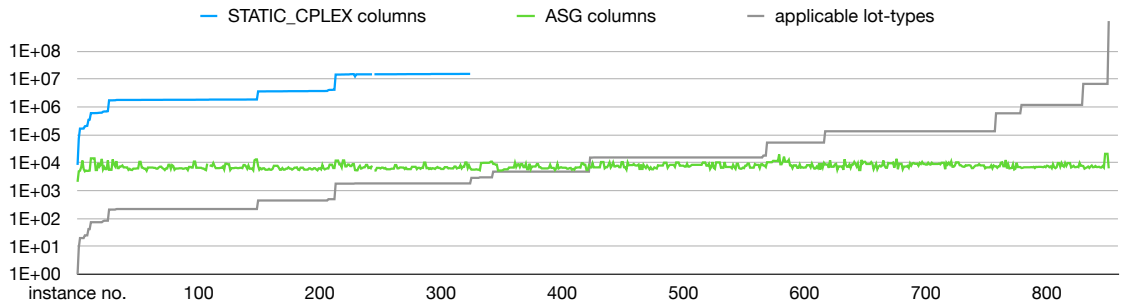

Figure 2: Comparison of columns needed in seconds on 860 real-world instances ordered by the numbers of applicable lot-types (indicated by the grey line); note that we connected all data points by straight lines for better visibility, although we have a discrete number of 860 instances on the category axis

The reason for this success can be explained from Figure 2, where the total numbers of

columns used in the solution procedure is shown on the logarithmic scale. It can be seen that ASG needs consistently only around 10,000 columns to find and prove the optimum, whereas STATIC_CPLEX, of course, uses all columns of the model, which are two to three orders of magnitude more for the considered instances. Although we know the numbers of columns that STATIC_CPLEX would have used if there had been enough memory available, we did not include these numbers in the chart because the computation did not take place.

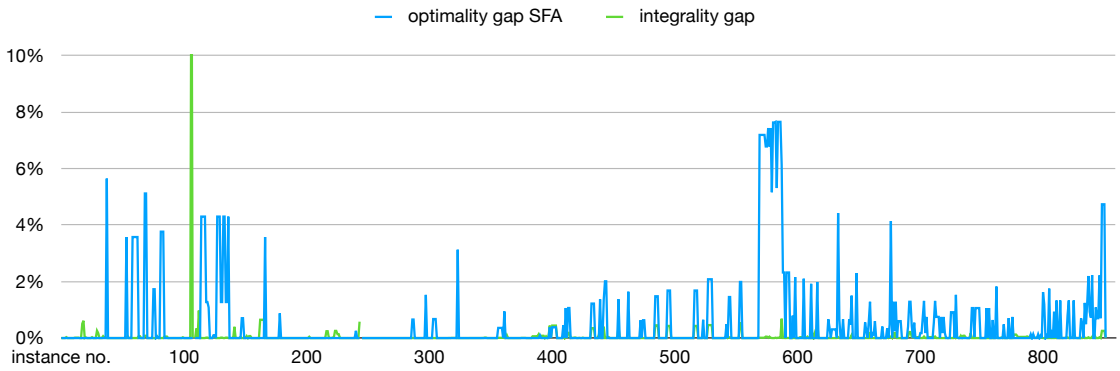

Figure 3: Gaps for the 860 real-world instances ordered by the numbers of applicable lot-types; note that we connected all data points by straight lines for better visibility, although we have a discrete number of 860 instances on the category axis

We wanted to find success factors of our model and algorithm. In our opinion, a tight model and a reasonable heuristics are two important success factors in a branch-and-price method. Figure 3 confirms that the integrality gap of our SLDP-model is small throughout with a single exception (instance no. 107 with 10 %), and also the optimality gap of the SFA heuristics is in most cases small (often it is zero, e.g., for instance 107 with the large integrality gap). We have also considered the polynomially-sized model in Appendix A. Because it exhibits a very large integrality gap (see Appendix A), we dismissed it in favor of the exponential model in this paper. Figure 3 provides evidence for the fact that our formulation with many variables is quite tight. The computation times show that, in both the smaller and the larger instances, our new algorithm ASG can take advantage of this increased tightness.

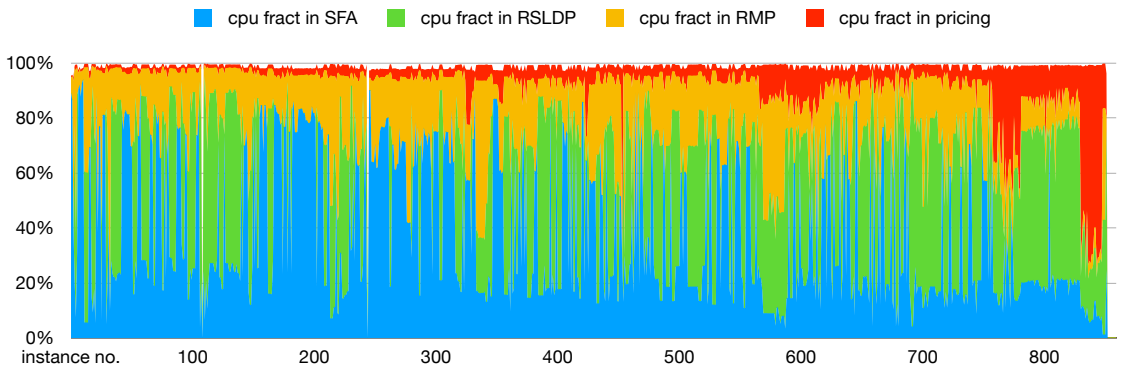

Figure 4: The relative CPU usage for ASG in percent on 860 real-world instances ordered by the numbers of applicable lot-types; note that we connected all data points by straight lines for better visibility, although we have a discrete number of 860 instances on the category axis

We wanted to know whether ASG has serious bottlenecks. To this end, we show in Figure 4 the fractions of the CPU time spent in the most important subroutines of ASG. It can be seen that while in the fast instances most of the (short) time is spent in the SFA heuristics, in the slow instances more time is spent in the RSLDP solving. Only in the largest instances the fraction of time spent in the pricing routine becomes dominant. We have checked the effectiveness of pricing with respect to the canonical lifting instead of the characteristic lifting in Section 4, and substantially more columns have been generated resulting in larger CPU times. Thus, the results in Section 4 are beneficial for the success of ASG.

Let us now turn to the results for the 5×9 random instances. Because we already concluded that the difference in solving the nominal LDP and the SLDP is minor compared to the difference to STATIC_CPLEX, we restricted ourselves to the solution of randomly generated deterministic LDPs.

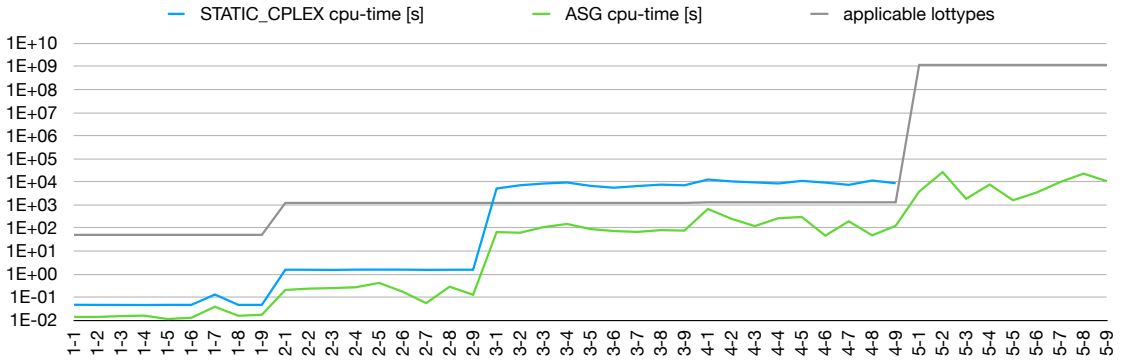

Figure 5: Comparison of CPU times in seconds on $5 \times 9 = 45$ random instances ordered by the numbers of applicable lot-types (indicated by the grey line); note that we connected all data points by straight lines for better visibility, although we have a discrete number of 5×9 instances on the category axis

Figure 5 shows the CPU times in seconds in the same way as for the real-world instances. We see that ASG is one to two orders of magnitude faster than STATIC_CPLEX and can solve all consistent instances. The instances from the largest group, however, take much longer this

time. The largest observed CPU time of 26,362.7 seconds was spent in instance 2 of the 5th group. The performance difference compared to the real-world instances (some of which have the about the same size) can be explained. In the real-world, the expected demand for sizes is not uniform over all branches. There is a hidden imbalance, and 5 lot-types are enough to cover all the variations quite well. In the random instances, the expected demands are identical for all sizes in all branches. Thus, any lot-type incurs almost the same cost as all its size permutations. Since for the large random instances with 12 sizes the number of permutations leading to a different lot-type is much larger than $k = 5$ (except for lot-types using the same number of pieces in every size), there are many distinct lot-type designs with almost the same costs. This makes the long computation times plausible. In a sense, the random instances constitute a stress test for ASG, and we find that the performance compared to STATIC_CPLEX is still impressive.

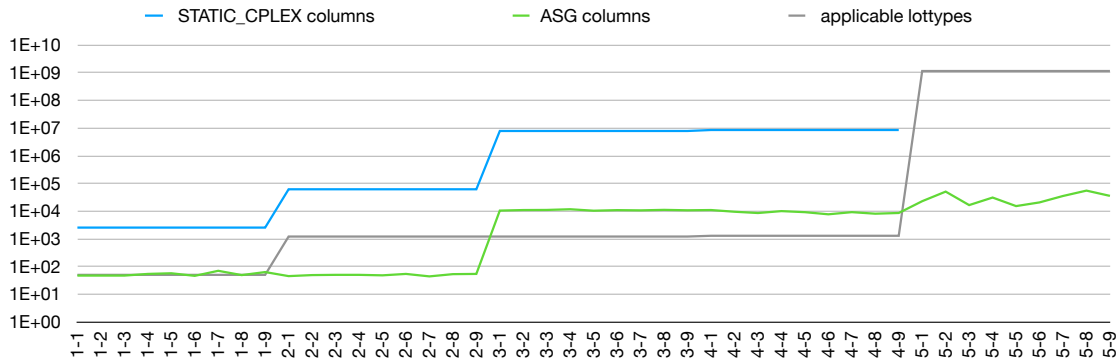

Figure 6: Comparison of columns needed in seconds on $5 \times 9 = 45$ random instances ordered by the numbers of applicable lot-types (indicated by the grey line); note that we connected all data points by straight lines for better visibility, although we have a discrete number of 5×9 instances on the category axis

The reason for the performance difference between ASG and STATIC_CPLEX again correlates with the total number of columns used during the solution process. Figure 6 shows that the number of columns used by ASG is two to three orders of magnitude smaller. The numbers of columns used is stable for the first four groups of instances and starts to vary in the group of the largest instances.

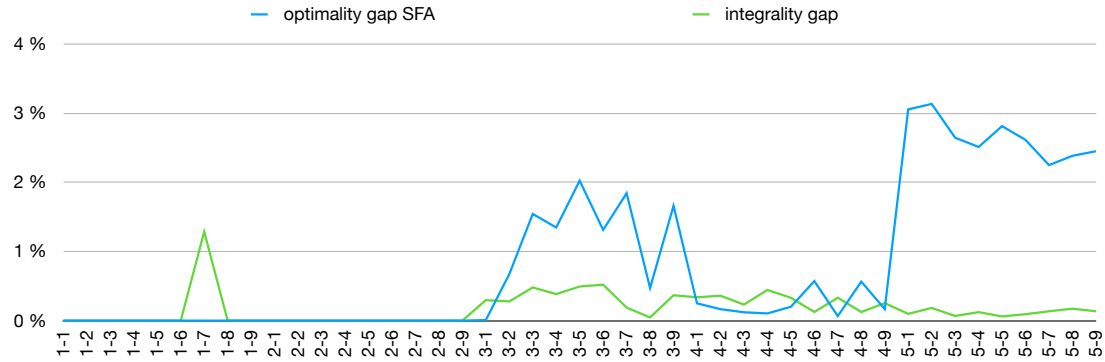

Figure 7: Gaps for the $5 \times 9 = 45$ random instances ordered by the numbers of applicable lot-types; note that we connected all data points by straight lines for better visibility, although we have a discrete number of 5×9 instances on the category axis

We see in Figure 7 showing the integrality gap and the SFA gap that the SFA gap in groups 3 and 5 is substantially larger than in the other groups whereas the integrality gap is very small throughout with only very few exceptions. Note, however, that the integrality gap is small everywhere by any practically relevant measure.

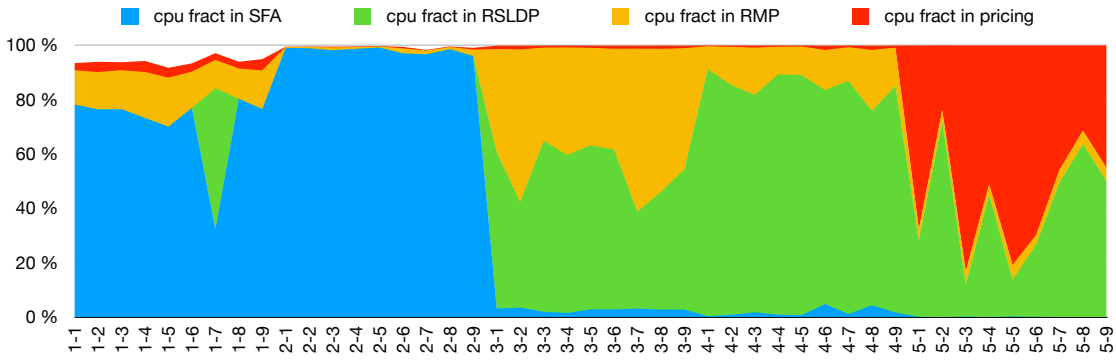

Figure 8: The relative CPU usage for ASG in percent on $5 \times 9 = 45$ random instances ordered by the numbers of applicable lot-types; note that we connected all data points by straight lines for better visibility, although we have a discrete number of 5×9 instances on the category axis

The analysis of possible bottlenecks in Figure 8 shows a similar result as for the real-world instances. Only in the very large instances, a substantial amount of time is spent in pricing. For the small instances, the CPU time is dominated by the time limit for SFA.

To sum up, we see that ASG is substantially faster and more memory-efficient than STATIC_CPLEX, even in smaller instances. That is, for this particular problem column generation should not only be used for the instances that do not fit into the computer but for *all* instances.

7 Conclusion and future work

We have considered the stochastic lot-type design problem SLDP, which is an industrially relevant problem. We provided a tight ILP model of it. This model, however, has in many cases too many

variables for solvers off the shelf. Thus, we presented a branch-and-price algorithm ASG that was able to solve a large number of real-world and random LDP of various scales exactly to proven optimality orders of magnitude more quickly than static ILP-solving. The pricing routine in ASG benefits from a characteristic lifting of dual variables that provably avoids the generation of only seemingly promising but actually ineffective columns.

It would be valuable to find out whether the principles of ASG can be generalized to a problem independent ILP setting. The main reason not to implement branch-and-price is the amount of effort to close a small gap after an ILP has been solved based on all the columns necessary for an LP optimum. ASG might serve as an intermediate technique whose implementation effort is reasonable. Another added bonus of ASG is that all progress in commercial ILP-solver technology is exploited.

Acknowledgments

We thank two anonymous referees for their suggestions that greatly improved the readability of the paper.

References

[1] A. Agra, J. O. Cerdeira, and C. Requejo. A decomposition approach for the p-median problem on disconnected graphs. *Comput. Oper. Res.*, 86:79–85, 2017.

[2] K. Ahsan and A. Azeem. Insights of apparel supply chain operations: a case study. *Int. J. Integr. Supply Manag.*, 5(4):322–343, 2010.

[3] A. Aouad, V. Farias, R. Levi, and D. Segev. The approximability of assortment optimization under ranking preferences. *Oper. Res.*, 66(6):1661–1669, November 2018.

[4] P. Avella, A. Sassano, and I. Vasilyev. Computational study of large-scale p-median problems. *Math. Program.*, 109(1):89–114, 2007.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.*, 46(3):316–329, 1998.

[6] J. R. Birge and F. Louveaux. *Introduction to stochastic programming. 2nd ed.* Springer Series in Operations Research and Financial Engineering. New York, NY: Springer. xxv, 485 p., 2011.

[7] M. Boccia, A. Sforza, C. Sterle, and I. Vasilyev. A cut and branch approach for the capacitated p-median problem based on Fenchel cutting planes. *J. Math. Model. Algorithms*, 7(1):43–58, 2008.

[8] Z. Drezner, J. Brimberg, N. Mladenović, and S. Salhi. New heuristic algorithms for solving the planar p-median problem. *Comput. Oper. Res.*, 62:296–304, 2015.

[9] C. C. Fast and I. V. Hicks. A branch decomposition algorithm for the p-median problem. *INFORMS J. Comput.*, 29(3):474–488, 2017.

[10] D. Feillet, M. Gendreau, A. Medaglia, and J. Walteros. A note on branch-and-cut-and-price. *Oper. Res. Lett.*, 38(5):346–353, 2010.

[11] M. Fisher and R. Vaidyanathan. A demand estimation procedure for retail assortment optimization with results from implementations. *Manage. Sci.*, 60(10):2401–2415, 2014.

[12] C. Gaul, S. Kurz, and J. Rambau. On the lot-type design problem. *Optim. Methods Softw.*, 25(2):217–227, 2010.

[13] J. Q. Hale, E. Zhou, and J. Peng. A Lagrangian search method for the P-median problem. *J. Global Optim.*, 69(1):137–156, 2017.

[14] P. Hansen and N. Mladenović. Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226, 1997.

[15] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. *A non-robust Branch-and-Cut-and-Price algorithm for the Vehicle Routing Problem with Time Windows.* Number 06-03 in Technical Report. DIKU, University of Copenhagen, Denmark, 2006.

[16] M. Kießling, S. Kurz, and J. Rambau. The integrated size and price optimization problem. *Numerical Algebra, Control and Optimization*, 2(4):669–693, Dezember 2012.

[17] A. Klose and S. Görtz. A branch-and-price algorithm for the capacitated facility location problem. *Eur. J. Oper. Res.*, 179(3):1109–1125, 2007.

[18] S. O. Krumke, J. Rambau, and L. M. Torres. Realtime-dispatching of guided and unguided automobile service units with soft time windows. In R. H. Möhring and R. Raman, editors, *Algorithms – ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17–21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*. Springer, 2002.

[19] S. Kurz, J. Rambau, J. Schlüchtermann, and R. Wolf. The top-dog index: a new measurement for the demand consistency of the size distribution in pre-pack orders for a fashion discounter with many small branches. *Annals OR*, 229(1):541–563, 2015.

[20] M. E. Lübbecke. Column generation. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010.

[21] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Oper. Res.*, 53(6):1007–1023, 2005.

[22] M. E. Lübbecke and J. Desrosiers. Branch-price-and-cut algorithms. In J. J. Cochran, editor, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2011.

[23] A. M. Marzouk, E. Moreno-Centeno, and H. Üster. A branch-and-price algorithm for solving the hamiltonian p-median problem. *INFORMS J. Comput.*, 28(4):674–686, 2016.

[24] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *Eur. J. Oper. Res.*, 179(3):927–939, 2007.

[25] I. Muter, S. Birbil, and K. Bulbul. Simultaneous column-and-row generation for large-scale linear programs with column-dependent rows. Technical Report SU_FENS_2010/0004, Sabanci University, 2010.

[26] A. Pessoa, M. P. de Aragão, and E. Uchoa. Robust branch-cut-and-price algorithms for vehicle routing problems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 297–325. Springer US, Boston, MA, 2008.

[27] P. Rebreyend, L. Lemarchand, and R. Euler. A computational comparison of different algorithms for very large p-median problems. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 13–24. Springer, 2015.

[28] A.-K. Rothenbächer, M. Drexl, and S. Irnich. Branch-and-price-and-cut for a service network design and hub location problem. *Eur. J. Oper. Res.*, 255(3):935–947, 2016.

[29] R. Sadykov. *Modern Branch-Cut-and-Price*. PhD thesis, Université de Bordeaux, 2019.

[30] R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, and E. Uchoa. Primal heuristics for branch and price: The assets of diving methods. *INFORMS J. Comput.*, 31(2):251–267, 2019.

[31] R. Václavík, A. Novák, P. Šucha, and Z. Hanzálek. Accelerating the branch-and-price algorithm using machine learning. *Eur. J. Oper. Res.*, 271(3):1055–1069, 2018.

[32] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Math. Program., Ser. A*, 130(2):249–294, 2011.

A A compact model for parameterizable sets of lot-types

For completeness, we want to mention that the large number of variables of our ILP formulation from Section 3 is far from being inavoidable. In the case of a parametrized set of lot-types, as described in Section 2, we can model the SLDP with fewer variables. However, the LP relaxation of the subsequent compact model yields large integrality gaps.

$$
\begin{aligned}
\min \quad & \sum_{b\in\mathcal{B}}\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \delta^a_{b,s} && \\
s.t. \quad & \sum_{i\in\mathcal{K}}\sum_{m\in\mathcal{M}} x_{b,i,m} = 1 && \forall b\in\mathcal{B} \\
& v_{b,s,i,m} - max_c \cdot x_{b,i,m} \le 0 && \forall (b,s,m,i)\in\mathcal{U} \\
& v_{b,s,i,m} - l_{i,s} \le 0 && \forall (b,s,m,i)\in\mathcal{U} \\
& v_{b,s,i,m} - l_{i,s} - max_c \cdot x_{b,i,m} \ge -max_c && \forall (b,s,m,i)\in\mathcal{U} \\
& \underline{I} \le \sum_{b\in\mathcal{B}}\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{K}} m\cdot v_{b,s,i,m} \le \overline{I} && \\
& l_{i,s} \le max_c && \forall i\in\mathcal{K}, s\in\mathcal{S} \\
& l_{i,s} \ge min_c && \forall i\in\mathcal{K}, s\in\mathcal{S} \\
& \sum_{s\in\mathcal{S}} l_{i,s} \le max_t && i\in\mathcal{K} \\
& \sum_{s\in\mathcal{S}} l_{i,s} \ge min_t && \forall i\in\mathcal{K} \\
& \delta^a_{b,s} + \sum_{i\in\mathcal{K}}\sum_{m\in\mathcal{M}} m\cdot v_{b,s,i,m} \ge d^a_{b,s} && \forall b\in\mathcal{B}, s\in\mathcal{S}, a\in\mathcal{A} \\
& \delta^a_{b,s} - \sum_{i\in\mathcal{K}}\sum_{m\in\mathcal{M}} m\cdot v_{b,s,i,m} \ge -d^a_{b,s} && \forall b\in\mathcal{B}, s\in\mathcal{S}, a\in\mathcal{A} \\
& x_{b,i,m} \in \{0,1\} && \forall b\in\mathcal{B}, i\in\mathcal{K}, m\in\mathcal{M} \\
& v_{b,s,i,m} \ge 0 && \forall (b,s,m,i)\in\mathcal{U}
\end{aligned}
$$

$$l_{i,s} \in \mathbb{Z} \qquad \forall i \in \mathcal{K}, s \in \mathcal{S}$$
$$\delta^a_{b,s} \geq 0 \qquad \forall b \in \mathcal{B}, s \in \mathcal{S}, a \in \mathcal{A},$$

where we use the abbreviations $\mathcal{K} := \{1, \ldots, k\}$ and $\mathcal{U} := \mathcal{B} \times \mathcal{S} \times \mathcal{M} \times \mathcal{K}$. Obviously, both the number of variables and the number of constraints is polynomial in the input parameters, in contrast to the model in Section 3. The meaning of the variables and constraints is as follows. We utilize the binary variable $x_{b,i,m}$ to model the assignment of the lot-type and multiplicity to a certain branch b, i.e., we have $x_{b,i,m} = 1$ if and only if branch b is supplied with the ith selected lot-type in multiplicity m. Of course, for each branch b only one $x_{b,i,m}$ is one. In order to incorporate the bounds on the total number of supplied items we utilize the auxiliary variables $v_{b,s,i,m}$. For a given branch b and size s we set $v_{b,s,i,m} = l_{i,s} \cdot x_{b,i,m}$, i.e. $v_{b,s,i,m} = l_{i,s}$ if branch b is supplied with lot-type i in multiplicity m and $v_{b,s,i,m} = 0$ otherwise. The linearization of this non-linear equation is quite standard using suitable big-M constants. The deviations $\delta^a_{b,s}$ then are given by

$$\delta^a_{b,s} = \left| d^a_{b,s} - \sum_{i=1}^{k} \sum_{m \in \mathcal{M}} m \cdot v_{b,s,i,m} \right|.$$

Again the linearization of this non-linear equation is standard.[3]

Since the symmetric group on k elements acts on the k different lot-types one should additionally destroy the symmetry in the stated ILP formulation. This can be achieved by assuming that the lot-types $(l_{i,s})_{s \in \mathcal{S}}$ are lexicographically ordered. For the ease of notation we assume that the set of sizes $\mathcal{S}$ is given by $\{1, \ldots, s\}$. As an abbreviation we set $t := \max_c - \min_c + 1$. With this the additional inequalities

$$\sum_{j=1}^{s} \left(l_{i,j} - \min_c \right) \cdot t^{s-j} \geq 1 + \sum_{j=1}^{s} \left(l_{i+1,j} - \min_c \right) \cdot t^{s-j} \qquad \forall 1 \leq i \leq k-1,$$

which are equivalent to $\sum_{j=1}^{s} (l_{i,j} - l_{i+1,j}) \cdot t^{s-j} \geq 1$ for all $1 \leq i \leq k-1$, will do the job. We remark that using some additional auxiliary variables a numerical stable variant can be stated easily. (This becomes indispensable if t^s gets too large.)

We remark that the LP relaxation of the compact model yields large integrality gaps [4]. The typical approach would now be to solve a Dantzig-Wolfe decomposition of the compact model by dynamic column generation, leading to a master problem very similar to the SLDP model we presented in the first place. And column generation on that SLDP model is what we proposed in the paper.

[3] We remark that one can express the deviations $\delta^a_{b,s}$ also using the binary assignment variables $x_{b,i,m}$ instead of the item counts $v_{b,s,i,m}$. In this case we have to replace the two inequalities containing $\delta^a_{b,s}$ by $\delta^a_{b,s} + m \cdot l_{i,s} - d^a_{b,s} \cdot x_{b,i,m} \geq 0$ and $\delta^a_{b,s} - m \cdot l_{i,s} + m \cdot max_c \cdot (1 - x_{b,i,m}) \geq -d^a_{b,s}$ for all $(b, s, m, i) \in \mathcal{U}, a \in \mathcal{A}$. Or we may use both sets of inequalities.

[4] Assuming some technical conditions on the demands, the four parameters for the lot-types, and the applicable multiplicities, which are not too restricting, one can construct a solution of the LP relaxation having an objective value of zero. E.g. we assume $\min_c \leq d^a_{b,s} \leq \max_c \cdot \max\{m \in \mathcal{M}\}$