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

December 13, 2019

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 sub-problems, 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 (S, M, L, 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 [9]. 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.

For many practical instances the set of applicable lot-types and thus the number of variables is so large that the ILP formulation from [9] cannot be solved directly. In this paper, we therefore propose a column generation approach. For our problem adding a new variable or column requires the introduction of additional constraints in most cases. So we have to generate columns and cuts simultaneously. Similar

1E.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.
problems and approaches have been addressed in [8, 13]. In order to overcome the integrality gap of the ILP relaxation we propose a tailored branching scheme complemented by the use of additional cover cuts. This results in an exact column generation approach for the LDP, which is enhanced by properly chosen algorithms for important subproblems. We apply this algorithm to a stochastic version SLDP of the LDP, where the expectation over more than one demand-scenario is optimized. The SLDP is of the same form as the LDP, and thus all optimization techniques viable for the LDP immediately apply to the SLDP. Since using the SLDP instead of the LDP can make decisions more robust against forecasting errors, we based our investigations in this paper on the SLDP.

Branch-and-price algorithms are common for large-scale integer programming problems [14]. Unifying general remarks can be found in [3, 23]. Branch-price-and-cut algorithms are surveyed in [15]. A few more recent papers on algorithmic issues are e.g. [21, 22, 24]. 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 [2, 5, 12, 19]. Other algorithmic approaches can e.g. be found in [11, 17, 10, 6, 1, 7, 16].

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.

Data. Let \( B \) be the set of branches, \( S \) be the set of sizes, and \( M \subseteq \mathbb{N} \) be an interval of possible multiples. A lot-type is a vector \( (l_s)_{s \in S} \in \mathbb{N}^{|S|} \), \( l \) is applicable if \( \min_c \leq l_s \leq \max_c \) for all \( s \in S \) and \( \min_t \leq \sum_{s \in S} l_s \leq \max_t \).

By \( L \) we abbreviate the set of applicable lot-types. There is an upper bound \( T \) and a lower bound \( I \) 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 expected demand at branch \( b \) in size \( s \).

Decisions. Consider an assignment of a unique lot-type \( l(b) \in L \) and an assignment of a unique multiplicity \( m(b) \in M \) to each branch \( b \in B \). These data specify that \( m(b) \) lots of lot-type \( l(b) \) are to be delivered to branch \( b \).

Objective. The goal is to find a subset \( L \subseteq L \) of at most \( k \) lot-types and assignments \( l(b) \in L \) and \( m(b) \in M \) such that the total supply is within the bounds \([I, T]\), and the deviation between inventory and demand is minimized.

We call this optimization problem the Lot-Type Design Problem (LDP), see [9] for more details. Using the introduced decision variables \( L \), \( l(b) \), and \( m(b) \), we can express the relevant decision-dependent entities as follows. The inventory of branch \( b \) in size \( s \) given assignments \( l(b) \) and \( m(b) \) is given by \( I_{b,s}(l, m) = m(b)l(b)_s \).

Moreover, the total supply resulting from \( l(b) \) and \( m(b) \) is given by \( I(l, m) = \sum_{b \in B} \sum_{s \in S} I_{b,s}(l, m) \).

This deterministic model can slightly be enhanced to a stochastic model by considering a set \( \Omega \) of scenarios (for the success of the product). For each scenario \( \omega \in \Omega \) we denote by \( p^\omega \) its probability and with \( d_{b,s}^\omega \in \mathbb{Q}_{\geq 0} \) the demand at branch \( b \) in size \( s \) in scenario \( \omega \) for all \( b \in B \) and \( s \in S \). The goal then is to minimize the expected total deviation between inventory and demand.

\footnote{A parameterizable set of applicable lot-types is a practically relevant case: By setting \( \min_c = 1 \) we can enforce that each branch is supplied in each size with at least one item, a requirement which legally arises for advertised products. Since the main advantage of using lot-types lies in the reduction of the number of picks in the central warehouse, we should guarantee, that this effect does not dwindle away by selecting lot-types with too few items, which can be controlled by a suitable value for \( \min_t \). There are practical reasons for the parameter \( \max_t \), too: combining too many winter coats in a lot would cause serious handling problems.}
We call this single-stage stochastic optimization problem the \textit{Stochastic Lot-Type Design Problem (SLDP)}. The SLDP is equivalent to an ordinary LDP with a modified objective function, since the expected total deviation $\Delta(l,m)$ can be written as $\sum_{b \in B} \sum_{s \in S} \sum_{w \in \Omega} p^w \Delta^w_{b,s}(l,m)$, where $\Delta^w_{b,s}(l,m) := |d^w_{b,s} - I^w_{b,s}(l,m)|$. In other words, the certainty equivalence principle (see e.g. [4, p. 28]) holds if the input data are the expected deviations for all branches and sizes. Certainty equivalence does not hold if the input data are the expected demands, though. In this paper, we restrict the experiments to the case of the deterministic LDP.

3 Modelling

For each branch $b \in B$ we must chose a set $L$ of selected lot-types with $|L| \leq k$ and assign a lot-type $l = l(b) \in L$ and a multiplicity $m = m(b) \in \mathcal{M}$. In order to model the SLDP as an integer linear program we use binary assignment variables $x_{b,l,m}$. For the used lot-types we use binary selection variables $y_l$ indicating whether $l \in L$. As an abbreviation we utilize $|l| := \sum_{s \in S} l_s$ for the number of pieces contained in lot-type $l$.

\begin{align*}
\min & \quad \sum_{b \in B} \sum_{l \in L} \sum_{m \in \mathcal{M}} c_{b,l,m} \cdot x_{b,l,m} \\
\text{s.t.} & \quad \sum_{l \in L} \sum_{m \in \mathcal{M}} x_{b,l,m} = 1 \quad \forall b \in B \\
& \quad \sum_{m \in \mathcal{M}} x_{b,l,m} \leq y_l \quad \forall b \in B, l \in L \\
& \quad \sum_{l \in \mathcal{L}} y_l \leq k \\
& \quad I \leq \sum_{b \in B} \sum_{l \in L} \sum_{m \in \mathcal{M}} m \cdot |l| \cdot x_{b,l,m} \leq T \\
& \quad x_{b,l,m} \in \{0,1\} \quad \forall b \in B, l \in \mathcal{L}, m \in \mathcal{M} \\
& \quad y_l \in \{0,1\} \quad \forall l \in \mathcal{L},
\end{align*}

where $c_{b,l,m} = \sum_{a \in A} p^a \cdot \sum_{s \in S} |d^a_{b,s} - m \cdot l_s| \geq 0$.

In the following, we only want to deal with reasonable SLDP instances.

\textbf{Definition 1.} An SLDP instance is \textit{consistent}, if

(i) it is feasible

(ii) the total demand lies in the cardinality interval, i.e., $I \leq \sum_{b \in B} \sum_{s \in S} d_{b,s} \leq T$ (demand consistency)

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

4 The theoretical foundations

In this subsection, we present the underlying theory in more detail. 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, 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 $\sum_{l \in \mathcal{L}\setminus \mathcal{L}'} y_l + s \geq 1$ with a slack variable $s$ whose cost coefficient $C$ is 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 $y_l \leq \sum_{m \in \mathcal{M}, b \in \mathcal{B}} x_{blm}$ 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:

\[
\begin{align*}
\min \quad & \sum_{b \in \mathcal{B}} \sum_{l \in L(b)} \sum_{m \in M(b,l)} c_{b,l,m} \cdot x_{b,l,m} & \quad \text{(duals:)} & \quad (8) \\
\text{s.t.} \quad & \sum_{l \in L(b)} \sum_{m \in M(b,l)} x_{b,l,m} = 1 & \quad \forall b \in \mathcal{B} & \quad (\alpha_b) \quad (9) \\
& \sum_{m \in M(b,l)} -x_{b,l,m} + y_l \geq 0 & \quad \forall b \in \mathcal{B}, l \in L(b) & \quad (\beta_{b,l}) \quad (10) \\
& \sum_{l \in \mathcal{L}'} -y_l \geq -k & \quad \quad & \quad (\gamma) \quad (11) \\
& \sum_{b \in \mathcal{B}, l \in L(b)} \sum_{m \in M(b,l)} x_{b,l,m} - y_l \geq 0 & \quad \forall l \in \mathcal{L}' & \quad (\delta_l) \quad (12) \\
& \sum_{l \in \mathcal{L}\setminus \mathcal{L}'} y_l + s \geq 0 & \quad \quad & \quad (\mu) \quad (13) \\
& \sum_{b \in \mathcal{B}, l \in L(b)} \sum_{m \in M(b,l)} m \cdot |l| \cdot x_{b,l,m} \geq I & \quad \quad & \quad (\phi) \quad (14) \\
& \sum_{b \in \mathcal{B}, l \in L(b)} \sum_{m \in M(b,l)} -m \cdot |l| \cdot x_{b,l,m} \geq -I & \quad \quad & \quad (\psi) \quad (15) \\
\end{align*}
\]

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

\[
\begin{align*}
\max \quad & \sum_{b \in \mathcal{B}} \alpha_b - k\gamma + I\phi - I\psi + \mu & \quad \text{(primals:)} & \quad (18) \\
\text{s.t.} \quad & \alpha_b - \beta_{b,l} + \delta_l + m|l|\phi - m|l|\psi \leq c_{b,l,m} & \quad \forall b \in \mathcal{B}, l \in L(b), m \in M(b,l) & \quad (16) \\
& y_l \geq 0 & \quad \forall l \in \mathcal{L}' & \quad (17)
\end{align*}
\]
of the dual variables (canonical lifting) fully compensating We call a cost-invariant lifting (uncompensated) reduced cost. The variables \((\hat{x}, \hat{y})\) for all \(b, l, m\) are optimal for \((\alpha, \beta, \gamma, \delta, \mu, \phi, \psi)\) 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 \((\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\delta}, \hat{\mu}, \hat{\phi}, \hat{\psi})\) for all \(b \in B, l \in L, m \in M\) 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 \((L, L, M)\) 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.,

\[
\bar{c}_{b,l,m} := c_{b,l,m} - \alpha_b + \beta_{b,l} - \gamma - \delta_l - m|l|(\hat{\phi} - \hat{\psi}).
\]

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.,

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

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:

$$\min \left\{ \min_{b \in B} \{ c_{b,l,m} - \hat{\alpha}_b + \hat{\beta}_{b,l} - \delta_l - m|l(\phi - \psi) : b \in B, l \in L, m \in M \} , \right.$$  

$$\min \left\{ - \sum_{b \in B} \hat{\beta}_{b,l} + \hat{\gamma} + \delta_l - \hat{\mu} : l \in L \} \right\},$$  \hspace{1cm} (28)

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 $(x)^- := \max\{-x, 0\} = -\min\{x, 0\}$.

**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

$$\hat{\beta}_{b,l} := \begin{cases} \beta_{b,l} & \text{if } l \in L(b), \\ \min_{m \in M} \bar{c}_{b,l,m}^- & \text{if } l \in L' \setminus L(b) \cup L \setminus L', \end{cases}$$  \hspace{1cm} (29)

$$\hat{\delta}_l := \begin{cases} \delta_l & \text{if } l \in L', \\ \min_{m \in M} \bar{c}_{b,l,m}^+ & \text{if } l \in L \setminus L'. \end{cases}$$  \hspace{1cm} (30)

Moreover, define as an abbreviation

$$\bar{c}_{b,l,m} := \begin{cases} - \sum_{l \in L \setminus \{ b \} \setminus L(b)} \hat{\beta}_{b,l} & \text{if } l \in L' \setminus \bigcap_{b \in B} L(b), \\ - \sum_{l \in L' \setminus \{ b \} \setminus L(b)} \hat{\beta}_{b,l} + \hat{\delta}_l & \text{if } l \in L \setminus L'. \end{cases}$$  \hspace{1cm} (31)

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

**Theorem 1.** Consider a primal-dual pair of optimal solutions $(x, y)$ 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

$$\bar{c}_{b,l,m} \geq 0 \hspace{2cm} \forall b \in B, l \in L(b), m \in M \setminus M(b,l),$$  \hspace{1cm} (32)

$$- \sum_{l \in L' \setminus \{ b \} \setminus L(b)} \left( \min_{m \in M} \bar{c}_{b,l,m}^- \right) \geq \sum_{l \in L' \setminus \{ b \} \setminus L(b)} \beta_{b,l} - \gamma - \delta_l + \mu \hspace{1cm} \forall l \in L' \setminus \bigcap_{b \in B} L(b),$$  \hspace{1cm} (33)

$$- \sum_{l \in L' \setminus \{ b \} \setminus L(b)} \left( \min_{m \in M} \bar{c}_{b,l,m}^+ \right) + \left( \min_{m \in M} \bar{c}_{b,l,m}^- \right) \geq -\gamma + \mu \hspace{1cm} \forall l \in L \setminus L',$$  \hspace{1cm} (34)

then the canonical lifting $(\bar{x}, \bar{y})$ of $(x, y)$ 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 \((x, y)\) and, therefore, the same objective as \((\hat{x}, \hat{y})\). Thus, \((\hat{x}, \hat{y})\) is optimal for the MP.

Next consider 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 \(\hat{c}_{b,l,m} = \tilde{c}_{b,l,m}\).

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

\[
\hat{c}_{b,l,m} - \alpha_b + \hat{\beta}_{b,l} - \hat{\delta}_l - m|l|(|\phi - \psi|) = \tilde{c}_{b,l,m} + \hat{\beta}_{b,l} \geq 0.
\]

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

\[
\hat{\beta}_{b,l} \geq \max_{m \in \mathcal{M}} (-\tilde{c}_{b,l,m}) = -\min_{m \in \mathcal{M}} \tilde{c}_{b,l,m}.
\]

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

\[
0 \geq \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu
\]

\[
\geq \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} + \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu
\]

\[
= \sum_{b \in \mathcal{B}} \max\{0, \hat{\beta}_{b,l}\} + \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu
\]

\[
\geq \sum_{b \in \mathcal{B}} \max\{0, -\min_{m \in \mathcal{M}} \tilde{c}_{b,l,m}\} + \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu
\]

\[
\geq \sum_{b \in \mathcal{B}} (\min_{m \in \mathcal{M}} \tilde{c}_{b,l,m}) - \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu.
\]

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

\[
c_{b,l,m} - \alpha_b + \tilde{\beta}_{b,l} - \tilde{\delta}_l - m|l|(|\phi - \psi|) = \tilde{c}_{b,l,m} + \tilde{\beta}_{b,l} - \tilde{\delta}_l \geq 0.
\]

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

\[
\hat{\beta}_{b^*, l} - \hat{\delta}_l \geq -\tilde{c}_{b^*, l,m^*} \geq 0,
\]

\[
\hat{\beta}_{b^*, l} \geq -\min_{m \in \mathcal{M}} \tilde{c}_{b,l,m}.
\]

We conclude for all \(l \in \mathcal{L} \setminus \mathcal{L}',\) using \(\hat{\delta}_l \geq 0, \hat{\beta}_{b,l} \geq 0:\n
\[
0 \geq \sum_{b \in \mathcal{B}} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu
\]
\[
\sum_{b \in B \setminus \{b^*\}} \hat{\beta}_{b,l} + (\hat{\beta}_{b^*,l} - \hat{\delta}_l) - \gamma + \mu \\
= \sum_{b \in B \setminus \{b^*\}} \max\{0, \hat{\beta}_{b,l}\} + \max\{0, \hat{\beta}_{b^*,l} - \hat{\delta}_l\} - \gamma + \mu \\
\geq \sum_{b \in B \setminus \{b^*\}} \max\{0, - \min_{m \in M} \bar{c}_{b,l,m}\} + \max\{0, - \min_{m \in M} \bar{c}_{b^*,l,m}\} - \gamma + \mu \\
= \sum_{b \in B} \max\{0, - \min_{m \in M} \bar{c}_{b,l,m}\} - \gamma + \mu \\
= \sum_{b \in B} \left( \min_{m \in M} \bar{c}_{b,l,m}\right)^{\bar{c}_{b^*,l,m^*}} - \gamma + \mu. 
\]

(41)

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

\[
0 \leq \hat{\delta}_l \leq \min_{b \in B, m \in M} \left( \bar{c}_{b,l,m} + \hat{\beta}_{b,l} \right) = \bar{c}_{b^*,l,m^*} + \hat{\beta}_{b^*,l},
\]

\[
\hat{\beta}_{b,l} \geq - \min_{m \in M} \bar{c}_{b,l,m}. 
\]

(42) (43)

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

\[
0 \geq \sum_{b \in B} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu \\
\geq \sum_{b \in B} \hat{\beta}_{b,l} - \bar{c}_{b^*,l,m^*} - \bar{c}_{b^*,l,m^*} - \gamma + \mu \\
= \sum_{b \in B \setminus \{b^*\}} \max\{0, \hat{\beta}_{b,l}\} - \bar{c}_{b^*,l,m^*} - \gamma + \mu \\
\geq \sum_{b \in B \setminus \{b^*\}} \max\{0, - \min_{m \in M} \bar{c}_{b,l,m}\} - \bar{c}_{b^*,l,m^*} - \gamma + \mu \\
= - \left( \bar{c}_{b^*,l,m^*} \right)^{\bar{c}_{b^*,l,m^*}} + \gamma + \mu. 
\]

(44)

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 \( m \in M \setminus M(b,l) \) form single-element promising sets of variables because for those we have \( \hat{c}_{b,l,m} = \bar{c}_{b,l,m} \), i.e., no lifting can compensate their negative uncompensated reduced costs.

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

\[
\hat{c}_l = - \sum_{b \in B, l \in L' \setminus L(b)} \left( \min_{m \in M} \bar{c}_{b,l,m}\right)^{\bar{c}_{b,l,m}} < \sum_{b \in B, l \in L(b)} \hat{\beta}_{b,l} - \gamma - \hat{\delta}_l + \mu. 
\]

(45)

The set contains all \( x_{b,l,m} \) with \( \min_{m \in M} \bar{c}_{b,l,m} < 0 \).
And finally, we have a promising set of variables from any \( l \in \mathcal{L} \setminus \mathcal{L}' \) for which
\[
\tilde{c}_l = -\sum_{b \in B} \left( \min_{m \in \mathcal{M}} \tilde{c}_{b,l,m} \right) + \left( \min_{m \in \mathcal{M}} \bar{c}_{b,l,m} \right)^+ < -\gamma + \mu. \tag{46}
\]
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 shift of dual variables (that is no lifting, by the way) that is feasible for the DMP.

The first idea is to shift the \( \alpha \)-variables of the DMP so as to compensate all the negative reduced costs of variables \( x_{b,l,m} \) with \( b \in B, l \in L(b), \) 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 \( \gamma \)-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 \( b \in B, l \in \mathcal{L} \setminus \mathcal{L}(b) \cup \mathcal{L} \setminus \mathcal{L}' \), 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 \( b \in B, l \in \mathcal{L}(b), \) and \( m \in \mathcal{M} \setminus M(b,l) \) as
\[
\bar{c}^*_b := -\left( \min_{l \in \mathcal{L}(b)} \min_{m \in \mathcal{M}(b,l)} \bar{c}_{b,l,m} \right) \leq 0. \tag{47}
\]
Next, define the maximal violation of the \( y \)-constraint in the DMP by the characteristic lifting as
\[
d^* := -\left( \min_{l \in \mathcal{L} \setminus \mathcal{L}(b) \cup \mathcal{L} \setminus \mathcal{L}'} - \sum_{b \in B} \tilde{c}_{b,l} + \delta_l + \gamma - \mu \right) \leq 0. \tag{48}
\]
Next, we define a shift of the characteristic lifting by
\[
\tilde{\alpha}_b := \alpha_b + \bar{c}_b^*, \tag{49}
\]
\[
\tilde{\beta}_{b,l} := \tilde{\beta}_{b,l}, \tag{50}
\]
\[
\tilde{\gamma} := \gamma - d^*, \tag{51}
\]
\[
\tilde{\delta}_l := \delta_l, \tag{52}
\]
\[
\tilde{\mu} := \mu, \tag{53}
\]
\[
\tilde{\phi} := \phi, \tag{54}
\]
\[
\tilde{\psi} := \psi. \tag{55}
\]
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 \( \sum_{b \in B} \bar{c}_b^* + kd^* \). In other words, if \( z^{\text{RMP}} \) is the optimal value of the RMP and \( z^{\text{MP}} \) is the optimal value of the MP, then we have
\[
z^{\text{MP}} \geq z^{\text{CSB}} := z^{\text{RMP}} + \sum_{b \in B} \bar{c}_b^* + kd^*. \tag{56}
\]

**Proof.** Since \( d^* \leq 0 \) we have that \( \tilde{\gamma} \) is non-negative. Note, that there is no non-negativity constraint for \( \alpha_b \).

We consider the \( x \)-constraints in two steps:
1. The $x$-constraints with $b \in B$, $l \in L(b)$, and $m \in M$ read:
\[
\tilde{\alpha}_b - \tilde{\beta}_{b,l} + \tilde{\delta}_l + m|l|(\tilde{\phi} - \tilde{\psi}) \leq c_{b,l,m}
\]
\[
\iff \alpha_b + \tilde{c}_b - \beta_{b,l} + \delta_l + m|l|(\phi - \psi) \leq c_{b,l,m}
\]
\[
\iff \bar{c}_{b,l,m} = c_{b,l,m} - \alpha_b + \beta_{b,l} - \delta_l - m|l|(\phi - \psi) \geq \tilde{c}_b,
\]
which holds by the minimality of $\tilde{c}_b$.

2. The $x$-constraints for all $(b, l, m)$ with $l \in L'(b) \cup L \setminus L'$ 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 $l \in \bigcap_{b \in B} L(b)$ 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 $\gamma$ relaxes the $y$-constraints, so that also the lower-bound shift satisfies these $y$-constraints.

2. The $y$-constraints with $l \in L' \setminus \bigcap_{b \in B} L(b) \cup L \setminus L'$ read:
\[
\sum_{b \in B} \tilde{\beta}_{b,l} - \tilde{\gamma} - \tilde{\delta}_l + \mu \leq 0,
\]
\[
\iff \sum_{b \in B} \tilde{\beta}_{b,l} - (\gamma - \tilde{d}^*) - \tilde{\delta}_l + \mu \leq 0,
\]
\[
\iff - \sum_{b \in B} \tilde{\beta}_{b,l} - \gamma - \tilde{\delta}_l + \mu \geq \tilde{d}^*,
\]
which holds by the minimality of $\tilde{d}^*$.

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 it is readily available in any algorithm that computes the promising sets of variables.

5 A branch-and-price algorithm for the SLDP

On smaller instances, there is evidence that the score-fix-adjust heuristic (SFA) yields close-to-optimal solutions [9]. However, it is of interest whether the performance of SFA is satisfying on larger instances, too.

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 $300 \leq |B| \leq 1600$ and $3 \leq |M| \leq 7$ while $|L|$ 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 [9] for SFA).
- An ILP solver can solve instances of the ILP model with few lot-types very fast.
- 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 ILP 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 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 \((x^*, y^*)\) 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 \((x^*, y^*)\) 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 \((x^*, y^*)\) and their cost \(z^{upper}\). Else enter the cutting phase.

4. The cutting phase: Add all lot-types \(L' \subset L\) from the RMP to the RSLDP and solve the RSLDP. If the RSLDP solution is cheaper than the current incumbent \((x^*, y^*)\), update the incumbent \((x^*, y^*)\) and the upper bound \(z^{upper}\). If \(z^{lower} \geq z^{upper}\), return \((x^*, y^*)\) and their cost \(z^{upper}\). Else strengthen the set covering constraint in the SLDP to \(\sum_{l \in L' \setminus L} y_l + s \geq 1\) yielding \(s \geq 1\) in the RMP, solve the resulting RMP, and enter the pricing phase.

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.

\[\text{In various applications with tight models, the procedure is stopped at this point because one hopes that the integrality gap is so small that further effort need not be spent (see, e.g., [13]). ASG can be seen as extending this into an exact algorithm finding the optimum ILP solution.}\]
5.2 The subroutine \textsc{GenerateLocBestLottypes}(K)

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

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

Since determining the \(K\) locally best lot-types for a branch requires a search in the complete, possibly very large lot-type set \(L\), we rather synthesize lot-types by determining the number of pieces for each size separately in a branch-and-bound algorithm named \textsc{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 \(j\)th extremal 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 \((l'_s)_{s' \in S'} \in \mathbb{N}^{S'}\) for some \(S' \subseteq S\). As for complete lot-types, the number of pieces in this partial lot-type is denoted by \(|l'| := \sum_{s' \in S'} l'_s\). A completion of a partial lot-type \(l'\) is a lot-type \(l\) with \(l_s = l'_s\) for all \(s' \in S'\). For any partial lot-type \(l'\) we consider the two extremal completions \(l'\) with \((l'_s)_{s \in S \setminus S'} := (\min_{c_s})_{s \in S \setminus S'}\) and \(\overline{l'}\) with \((\overline{l}'_s)_{s \in S \setminus S'} := (\max_{c_s})_{s \in S \setminus S'}\), 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 sizes are given by \(|l' - l'|\) and \(|\overline{l'} - l' - l'|\), respectively.

A partial lot-type \(l'\) is \textit{applicable} if \(\min_{s' \in S'} l'_s \leq \max_{s' \in S'} l'_s\) and, moreover, \(|l'| \geq \min_{s' \in S'} l'_s\) for all \(s' \in S'\). Hence, there exists an applicable completion to an applicable lot-type. For a given branch \(b\), the \textit{partial cost} of a partial lot-type \(l'\) if delivered with multiplicity \(m\) is given by \(c_{b,l',m} := \sum_{s' \in S'} (d_{b,s'} - m l'_s)\).

A lower bound for the cost \(c_{b,l,m}\) of any completion \(l\) of \(l'\) can be derived as follows: Denote by
\[
c_{b,l^*,m} := \min \{ |d_{b,s} - ml_s| : \min c_s \leq l_s \leq \max c_s \}
\]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 \(m\):
\[
c_{b,l,m} \geq \lambda_b(l', m) := c_{b,l',m} + \sum_{s \in S \setminus S'} c_{b,l^*_s,m}.
\]
A lower bound for the cost in branch \(b\) of any completion \(l\) of \(l'\) with any multiplicity is then
\[
c_{b,l,m} \geq \lambda_b(l') := \min_{m \in M} \lambda_b(l', m).
\]
Input: A consistent SLDP  
Output: An optimal solution for the SLDP  
1 \( z^{upper} \leftarrow -\infty \);  
2 \( z^{lower} \leftarrow -\infty \);  
3 \( L' \leftarrow \text{GENERATELOCBESTLOTYPES}(K) \);  
4 \( L'' \leftarrow \emptyset \);  
5 \( (\bar{x}^*, \bar{y}^*), z^{upper} ) \leftarrow \text{SCOREFIXADJUST}(L') \);  
6 \( (L', L, M) \leftarrow \text{INITRMP} \);  
7 while true do  
8   repeat  
9      // The pricing phase:  
10        \( (\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}) \leftarrow \text{SOLVERMP}(L', L, M; L'') \);  
11        \( ((L'_{new}, L_{new}, M_{new}), z^{lower}) \leftarrow \text{SOLVEPP}(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; L', z^{lower}; K) \);  
12        if \( z^{upper} \leq z^{lower} \) then  
13            return \( (\bar{x}^*, \bar{y}^*) \);  
14        else  
15            \( (L', L, M) \leftarrow (L' \cup L'_{new}, L \cup L'_{new}, M \cup M_{new}) \);  
16      until \( (L'_{new}, L_{new}, M_{new}) = (\emptyset, \emptyset, \emptyset) \);  
17   // The cutting phase:  
18        \( ((x^*, y^*), z^{upper}) \leftarrow \text{SOLVERSLDP}(L') \);  
19        if \( z^{upper} \leq z^{lower} \) then  
20            return \( (x^*, y^*) \);  
21        else  
22            \( L'' \leftarrow L' \);  
23   \}

Algorithm 1: Top Level Algorithm ASG for an SLDP

Input: A consistent SLDP and \( K \in \mathbb{N} \)  
Output: A subset of lot-types \( L' \subseteq \mathcal{L} \)  
// store lot-types (keys) with scores (values) in a table, initially empty:  
1 ScoreTable \( \leftarrow \emptyset \);  
2 for \( b \in \mathcal{B} \) do  
3    \( (l_1(b), \ldots, l_K(b)) \leftarrow \text{FINDLOCBESTLOTYPES}(b, K) \);  
4    for \( j \) from 1 to \( K \) do  
5       if ScoreTable does not yet contain \( l_j \) then  
6         insert \( l_j \) into ScoreTable with value 0;  
7     \( \text{ScoreTable}(l_j) \leftarrow \text{ScoreTable}(l_j) - 10^{K-j} \);  
8 \( L' \leftarrow \{ \text{lot-types in ScoreTable} \} \);  
9 return \( L' \);  

Algorithm 2: GENERATELOCBESTLOTYPES
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 \( |\mathcal{M}| \leq 7 \).

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. For our tests, we have used \( K = 3 \).

### 5.3 The subroutine \( \text{ScoreFixAdjust}^+ (L') \)

We briefly recall the SFA heuristics from [9]. 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, and third-best lot-types for a branch.

This scoring implies a lexicographic order on all \( k \)-subsets of lot-types. In the fix step, all \( k \)-subsets of lot-types with a non-zero score 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 non-zero-scored lot-types 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 also allowing for exchanges of lot-type-multiplicity assignments. This has the advantage that we can find feasible solutions in more instances. To obtain the scoring information in our context, we can reuse the \( \text{ScoreTable()} \) generated in Algorithm 2. An overview in pseudo code can be found in Algorithm 5.

In order to find a feasible solution for all “reasonable” instances we suggest 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 [9]. We consider ALH as part of SFA that is run at the very beginning of SFA.

ALH works as follows: Compute the average demand \( d_b := \frac{1}{|B|} \sum_{b \in B} d_b \) per branch over all branches with \( d_b := \sum_{s \in S} d_{b,s} \). 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 \( \bar{d}_c := \frac{1}{|S|} \bar{d}_c \). 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

\[
\tilde{l}_s := \max \{ |\tilde{d}_c|, \min_c \} \quad \text{and} \quad \tilde{l}_s := \min \{ |\tilde{d}_c|, \max_c \}.
\]

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

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

Now assign \( \tilde{l} \) and \( \tilde{l} \) 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
Input: A consistent SLDP, $b \in B$, $K \in \mathbb{N}$

Output: A list $\text{LotTypeList} = (l_1, \ldots, l_K)$ of $K$ locally best lot-types for branch $b$

// start with an empty sorted list:

1. $\text{LotTypeList} \leftarrow ()$;

// start with an empty partial lot-type:

2. $l' \leftarrow ()$;

// applicable range of cardinalities for the first size:

3. $n_{\text{min}} \leftarrow \min_t (|S| - 1) \max_c$;
4. $n_{\text{max}} \leftarrow \max_t (|S| - 1) \min_c$;
5. $\text{RecLocBestLotTypes}(b, K, l', n_{\text{min}}, n_{\text{max}}; \text{LotTypeList})$;
6. return $\text{LotTypeList}$;

Algorithm 3: $\text{FindLocBestLotTypes}$

---

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

Output: An updated list $\text{LotTypeList}$ of lot-types

1. if $l'$ is complete then

   // lot-type reached - check cost and update lot-type list:

   2. if $|\text{LotTypeList}| < K$ or $c_{b, l', m} < c_{b, l_K, m}$ then

      insert $l'$ into $\text{LotTypeList}$;

   else

   // list extensions of $l'$ sorted by lower bounds:

   3. CardList $\leftarrow ()$;

   4. for $n$ from $n_{\text{min}}$ to $n_{\text{max}}$ do

      5. $l'' \leftarrow (l', n)$;

      6. if $\lambda_b(l'') \leq c_{b, l_K, m}$ then

         7. insert $n$ into CardList;

      8. sort CardList by increasing $\lambda_b(l'')$;

   9. for $n$ in CardList do

      10. if $n' \leftarrow (l'', n)$;

      11. $n'_{\text{min}} \leftarrow n_{\text{min}} + (\max_c - n)$;

      12. $n'_{\text{max}} \leftarrow n_{\text{max}} - (n - n_{\text{min}})$;

      13. $\text{RecLocBestLotTypes}(b, K, l'', n'_{\text{min}}, n'_{\text{max}}; \text{LotTypeList})$;

Algorithm 4: $\text{RecLocBestLotTypes}$
the first $b - 1$ branches is $D_{b-1}$. If $n_{b-1} + |\hat{l}| + (|B| - b - 1)|\hat{\ell}| < L$, we have to assign $\hat{l}$ to branch $b$. If $n_{b-1} + |\hat{l}| + (|B| - b - 1)|\hat{\ell}| > \hat{T}$, we have to assign $\hat{l}$ to branch $b$. Otherwise, assign to branch $b$ the lot-type $\hat{l}$ if and only if $n_{b-1} + |\hat{l}| + \frac{|\hat{l}| + |\ell|}{2} > D_{b-1} + d_b$, 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 $\hat{T} - I \geq |\hat{l}| - |\ell|$ this sequential algorithm yields a feasible solution. For example, if $\hat{T} - I \geq \max_t - \min_t$ this is satisfied because, by construction, $\max_t - \min_t \geq |\hat{l}| - |\ell|$. 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

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 3 highest-scored lot-types, and for each of these 3 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 3 lot-types for $b$ and the corresponding locally optimal lot-type-multiplicities, together with the implied $y$-columns.

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 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 SolverRMP((L', L, M); \mathcal{L}'')

In this subroutine, we solve the current RMP containing $y$-variables corresponding the a subset of lot-types $\mathcal{L}'$ and $x$-variables for assignments 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.
Input: A consistent SLDP, a sorted ScoreTable of lot-types
Output: A feasible solution and its objective value \((x^*, y^*), z^{\text{upper}}\) or \((-\infty, -\infty)\)
1. \(((x^*, y^*), z^{\text{upper}}) \leftarrow (-\infty, -\infty)\);
2. If \(k > 1\) then
3. \(((x^*, y^*), z^{\text{upper}}) \leftarrow \text{AVERAGELOTTYPEHEURISTICS}(\text{SLDP});\)
4. repeat
5. for \(L\) in \(k\)-subsets of lot-types in ScoreTable do
6. for \(b \in B\) do
7. \((b, l_L(b), m_L(b)) \leftarrow (b, l^*, m^*)\); // locally optimal assignment
8. while \((b, l_L(b), m_L(b))\) violates cardinality constraint do
9. AdjTable \leftarrow \{ (b, l, m) : (b, l, m) decreases violation of \((b, l_L(n), m_L(b))\) \};
10. if AdjTable = \(\emptyset\) then
11. \(z(L) \leftarrow \infty\), break;
12. else
13. \((b^*, l^*, m^*) \leftarrow\) the marginally cheapest element in AdjTable;
14. \((b^*, l_L(b^*), m_L(b^*)) \leftarrow (b^*, l^*, m^*)\);
15. \(z(L) \leftarrow\) cost of assignment \((b, l_L(b), m_L(b))\) for \(b \in B\);
16. if \(z(L) < z^{\text{upper}}\) then
17. \(z^{\text{upper}} \leftarrow z(L)\), \((x^*, y^*) \leftarrow\) variable encoding of \((l_L(b), m_L(b))\) for \(b \in B\);
18. until time limit reached;
19. return \(((x^*, y^*), z^{\text{upper}})\);

Algorithm 5: The subroutine \text{SCOREFIXADJUST}^+.

---

Input: A consistent SLDP with \(k > 1\)
Output: A feasible solution and its objective value \(((x^*, y^*), z^{\text{upper}})\)
1. \(d_i \leftarrow\) max\(\{\text{min}_t, \text{min}_m, \frac{1}{|B|} \sum_{b \in B} \sum_{s \in S} d_{b,s}\}\);
2. \(d_c \leftarrow\) max\(\{\text{min}_t, \text{min}_m, \frac{1}{|S|} \sum_{t \in S} d_t\}\);
3. \(l \leftarrow\) \(\{d_c\}_{s \in S}\), \(\tilde{l} \leftarrow\) \(\{d_c\}_{s \in S}\);
4. while \(|l| < \text{min}_t\) do
5. \(\hat{l}_s \leftarrow l_s + 1\) for some \(s \in S\) with \(\hat{l}_s < \text{max}_c\);
6. while \(|\hat{l}| > \text{max}_t\) do
7. \(\hat{l}_s \leftarrow l_s - 1\) for some \(s \in S\) with \(\hat{l}_s > \text{max}_c\);
8. \(\hat{n} \leftarrow 0\), \(D \leftarrow 0\), \(m(b) \leftarrow 1\);
9. for \(b \in B\) do
10. if \(\hat{n} |l| + |B|- b - 1) |\hat{l}| < \hat{l}\) then
11. \(l(b) \leftarrow \hat{l}\);
12. else if \(\hat{n} |l| + |B|- b - 1) |\hat{l}| > \hat{l} + 7\) then
13. \(l(b) \leftarrow \hat{l}\);
14. else if \(\hat{n} + \frac{|l|}{2} > D + d_b\) then
15. \(l(b) \leftarrow \hat{l}\);
16. else
17. \(l(b) \leftarrow \hat{l}\);
18. \(\hat{n} \leftarrow \hat{n} + |l(b)|\), \(D \leftarrow D + d_b\);
19. \((x^*, y^*) \leftarrow\) variable encoding of \((l(b), m(b))\) for \(b \in B\), \(z^{\text{upper}} \leftarrow\) cost of \((x^*, y^*)\);
20. return \(((x^*, y^*), z^{\text{upper}})\);

Algorithm 6: The subroutine \text{AVERAGELOTTYPEHEURISTICS}.
5.6 The subroutine \textsc{SolvePP}(\(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; z^{\text{lower}}, K\))

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.

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 \(m \in M \setminus M(b,l)\). We add all such variables to the RMP and store for each branch the minimal reduced cost observed.

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

\[
\bar{c}_l = - \sum_{b \in B; \ l \in \mathcal{L}' \setminus L(b)} \left( \min_{m \in M} \bar{c}_{b,l,m} \right)^- < \sum_{b \in B; \ l \in L(b)} \beta_{b,l} - \gamma - \delta_l + \mu. \tag{67}
\]

For any \(l\) for which this happens, we add all \(x_{b,l,m}\) with \(\min_{m \in M} \bar{c}_{b,l,m} < 0\) to the RMP.

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

\[
\bar{c}_l = - \sum_{b \in B} \left( \min_{m \in M} \bar{c}_{b,l,m} \right)^- + \left( \min_{m \in M} \bar{c}_{b,l,m} \right)^+ < -\gamma + \mu. \tag{68}
\]

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 \(\bar{c}_l\) among those satisfying (68). Then, we add all corresponding \(y_l\) and all \(x_{b,l,m}\) with \(\min_{m \in M} \bar{c}_{b,l,m} < 0\) 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:

\[
\bar{c}_{b,l',m} \geq \lambda_b(l', m) := \lambda_b(l', m) - \alpha_b - m \max_{l_\gamma \neq l'} |l'_\gamma| (\phi - \psi)^+ + m \min_{l_\gamma \neq l'} |l'_\gamma| (\phi - \psi)^-. \tag{69}
\]

Together with

\[
\max_{l_\gamma = l'} |l'_\gamma| = |l'| + \min\left\{ (|S| - |S'|) \max_c \max_{l'_\gamma = l'} |l'_\gamma| \right\}, \tag{70}
\]

\[
\min_{l_\gamma = l'} |l'_\gamma| = |l'| + \max\left\{ (|S| - |S'|) \min_c \min_{l'_\gamma = l'} |l'_\gamma| \right\}, \tag{71}
\]

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 sum up. Thus, for each completion \(l\) of \(l'\) we have, using the short-hand \(m^* := m^*(b, l)\) for minimizing multiplicities:

\[
\bar{c}_l \equiv \sum_{b \in B} - (\bar{c}_{b,l,m^*})^- + (\min_{b \in B} \bar{c}_{b,l,m^*})^+ \tag{72}
\]

\[
\geq \sum_{b \in B} (\bar{\lambda}_b(l', m^*))^- + \min_{b \in B}(\bar{\lambda}_b(l', m^*))^+ \tag{73}
\]

\[
:= \bar{\lambda}(l'). \tag{74}
\]
Again, the function $\tilde{\lambda}(l', m)$ is not convex in $m$, and we implement the optimization over $m \in M$ 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 $z^{\text{lower}}$ if $z^{CSB} > z^{\text{lower}}$, i.e., the new bound is tighter. Algorithms 7 through 9 list a possible pseudo code for this. For our tests, we have used $K = 3$.

In our test, we used 

\textit{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 $l \in \mathcal{L}' \setminus \bigcup_{b \in B} L(b)$; if we find promising sets of variables there, then again we stop. If we still have not found a promising set of variables, 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 \texttt{SolveRSLDP($\mathcal{L}'$)}

This subroutine builds an RSLDP with all variables corresponding to all branches $b \in B$, lot-types $l \in \mathcal{L}'$, and multiplicities $m \in M$. This RSLDP is then solved by the black-box ILP solver. There is one speciality for our implementation: We used the original model from Section 3 without the augmenting constraints. This means, the RSLDP will reproduce earlier solutions, which does not harm. The reason for this is simply that our particular solver (\texttt{cplex}) could prove faster the optimality of an old solution in the smaller model than the infeasibility of the larger model with a cut-off at the current upper bound $z^{\text{upper}}$. Since this can be dependent on the solver software and even the version of it, we cannot make a general statement about what to prefer. It is just a guess, but it seems that current solver technology can easier prove optimality than infeasibility.

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.

The computational environment was as follows. The computer was an iMac (Retina 5K, 27 Zoll, 2019) 3 GHz Intel Core i5 with 32GB 2667 MHz DDR4 RAM running MacOSX 10.14.5 based on Darwin Kernel Version 19.0.0. We used \texttt{gcc/g++} from Apple clang version 11.0.0 (clang-1100.0.33.12). As an LP- and ILP-solver we used \texttt{IBM ILOG CPLEX 12.9.0.0}, both for the static solution and for the RMP and RSLDP during the run of ASG. Profiling was performed using the \texttt{clock()} utility from \texttt{time.h}. Timing was consistent to an accuracy of $\sim \pm 5\%$. We performed all test for zero with $\epsilon = 10^{-9}$. The accuracy of the solver was set to the same value. Because of scaling effects, we can expect the results to be accurate up to a relative error of $10^{-8}$. We used the defaults for all other solver parameters.

For ASG, we used a time limit of $\max\{1.0, \log(|\mathcal{L}|)/10.0\}$ for SFA. Moreover, we set the number of used locally best lot-types as well as the number of generated new lot-types in pricing to $K = 3$. Furthermore, we used incomplete pricing to avoid the time-consuming branch-and-bound search for promising new lot-type whenever possible.

Our tests encompass two substantially different sets of instances of the LDP. Since the SLDP can be reduced to an LDP (see Section 2), we used only LDP instances for our test to reduce the influence of too many parameters. The first set of LDPs contains 860 real-world instances from November 2011 from the daily production of our industry partner from different apparel groups. The second set of LDPs contains 5 groups of 9 random instances with identical parameters each.

In the 860 real-world instances, the numbers of applicable lot-types ranges from 9 to 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, 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.
Algorithm 7: SOLVEPP

Input: Duals/objective $\alpha, \beta, \gamma, \delta, \mu, \phi, \psi$; $z^{RMP}$ optimal for the DRMP with lot-types $\mathcal{L}'$, current lower bound $z^{lower}$; $K \in \mathbb{N}$

Output: New variables and a new lower bound $((L^{new}, L^{new}, M^{new}), z^{lower})$

1. $(L^{new}, L^{new}, M^{new}) \leftarrow \emptyset$
2. for $b \in B$ do
   3. for $l \in L(b)$ do
      4. if $\min_{m \in M} \bar{c}_{b,l,m} < 0$ then
         5. $(L^{new}, L^{new}, M^{new}) \leftarrow (b, l, m^*)$
   6. if $(L^{new}, L^{new}, M^{new}) \neq \emptyset$ then
      7. return $((L^{new}, L^{new}, M^{new}), z^{lower})$
   8. for $l \in \mathcal{L}' \setminus \bigcap_{b \in B} L(b)$ do
      9. if $\bar{c}_l < \sum_{b \in B : l \in L(b)} \bar{c}_{b,l} - \gamma - \delta_l + \mu$ then
         10. $(L^{new}, L^{new}, M^{new}) \leftarrow (L^{new}, L^{new}, M^{new}) \cup \{ (b, l, m^*) : b \in B, l \in \mathcal{L}' \setminus L(b) \}$
   11. if $(L^{new}, L^{new}, M^{new}) \neq \emptyset$ then
      12. return $((L^{new}, L^{new}, M^{new}), z^{lower})$
   13. $(L^{new}, L^{new}, M^{new}) \leftarrow (L^{new}, L^{new}, M^{new}) \cup \text{FindPromLottypes}(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; K)$
   14. if $z^{CSB} \geq z^{lower}$ then
      15. $z^{lower} \leftarrow z^{CSB}$
   16. return $((L^{new}, L^{new}, M^{new}), z^{lower})$

Algorithm 8: FINDPROMLOTYPES

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 $((L^{new}, L^{new}, M^{new})$ for the $K$ most promising new lot-types

// start with an empty sorted list:
1. $(L^{new}, L^{new}, M^{new}) \leftarrow ()$

// start with an empty partial lot-type:
2. $l' \leftarrow ()$

// applicable range of cardinalities for the first size:
3. $n_{\min} \leftarrow \min_t (|S| - 1) \max_t$
4. $n_{\max} \leftarrow \max_t (|S| - 1) \min_t$
5. $\text{RecPromLottypes}(\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}; \mathcal{L}'; K; l', n_{\min}, n_{\max}; (L^{new}, L^{new}, M^{new}))$
6. return $(L^{new}, L^{new}, M^{new})$
In order to save time, we solved the static model 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.

In the $5 \times 9$ 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.

Figure 1: Comparison of CPU times in seconds on 860 real-world instances ordered by the numbers of applicable lot-types (indicated on the category axis)

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. Figure 1 shows the CPU times for the static solution and the ASG solution. The instances have been ordered from left to right by their numbers of applicable lot-types. Some of these numbers have been indicated on the category axis. 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 solving. 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.
Input: $\alpha, \beta, \gamma, \delta, \mu, \phi, \psi; z^{RMP}$ optimal for the DMP with lot-types $L', K \in \mathbb{N}$, a partial lot-type $l'$, $n_{min}, n_{max} \in \mathbb{N}$, a set $(L^{new}, L^{new}, M^{new})$ of columns corresponding to lot-types found so far with $L' = \langle l_1, \ldots, l_K \rangle$ sorted by sum of negative reduced costs

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

Algorithm 9: RecPromLottypes

| $|B|$ | $|S|$ | $|M|$ | 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     |

| $|L|$ | $k$ | #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 | 3,965,169,055,024 | 790,801,904,973 |

Table 1: The parameters of the $5 \times 9$ instances.
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 1000 columns to find and prove the optimum, whereas the static solution, 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 the static solution would have used if it had enough memory available, we did not include these numbers in the chart because the computation did not take place.

In order to characterize the difficult instances, we evaluated the relative integrality gap and the relative gap of the SFA heuristic in Figure 3. The excellent performance really depends on the small integrality gap of the model. However, making a tight model is the whole point of using many variables in ILP models. We see once more that the model tightness is much more important than the numbers of columns when it comes to ILP solving – the column generation is very effective.
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 $5 \times 9$ random instances.

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 solving and can solve all instances. The instances from the largest group, however, this time take much longer. 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 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, all the imbalance is just noise. If real-world instances

![Figure 4: The relative CPU usage for ASG in percent on 860 real-world instances ordered by the numbers of applicable lot-types (indicated on the category axis)](image)

![Figure 5: Comparison of CPU times in seconds on $5 \times 9 = 45$ random instances ordered by the numbers of applicable lot-types (indicated on the category axis)](image)
could be expected to be like the random instances, then probably the lot-type based process would not be
used in practice in the first place. In a sense, the random instances constitute a stress test for ASG, and
we find that the performance compared to static solving is still impressive.

The reason for the performance difference between ASG and static solving again correlates with the
total number of columns used during the solution process. Figure 6 shows that ASG gets away (even in the
random instances) with two to three orders of magnitude fewer columns. 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.

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, though the integrality gap is small everywhere by any practically relevant measure.

Figure 6: Comparison of columns needed in seconds on $5 \times 9 = 45$ random instances ordered by the numbers
of applicable lot-types (indicated on the category axis)

Figure 7: Gaps for the $5 \times 9 = 45$ random instances ordered by the numbers of applicable lot-types
(indicated on the category axis)
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 (indicated 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.

Summarized, we see that ASG is substantially faster and more memory-efficient than static solving, 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 of 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 faster than static ILP-solving. The pricing routine in ASG takes profit of 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 large effort for the small gap to close 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. A bonus of ASG is that all progress in commercial ILP-solver technology is exploited.

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

[3] 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.
[4] D. P. Bertsekas. Dynamic Programming & Optimal Control, Vol. I. Athena Scientific, 3rd edition, 2005.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] M. E. Lübbecke and J. Desrosiers. Branch-Price-and-Cut Algorithms. John Wiley & Sons, Inc., 2011.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] A.-K. Rothenbächer, M. Drexel, 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.
In the special case of a parameterizable set of lot-types we can model the SLDP with fewer variables:

\[
\begin{align*}
\min & \quad \sum_{b \in B} \sum_{s \in S} \sum_{a \in A} \delta_{b,s}^a \\
\text{s.t.} & \quad \sum_{i \in K} \sum_{m \in M} x_{b,i,m} = 1 \quad \forall b \in B \\
& \quad v_{b,s,i,m} - \max_c \cdot x_{b,i,m} \leq 0 \quad \forall (b, s, m, i) \in U \\
& \quad v_{b,s,i,m} - l_{i,s} \leq 0 \quad \forall (b, s, m, i) \in U \\
& \quad v_{b,s,i,m} - \max_c \cdot x_{b,i,m} \geq -\max_c \quad \forall (b, s, m, i) \in U \\
& \quad L \leq \sum_{b \in B} \sum_{s \in S} \sum_{m \in M} \sum_{i \in K} m \cdot v_{b,s,i,m} \leq I \\
& \quad l_{i,s} \leq \max_c \quad \forall i \in K, s \in S \\
& \quad l_{i,s} \geq \min_c \quad \forall i \in K, s \in S \\
& \quad \sum_{s \in S} l_{i,s} \leq \max_t \quad i \in K \\
& \quad \sum_{s \in S} l_{i,s} \geq \min_t \quad \forall i \in K \\
& \quad \delta_{b,s}^a + \sum_{i \in K} \sum_{m \in M} m \cdot v_{b,s,i,m} \geq d_{b,s}^a \quad \forall b \in B, s \in S, a \in A \\
& \quad \delta_{b,s}^a - \sum_{i \in K} \sum_{m \in M} m \cdot v_{b,s,i,m} \geq -d_{b,s}^a \quad \forall b \in B, s \in S, a \in A \\
& \quad x_{b,i,m} \in \{0, 1\} \quad \forall b \in B, i \in K, m \in M \\
& \quad v_{b,s,i,m} \geq 0 \quad \forall (b, s, m, i) \in U \\
& \quad l_{i,s} \in \mathbb{Z} \quad \forall i \in K, s \in S \\
& \quad \delta_{b,s}^a \geq 0 \quad \forall b \in B, s \in S, a \in A,
\end{align*}
\]

where we use the abbreviations \( K := \{1, \ldots, k\} \) and \( U := B \times S \times M \times K \). 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 lot-type \( i \) 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 $\delta_{b,s}^a$ then are given by

$$\delta_{b,s}^a = \left| d_{b,s}^a - \sum_{i=1}^k \sum_{m \in M} m \cdot v_{b,s,i,m} \right|.$$ 

Again the linearization of this non-linear equation is standard.\footnote{We remark that one can express the deviations $\delta_{b,s}^a$ 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 $\delta_{b,s}^a$ by $\delta_{b,s}^a + m \cdot l_{i,s} - d_{b,s}^a \cdot x_{b,i,m} \geq 0$ and $\delta_{b,s}^a - m \cdot l_{i,s} + m \cdot max_c \cdot (1 - x_{b,i,m}) \geq -d_{b,s}^a$ for all $(b,s,m,i) \in \mathcal{I}, a \in \mathcal{A}$. Or we may use both sets of inequalities. It turns out that this alternative formulations is solved slightly faster on some inspected instances.}

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 $(l_{i,s})_{s \in S}$ are lexicographically ordered. For the ease of notation we assume that the set of sizes $S$ is given by $\{1, \ldots, s\}$. As an abbreviation we set $t := \max_c - \min_c + 1$. With this the additional inequalities

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

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.\footnote{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 $\min_c \leq d_{b,s}^a \leq \max_c \cdot \max \{m \in M\}$}

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.