Lexicographic Bin-Packing Optimization for Loading Trucks in a Direct-Shipping System

Katrin Heßler[*,a], Stefan Irnich[a], Tobias Kreiter[b,c], Ulrich Pferschy[c]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany*
[b] *scc EDV-Beratung AG, Wambachergasse 10, 1130 Wien, Austria*
[c] *University of Graz, Department of Statistics and Operations Research, Universitaetsstrasse 15/E3, 8010 Graz, Austria*

Abstract

We consider a packing problem that arises in a direct-shipping system in the food and beverage industry: Trucks are the containers and products to be distributed are the items. The packing is constrained by two independent quantities, weight (e.g., measured in kg) and volume (number of pallets). Additionally, the products are grouped into the three categories standard, cooled, and frozen (the latter two require refrigerated trucks). Products of different categories can be transported in one truck using separated zones, but the cost of a truck depends on the transported product categories. Moreover, product splitting should be avoided so that (un-)loading is simplified. As a result, we seek for a feasible packing optimizing the following objective functions in a strictly lexicographic sense: minimize the (1) total number of trucks; (2) number of refrigerated trucks; (3) number of refrigerated trucks which contain frozen products; (4) number of refrigerated trucks which also transport standard products; (5) and minimize product splitting. This is a real-world application of a bin-packing problem with cardinality constraints a.k.a. the two-dimensional vector packing problem, with additional constraints. We provide a heuristic and an exact solution approach. The heuristic meta-scheme considers the multi-compartment and item-fragmentation features of the problem and applies various problem-specific heuristics. The exact solution algorithm covering all five stages is based on branch-and-price using stabilization techniques exploiting dual-optimal inequalities. Computational results on real-world and difficult self-generated instances prove the applicability of our approach.

Key words: bin packing, lexicographic objective, heuristics, column generation, dual-optimal inequalities

1. Introduction

In this paper, we present a system of bin-packing problems that arise in a direct-shipping system in the food and beverage industry. More than 1.2 million units of different products leave the factory of our industry partner every day. By far the largest share of the products is transported by trucks from the factory to the distribution centers of supermarket chains (in the following denoted as warehouses). As shipping is done directly from the factory to individual warehouses with full-truck volumes, the routing of trucks plays no role. For each warehouse, the use of the utilized trucks has to be optimized by assigning shipments to trucks. Hence, the overall problem naturally decomposes by warehouse.

The transportation of products is standardized and uses euro-pallets (which are typically packed with a set of uniform boxes of a product). In our application, quantities are large so that pallets are not mixed, i.e., each pallet contains only one product. Moreover, it is legitimate to assume that products are equally distributed on pallets such that each pallet loaded with a certain product has the same weight.

*Corresponding author.
Email addresses: `khessler@uni-mainz.de` (Katrin Heßler), `irnich@uni-mainz.de` (Stefan Irnich), `tobias.kreiter@scc.at` (Tobias Kreiter), `pferschy@uni-graz.at` (Ulrich Pferschy)

Let $K = \{1, 2, \ldots, m\}$ denote the set of products (to be shipped to a particular warehouse on a specific day). As the range of food products includes a huge variety of commodities, e.g., fruit juices, spirits, baked goods, jams, wafers, and ketchup, the weight and space requirements for the transport differ substantially by product. For each product $k \in K$, these requirements are therefore described by two *attributes*:

- the *unit weight* ρ_k (in *kg/pallet*) of a pallet loaded with product k and
- the *number* n_k of pallets that need to be shipped.

In addition, all products can be uniquely grouped into three *categories*:

- *Standard products* I^S require no cooling;
- *Cooled products* I^C require cooling;
- Shipping of *frozen products* I^F requires deep cooling and is the most costly transport.

As a result, the set of products is partitioned into $I = I^S \cup I^C \cup I^F$.

The fleet consists of *standard trucks* and *refrigerated trucks*, i.e., trucks without and with a cooling system. Frozen and cooled products must be transported in refrigerated trucks, which are more costly than standard trucks. Frozen and cooled products can be mixed (using different zones), but costs are higher if frozen products are transported. Also, standard products can be transported by refrigerated trucks (again using separated zones). There are no conflicts between products, i.e., any subset of products can be packed together into a truck. All trucks are homogeneous regarding capacity. Let W be the *capacity* (in *kg*) for the physical weight and B be the *capacity* for the number of pallets that fit into one truck (usually $B = 33$ *pallets*).

Moreover, depending on the customer or warehouse, two *policies* are applied regarding the *splitting* of pallets of the same product:

forbidden: Splitting is forbidden so that all n_k pallets loaded with product $k \in K$ must be transported by the same truck. This is the standard case because it simplifies the handling and processing in the loading and unloading phases.

allowed: Splitting is allowed so that the n_k pallets of a product $k \in K$ may be distributed arbitrarily over the trucks. This splitting allowed policy can help to reduce the total number of used trucks. However, distributing a product over several trucks causes inconveniences. It should not occur more often than necessary (see objective 5 below).

For a given solution, we denote a product assigned to two or more trucks as *split product*.

We want to present both policies, splitting forbidden and splitting allowed, as variants of the same packing problem. For this purpose, we introduce *items* which are the atomic, not-divisible objects in the bin-packing model. The set K of products and the set I of items are identical, i.e., $K = I$. However, we write $k \in K$ to refer to products and $i \in I$ to refer to items. Depending on the policy, we define two weights for each item: w_i refers to the physical weight and b_i to the volume expressed as number of pallets. Table 1 and Example 1 clarify the correspondence.

Product $k \in K$	Corresponding item $i \in I$		
	Policy	splitting forbidden	splitting allowed
Unit weight ρ_k	Weight 1	$w_i = \rho_k \cdot n_k$	$w_i = \rho_k$
Number of pallets n_k	Weight 2	$b_i = n_k$	$b_i = 1$
	Demand	$q_i = 1$	$q_i = n_k$

Table 1: Relationship between products and items for the policies splitting forbidden and splitting allowed.

Example 1. *Consider a set of products consisting of three standard products ($n_1 = 5, n_2 = 5, n_3 = 3, \rho_1 = \rho_2 = \rho_3 = 1$), (gray in Figure 1), one cooled product ($n_4 = 2, \rho_4 = 2$) (green in Figure 1), and one frozen product ($n_5 = 3, \rho_5 = 3$) (blue in Figure 1), i.e., $K = I = \{1, \ldots, 5\}$ with $I^S = \{1, 2, 3\}, I^C = \{4\}, I^F = \{5\}$. Then, the corresponding instance of the splitting forbidden policy has parameters $\boldsymbol{w} = (5, 5, 3, 4, 9), \boldsymbol{b} = (5, 5, 3, 2, 3), \boldsymbol{q} = \mathbb{1}$, whereas the corresponding instance of the splitting allowed policy has parameters $\boldsymbol{w} = (1, 1, 1, 2, 3), \boldsymbol{b} = \mathbb{1}, \boldsymbol{q} = (5, 5, 3, 2, 3)$.*

With these definitions of items $i \in I$ with two weights $(w_i, b_i)_{i \in I}$ and two capacities (W, B), the problem of assigning products to trucks becomes a two-dimensional bin/vector-packing problem (see Section 2 for references to the pertinent literature). Trivially, a feasible packing exists only if $w_i \leq W$ and $b_i \leq B$ holds for all items $i \in I$.

What also makes our real-world problem interesting is the non-standard objective. We are seeking for a feasible product-truck assignment optimizing the following five objectives in a strictly *lexicographic* sense:

1. minimize the total number of trucks;
2. minimize the number of refrigerated trucks;
3. minimize the number of refrigerated trucks transporting frozen products;
4. minimize the number of refrigerated trucks which also transport standard products; and
5. minimize product splittings, which is either
   5a. minimize the overall number of splits; or
   5b. minimize the number of products which are split (over any number of trucks).

The fifth objective is only relevant for the policy splitting allowed. Altogether, we show that this lexicographic objective can be optimized by solving a sequence of extended bin-packing problems. Note that it is not a-priori given how to measure the impact of splitting products. The corresponding objective will depend on the actual handling of split products after their delivery at the destination warehouse. According to our industry partner, there are two different strategies observed with its customers: One customer handles products separately for every truck. This means that for every truck the processing effort depends on the number of different products (but also on other factors, such as the number of pallets). Therefore, it is cost efficient to minimize the overall number of splits, i.e., packing one product separated into two trucks is preferred over packing it separated into three trucks, as stated in objective 5a.A second customer collects all pallets carrying the same product, but arriving on different trucks, in the unloading zone and then inserts them into the warehouse in one batch. Thus, the main effort of splitting is due to the necessity of reserving a collection area, which happens whenever a product is split. Whether pallets are collected from two or more trucks hardly plays a role, as stated in objective 5b.

Note that for both splitting policies, objectives 1.–4. have to be observed in a lexicographic sense. This means that, e.g., a solution with 20 trucks and many refrigerated trucks is preferred over a solution with 21 trucks and less refrigerated trucks. An example of three different optimal solutions (splitting forbidden, splitting allowed and minimize either the number of splits or the number of split products) is given in Figure 1.

The methodological contribution of our paper is the development of an exact and heuristic solution approach. The exact approach is based on *branch-and-price* (BaP) (Desaulniers *et al.*, 2005; Lübbecke and Desrosiers, 2005). For the different objectives and corresponding extended bin-packing problems, we develop a unified description of the respective column-generation subproblems. We show that, in turn, each subproblem can be solved by one or several modified 2-dimensional knapsack problems (Martello and Toth, 2003), (Kellerer *et al.*, 2004, ch. 9.6). These are especially involved in the case of the policy splitting allowed. Moreover, an important and here non-trivial algorithmic component is the branching scheme. We adopt the branching scheme originally developed and successfully applied by Heßler *et al.* (2018) to the vector-packing problem. Stabilization with dual-optimal inequalities (Ben Amor *et al.*, 2006; Gschwind and Irnich, 2016) helps to increase the performance of the BaP approach.

The heuristic approach works in two levels: In the first level, several constructive heuristics are employed to reach a packing of items from the same category. They strive for a balanced utilization of weight and volume constraints based on weight-per-pallet, i.e., *density* considerations and on the surrogate weights as introduced by Caprara and Toth (2001). In the second level, a more complicated meta-scheme considers the multi-compartment and product-fragmentation features of the problem at hand and applies various problem-specific heuristics to search for good solutions. They are based on decomposing and regrouping the best solution obtained from the first level to improve the lexicographic objectives for the different product categories. At first, only the splitting forbidden policy is considered. Then, for the splitting allowed policy we try to improve the given solution by separating certain pallets of the same product and possibly reduce the number of trucks by reassigning these items. Throughout the heuristic process, we compare the current solutions to lower bounds as a stopping criterion. It should be noted that the heuristics were also

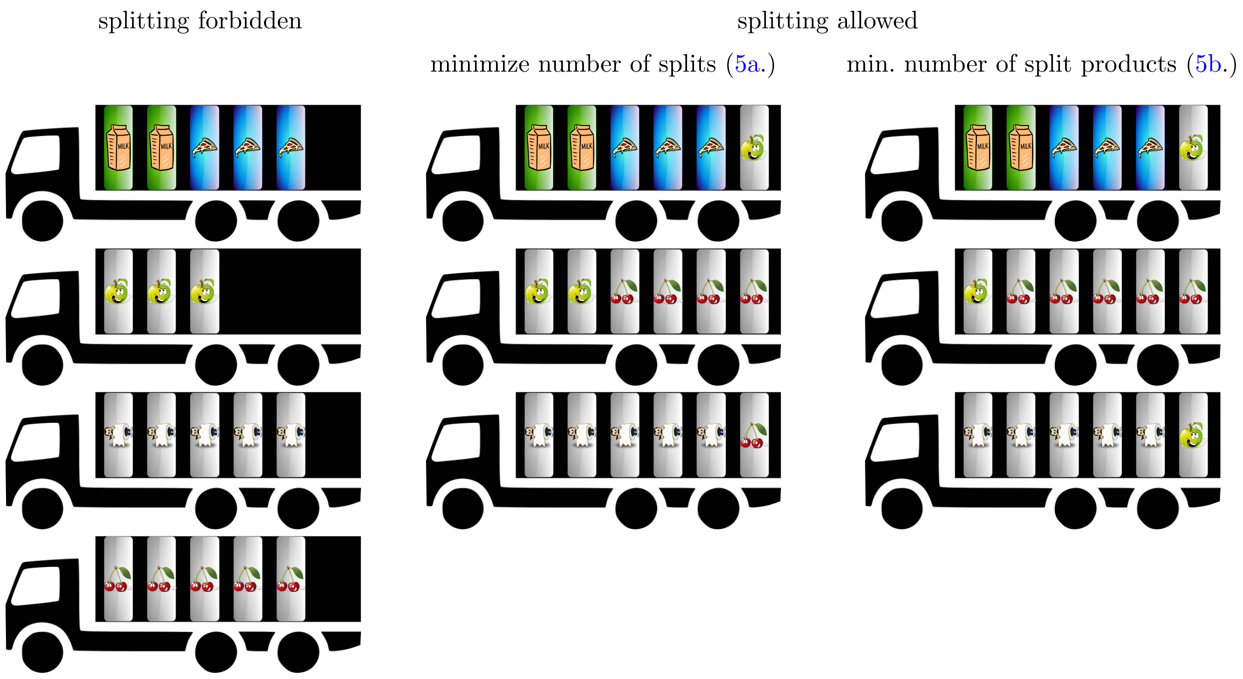
splitting forbidden
splitting allowed
minimize number of splits (5a.)
min. number of split products (5b.)

Figure 1: Optimal solutions of three objectives for Example 1 with capacity $B = 6$ pallets.

incorporated in the SAP ERP software system of our industrial partner, a European producer of foodstuffs and beverages, and are used in the daily planning process.

We conducted a series of computational experiments with the exact and heuristic algorithms based on real-world data sets with up to 430 items and on more difficult test instances derived from these. We compare the performance of our heuristic to the previously employed strategy of the company and to the proven optimal solutions. It turns out that the heuristic saves on average one truck per day compared to the previously used approach for real-world instances, with an even higher advantage for more difficult instances. At the same time, running times are moderate, often less than one second. The gap to the optimal solution is fairly moderate for real-world instances but increases for the self-generated hard test cases. Regarding the performance of the exact branch-and-price algorithm, most of the test instances can be solved to optimality within the time limit of 10 minutes for every subproblem. Notably exceptions arise for some of the generated, difficult instances in case of the splitting allowed policy. We can also illustrate the highly positive effect of strengthening the ILP-models by lower bounds and of solving pricing problems on a reduced graph. Finally, combining the two approaches it turns out that adding the best heuristic solution to the set of columns of the branch-and-price approach has hardly any effect for the splitting forbidden policy, but it gives a major improvement for the splitting allowed policy.

The remainder of the paper is organized as follows: In Section 2, a brief overview of the literature on two-dimensional bin-packing problems with and without item fragmentation and multi-compartment vehicle-routing problems is given. Analytically computed lower bounds are presented in Section 3. Details on the exact BaP algorithm are provided in Section 4, including the sequence of extensive formulations used for the different objectives, the unified column-generation algorithm, and stabilization techniques. The heuristic approach is presented in Section 5. In Section 6, the results of our computational experiments are presented and discussed. Final conclusions are drawn in Section 7.

2. Literature Review

The classical one-dimensional *bin-packing problem* (BP) is one of the most fundamental problems in combinatorial optimization: A given set of items has to be packed into a minimum number of bins such that the total weight of the items in each bin does not exceed the bin-capacity. Its mathematical foundations were first studied in the early 1970s (Garey *et al.*, 1972). Classical heuristics for BP, such as *first-fit* (FF), *best-fit* (BF) and *worst-fit* (WF), and their variants based on a decreasing ordering of item weights, i.e., FFD, BFD, and WFD (see, e.g., Johnson, 1973), will be assumed common knowledge throughout this paper.

For the exact solution of BP, two alternative and competing solution principles can be observed in the recent literature (see Delorme *et al.*, 2016, for an overview of BP formulations). The first type of solution approach relies on pseudo-polynomial arc-flow formulations (Valério de Carvalho, 1999) solved with general-purpose *mixed-integer linear programming* (MIP) solvers. Important refinements are network/graph compression techniques (Brandão and Pedroso, 2016) and the use of *reflect networks* modeling packings with only half of the bin capacity (Côté and Iori, 2018). The first half and the second half of a packed bin are represented and feasibly combined in the reflect network. The second type of solution approach uses a pattern-based formulation a.k.a. the Gilmore-Gomory model (Gilmore and Gomory, 1961), which is solved by column-generation techniques. While early works focused on the solution of the linear relaxation in order to quickly compute a strong lower bound, the necessary theory providing a complete branching scheme leading to a fully-fledged BaP algorithm has been provided with the work of Vanderbeck (1999). A state-of-the-art branch-price-and-cut approach for BP is presented by Wei *et al.* (2019). A hybrid of both approaches, denoted as *reflect+*, has been recently coined by Delorme and Iori (2020) and seems to be a very competitive approach making use of the excellent dual bounds provided by column generation, heuristically and exactly truncated reflect networks, and the power of modern MIP solvers.

Due to the high practical relevance of packing problems many variations of the classical one-dimensional BP have been investigated. For a comprehensive overview, we refer to the surveys by Christensen *et al.* (2017) and Coffman Jr *et al.* (2013). In the context of our real-world bin packing variant especially publications dealing with two-dimensional, multi-compartment, and item fragmentation aspects are of particular relevance.

Two-dimensional BPs represent a well-studied generalization and, hence, a large number of publications in this field appeared over the last decades. We do *not* consider packing problems in which geometric items have to be placed into bins, known as rectangle packing (Lodi *et al.*, 2002). Relevant for our application are p-dimensional BPs with $p = 2$ independent dimensions (such as physical weight and volume, or value and weight). In the literature these problems are given different names, e.g., (2-dimensional) *vector (bin) packing problem* (VPP, 2D-VPP) and *two-constraint bin packing problem.* Our problem is a variant of the 2D-VPP for which an early exact approach is given by Spieksma (1994). Recent exact approaches are based on BaP and are described in Heßler *et al.* (2018) and Wei *et al.* (2020). General approximation results are given in Bansal *et al.* (2016). The case, where weights and volume are strictly ordered is considered in Caprara *et al.* (2003). Simple greedy-type heuristics are presented in Aringhieri *et al.* (2018). A comprehensive survey of vector packing problems can be found in Christensen *et al.* (2017).

Multi-compartment problems arise when different item types can be packed into the same bin but have to be separated from each other (incompatibility constraints). Those settings are mainly studied in vehicle-routing problems, see the surveys (Pollaris *et al.*, 2014; Henke, 2018). In our application, there are no incompatibilities between products/items. However, the use of compartments for cooled and frozen products imposes additional costs.

The introduction of *item fragmentation* generalizes the classical BP by allowing items to be split among several bins at a cost. Item fragmentation for BPs is introduced and proved NP-hard by Mandal *et al.* (1998). More recent publications provide exact solution algorithms based on column generation and dual cuts techniques as well as employing heuristics and implicit enumeration (Casazza and Ceselli, 2016). LeCun *et al.* (2015) presents models for practical applications of item fragmentation like minimizing the number of money transfers after a group trip and present approximation algorithms for identical as well as for the more generic case of non-equal-sized bins. The generalization to bins that differ in both cost and capacity is tackled in the contribution Casazza (2019) by proposing new mathematical programming models that even avoid the use of fractional variables. A worst-case analysis for BP with item fragmentation is performed by Bertazzi *et al.* (2019). General models of valuations for fragmented items are considered for the closely related knapsack problem in the recent work Malaguti *et al.* (2019). Recall from Section 1 that in our application, products might be divisible (depending on the splitting policy) but that by definition our items are never split. Our definition of an item has implicitly solved issues related to product fragmentation by discretely splitting demand along the volume (=pallets) dimension.

3. Lower bounds

Various lower bounds for subsets of items or all items are used in different parts of the algorithms. For convenience, we introduce these lower bounds in this brief section.

A straightforward lower bound on the number of bins needed to pack a subset $\mathcal{I} \subseteq I$ can be obtained by dividing the problem along the two dimensions. For each dimension one can compute a bound on the respective one-dimensional bin packing problems and then use the better bound of the two:

$$\mathrm{LB}_1(\mathcal{I}) = \max\left\{ \left\lceil \frac{\sum_{i\in\mathcal{I}} w_i}{W} \right\rceil, \left\lceil \frac{\sum_{i\in\mathcal{I}} b_i}{B} \right\rceil \right\}.$$

Caprara (1998) proved that this lower bound equals the rounded up value of the linear relaxation to the standard assigment-type ILP model for the two-dimensional bin-packing problem, see e.g. (1)–(6) in Caprara and Toth (2001).

Another lower bound is inspired by the lower bounding procedure of Martello and Toth (1990) for bin-packing problems. We use two distinct subsets of $\mathcal{I}$, namely the subset

$$\mathcal{I}_1 = \{i \in \mathcal{I} : w_i > W/2 \text{ and } b_i > B/2\}$$

of items consuming more than half of a bin's capacity, and the disjoint subset

$$\mathcal{I}_2 = \{i \in \mathcal{I} \setminus \mathcal{I}_1 : w_i \geq W/2 \text{ or } b_i \geq B/2\}$$

of items that do not fit together into a bin with any item of the subset $\mathcal{I}_1$. We obtain the lower bound

$$\mathrm{LB}_2(\mathcal{I}) = |\mathcal{I}_1| + \mathrm{LB}_1(\mathcal{I}_2).$$

In the following, the lower bound $\mathrm{LB}(\mathcal{I}) = \max\{LB_1(\mathcal{I}), LB_2(\mathcal{I})\}$ is used for various subsets $\mathcal{I}$. For the sake of brevity, we define the following shorthand notation for the lower bounds

$$\mathrm{LB} = \mathrm{LB}(I), \qquad \mathrm{LB}^S = \mathrm{LB}(I^S), \qquad \mathrm{LB}^F = \mathrm{LB}(I^F), \qquad \text{and} \quad \mathrm{LB}^{C,F} = \mathrm{LB}(I^C \cup I^F).$$

4. Branch-and-price algorithm

We propose the exact solution of the overall lexicographic minimization problem using a stage-wise modified BaP algorithm (Desaulniers *et al.*, 2005; Lübbecke and Desrosiers, 2005). While the first objective is minimized by solving a standard 2D-VPP, the following four stages first restrict the solution space to previously computed optimal value(s) and then apply BaP to the restricted formulation. Since the initial formulation is of the Gilmore-Gomory type (Gilmore and Gomory, 1961; Heßler *et al.*, 2018), the linear programs can be characterized as *extensive*, meaning that they typically exhibit an enormous number of variables. BaP is tailored to solving such extensive optimization problems. Technically, BaP is a branch-and-bound algorithm in which the linear relaxation at each node of the search tree is solved by *column generation* (CG). CG is an iterative method that solves a linear program by decomposing it into a *restricted master problem* (RMP) and a pricing *subproblem* (SP).

In Section 4.1, we start by describing the set of all feasible packing patterns and define some subsets of patterns which are later used for presenting the restricted formulations that address the objectives 2.–4., 5a., and 5b. at subsequent stages. The unifying approach for the pricing subproblems presented in Section 4.2 is followed by the discussion of stabilization, branching schemes, and acceleration techniques in Sections 4.3–4.5.

4.1. Pattern-based formulations

The models of all stages are variations of the covering formulation of Gilmore and Gomory (1961) that are based on *patterns*. A pattern describes a feasible packing of a bin with items. For each item $i \in I$ the value

$$u_i = \min \left\{ q_i, \left\lfloor \frac{W}{w_i} \right\rfloor, \left\lfloor \frac{B}{b_i} \right\rfloor \right\}$$

is an upper bound on the number of times that item i may occur in a pattern. The set of all feasible patterns is therefore

$$P = \left\{ (a_1, \dots, a_m)^\top \in \mathbb{Z}_+^m : \sum_{i=1}^{m} a_i w_i \le W, \sum_{i=1}^{m} a_i b_i \le B \text{ and } a_i \le u_i \text{ for all } i \in I \right\}.$$

Recall that for the standard case of the splitting forbidden policy, we have $q_i = 1$ and thus $u_i = 1$ so that all feasible patterns are binary, i.e., $(a_1, \dots, a_m) \in \{0,1\}^m$. Thus, the resulting VPP is a *binary VPP* (01-VPP). For the splitting allowed policy, the coefficients a_i are upper bounded by integers u_i so that the resulting VPP is a *bounded VPP* (B-VPP).

For Stages 2.–5., pattern subsets taking different product types into account have to be considered. We use the superscripts S, C, and F to refer to standard, cooled, and frozen products/items, respectively. A combination of superscripts refers to patterns that only include items of the respective categories. If a superscript is supplemented with > 0, at least one item of the category must be included. Table 2 exactly defines the relevant pattern subsets.

	Specification regarding categories of items				
Pattern subset	standard $\sum_{i \in I^S} a_i$		cooled $\sum_{i \in I^C} a_i$		frozen $\sum_{i \in I^F} a_i$
P^S			$= 0$	and	$= 0$
P^C	$= 0$				$= 0$
P^F	$= 0$	and	$= 0$		
$P^{C,F}$	$= 0$				
$P^{S,C>0}$			≥ 1	and	$= 0$
$P^{C,F>0}$	$= 0$			and	≥ 1
$P^{S>0,C>0}$	≥ 1	and	≥ 1	and	$= 0$
$P^{S,C,F>0}$					≥ 1
$P^{S>0,C,F>0}$	≥ 1			and	≥ 1
$P^{S,C \cup F>0}$			$\sum_{i \in I^C} a_i + \sum_{i \in I^F} a_i \ge 1$		
$P^{S>0,C \cup F>0}$	≥ 1	and	$\sum_{i \in I^C} a_i + \sum_{i \in I^F} a_i \ge 1$		

Table 2: Overview of pattern subsets.

Stage 1.: Minimize the total number of trucks

At Stage 1., the minimization of the number of trucks is equivalent to the minimization of the total number of bins that are used. Therefore, the problem is a standard 2D-VPP. The Gilmore-Gomory formulation comprises non-negative integer variables x_p for all patterns $p \in P$ and is given by

$$\begin{aligned} z^{1.} = \quad & \min \sum_{p \in P} x_p & & \text{duals:} & \text{(1a)} \\ \text{(Stage 1.)} \quad & \text{subject to } \sum_{p \in P} a_i^p x_p \ge q_i, & i \in I \quad & [\pi_i] & \text{(1b)} \\ & x_p \ge 0 \text{ integer}, & p \in P. \quad & & \text{(1c)} \end{aligned}$$

The objective (1a) minimizes the number of bins, i.e., patterns used. Constraints (1b) ensure that the demand of all items is covered. Note that for an RMP, i.e., the linear relaxation of a model with a restricted variable/pattern set $P' \subset P$, we also show the associated dual variables $(\pi_i)_{i\in I}$ of constraints (1b) that we later use in Section 4.2 to describe the CG process. The domain of the pattern variables is defined by (1c). The first stage can be exactly solved with the BaP algorithm presented by Heßler *et al.* (2018) without any adaptation.

Stage 2.: Minimize the number of refrigerated trucks

The secondary objective of minimizing the number of refrigerated trucks is equivalent to minimizing the number of patterns of the subset $P^{S,C\cup F>0}$ (see Table 2). The first objective imposes the additional condition that the total number of all trucks/bins is restricted to $z^{1\cdot}$. It can be incorporated into model (1) with a single additional constraint, leading to:

$$
\begin{array}{llr}
z^{2\cdot} = & \min \sum_{p\in P^{S,C\cup F>0}} x_p & \text{duals:} \quad (2a) \\
\text{(Stage 2.)} & \text{subject to (1b)–(1c)} & \\
& \sum_{p\in P} x_p \le z^{1\cdot} & [\delta_{2b}] \quad (2b) \\
& \sum_{p\in P^{S,C\cup F>0}} x_p \ge \mathrm{LB}^{C,F}. & [\delta_{2c}] \quad (2c)
\end{array}
$$

The objective (2a) minimizes the number of refrigerated trucks. Constraint (2b) restricts the total number of trucks to the minimum number resulting from the solution of formulation (1). A lower bound on the number of refrigerated trucks is imposed by constraint (2c). Note that constraint (2c) is not mandatory for the correctness of the model but its addition often yields a better lower bound of the linear relaxation.

Stage 3.: Minimize the number of refrigerated trucks which contain frozen products

The tertiary objective of minimizing the number of refrigerated trucks that contain frozen products is equivalent to minimizing the number of patterns of the subset $P^{S,C,F>0}$ (see Table 2). At Stage 3., we additionally have to bound the number of refrigerated trucks by $z^{2\cdot}$, which yields to the model

$$
\begin{array}{llr}
z^{3\cdot} = & \min \sum_{p\in P^{S,C,F>0}} x_p & \text{duals:} \quad (3a) \\
\text{(Stage 3.)} & \text{subject to (1b)–(1c), (2b)} & \\
& \sum_{p\in P^{S,C\cup F>0}} x_p \le z^{2\cdot} & [\delta_{3b}] \quad (3b) \\
& \sum_{p\in P^{S,C,F>0}} x_p \ge \mathrm{LB}^{F}. & [\delta_{3c}] \quad (3c)
\end{array}
$$

The objective (3a) is the minimization of the number of trucks transporting frozen products. Constraint (3b) fixes the total number of refrigerated trucks. Since this fixation makes constraint (2c) redundant, it is omitted. We add the constraint (3c) imposing a lower bound on the number of frozen trucks to strengthen the linear relaxation.

Stage 4.: Minimize the number of refrigerated trucks which also contain standard products

The fourth rate objective of minimizing the number of refrigerated trucks which also contain standard products is equivalent to the minimization of patterns of the subset $P^{S>0,C\cup F>0}$. Optimal values from the

Stages 1.–3. are again incorporated as additional constraints:

$$
\begin{aligned}
&z^{4.} = && \min \sum_{p \in P^{S>0, C\cup F>0}} x_p && \text{duals:} && \text{(4a)}\\
&\text{(Stage 4.)} && \text{subject to (1b)–(1c), (2b), (3b)} && && \\
& && \sum_{p \in P^{S,C,F>0}} x_p \le z^{3.} && [\delta_{4b}] && \text{(4b)}\\
& && \sum_{p \in P^{S>0, C\cup F>0}} x_p \ge \mathrm{LB}^S + z^{2.} - z^{1.}. && [\delta_{4c}] && \text{(4c)}
\end{aligned}
$$

The objective (4a) minimizes the number of refrigerated trucks which also transport at least one standard product. In the following, such a truck is denoted as *mixed truck*. A lower bound on the total number of mixed trucks is $\mathrm{LB}^S - (z^{1.} - z^{2.})$. The difference $z^{1.} - z^{2.}$ gives the number of trucks containing only standard products. Subtracting this number from the lower bound for the number of trucks required to pack all standard products, gives a lower bound on the number of mixed trucks. Constraint (4b) restricts the number of frozen trucks. As constraint (3c), constraint (4c) is not mandatory but helps to strengthen the linear relaxation.

Stage 5.: Minimize product splitting

Recall from Section 1 that the last objective of minimizing product splitting applies only to the splitting allowed policy and can be put into effect in two different ways using objective 5a. or 5b.

In the case of objective 5a., minimizing the overall number of splits is equivalent to minimizing the number of different products packed into all bins. To this end, let $s_p = |\{a_i^p > 0 : i \in I\}|$ denote this number of different products/items in a pattern $a_i^p \in P$. Then, the actual number of splits is given by the difference of the total number of different products in all used bins and the total number m of products. Therefore, minimizing the number of splits can be formulated as

$$
\begin{aligned}
&z^{5a.} = && \min \sum_{p \in P} s_p x_p && \text{duals:} && \text{(5a)}\\
&\text{(Stage 5a.)} && \text{subject to (1b)–(1c), (2b), (3b), (4b)} && && \\
& && \sum_{p \in P^{S>0, C\cup F>0}} x_p \le z^{4.}. && [\delta_{5b}] && \text{(5b)}
\end{aligned}
$$

Constraint (5b) restricts the number of mixed trucks as determined at Stage 4.

In the case of objective 5b., it only counts whether a product is split or not, but the number of trucks a split product occupies is irrelevant. Minimizing the number of split products is equivalent to maximizing the number of *integrally packed products*. A product is integrally packed if all its pallets are packed on the same truck. Accordingly, for a pattern $p \in P$, we define for our minimization problem the negative number of integrally packed products as $d_p = -|\{i \in I : a_i^p = q_i\}|$. Hence, our objective is expressed by the minimization of the total value of d_p over all patterns used:

$$
\begin{aligned}
&z^{5b.} = && \min \sum_{p \in P} d_p x_p && \text{duals:} && \text{(5c)}\\
&\text{(Stage 5b.)} && \text{subject to (1b)–(1c), (2b), (3b), (4b), (5b)} && && \\
& && \sum_{p \in P} \left(\sum_{i \in I} a_i^p \right) x_p \le \sum_{i \in I} q_i. && [\gamma] && \text{(5d)}
\end{aligned}
$$

Constraint (5d) ensures that each item $i \in I$ is packed at most q_i times. Note that the latter constraint is indeed necessary because otherwise one might artificially improve the objective function by adding additional integrally packed products more often than demanded. In combination with the covering constraint (1b), both constraints (5d) and (1b) are always binding. Therefore, adding disaggregated constraints of the form $\sum_{p \in P} a_i^p x_p = q_i$ for all $i \in I$ does not yield a tighter formulation.

4.2. Column generation

With the models (1), (2), (3), (4), (5a)–(5b), and (5c)–(5d) we have presented six different extensive formulations that all require a CG-based solution approach. In principle, pattern generation is algorithmically simple to manage, since we will show that the subproblem solution finally boils down to solving binary or bounded *2-dimensional knapsack problems*, see (Martello and Toth, 2003), (Kellerer *et al.*, 2004, ch. 9.6). In the following, we assume that a linear relaxation of the RMP is solved and has produced dual prices $(\pi_i)_{i\in I}$ and (δ_{con}) for the respective constraint(s) *con*. The task of the SP is to determine at least one negative reduced-cost pattern or to prove that no negative reduced-cost pattern exists.

A first difficulty for precisely describing the associated SPs lies in the fact that each of the six SPs associates different dual prices to groups of patterns depending on whether they include standard, cooled, or frozen products/items. Handling these multiple cases is indeed confusing. A fundamental step towards a concise and unified handling is the idea to further divide each SP into 2D-KPs for subsets of patterns. Solving the SP then amounts to solving the associated 2D-KPs. The decisive point is that the reduced cost of a pattern $p = (a_i)$ can be described as (at least for the first four stages, see below)

$$\tilde{c}_p = \delta - \sum_{i\in I} \sigma_i a_i,$$

where the coefficients $\delta = \delta^X$ and $\sigma_i = \sigma_i^X$ for $i \in I$ depend of the particular subcase X. Table 3 formalizes the different (sub)cases: Depending on the stage/objective, there are between one and five different pattern subsets (see Table 2 for their formal definition) for which independent 2D-KPs need to be solved. For each pattern subset, the columns σ_i and δ describe how these coefficients can be computed from the given dual solution $(\pi_i)_{i\in I}$ and (δ_{con}) (using only dual prices of constraints *con* that exist at the actual stage). Moreover, the 2D-KP can be restricted to an item subset $\mathcal{I} \subset I$. Finally, some pattern subsets require the presence of one or two subsets of items. For example, $P^{S>0,C,F>0}$ requires that at least one item of a standard product and one item of a frozen product are present in the pattern. For this purpose, let $\mathcal{R}$ be the *set of required subsets of items*, i.e., for $P^{S>0,C,F>0}$ there is $\mathcal{R} = \{I^S, I^F\}$. The elements of the respective sets $\mathcal{R}$ are displayed in the last column of Table 3.

A second difficulty stems from the fact that, for the objectives 5a. and 5b., the reduced cost of a pattern is no longer completely linear in the coefficients of the pattern. However, all 2D-KPs can be formalized as follows. Nonnegative bounded integer (binary for $u_i = 1$) variables y_i for $i \in I$ describe the unknown pattern coefficients a_i. Moreover, for these variables $\mathbf{y} = (y_i)_{i\in I}$, the function $f(\mathbf{y})$ captures the non-linear part of the reduced cost (described in detail in the next paragraphs). For a given subset of admissible items $\mathcal{I}$ (in the sense of the second last column of Table 3), the extended 2D-KP can now be described as

$$z^{\mathrm{KP}(\mathcal{I},\sigma_i,\delta,\mathcal{R})} = \quad \delta - \max\left(\sum_{i\in\mathcal{I}} \sigma_i y_i - f(\mathbf{y})\right) \tag{6a}$$

$$(\mathrm{KP}(\mathcal{I},\sigma_i,\delta,\mathcal{R})) \quad \text{subject to} \sum_{i\in\mathcal{I}} w_i y_i \le W \tag{6b}$$

$$\sum_{i\in\mathcal{I}} b_i y_i \le B \tag{6c}$$

$$\sum_{i\in J} y_i \ge 1 \qquad J \in \mathcal{R} \tag{6d}$$

$$y_i \in \{0,1,\ldots,u_i\}, \qquad i \in \mathcal{I}. \tag{6e}$$

The objective (6a) is the minimization of the reduced cost of the pattern. The constraints (6b) and (6c) are the capacity constraints. The defining property of some pattern subsets to contain at least one item of one or two particular categories is enforced by condition(s) (6d). The domain of the decision variables is defined by (6e).

At Stages 1.–4., the function f vanishes, i.e., $f \equiv 0$, so that $\mathrm{KP}(\mathcal{I},\sigma_i,\delta,\mathcal{R})$ reduces to a standard binary or bounded 2D-KP with the additional constraints (6d). A straightforward solution approach could be to solve

Stage	Pattern	Dual prices		Item	Elements of subsets $\mathcal{R}$
	subsets	σ_i	δ	subset $\mathcal{I}$	of required items
1.	P	π_i	1	I	
2.	P^S	π_i	$1-\delta_{2b}$	I^S	I^S
	$P^{S,C\cup F>0}$	π_i	$1-\delta_{2b}-\delta_{2c}$	I	$I^C \cup I^F$
3.	P^S	π_i	$1-\delta_{2b}$	I^S	I^S
	$P^{S,C>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}$	$I^S \cup I^C$	I^C
	$P^{S,C,F>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}$	I	I^F
4.	P^S	π_i	$1-\delta_{2b}$	I^S	
	P^C	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}$	I^C	
	$P^{C,F>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}-\delta_{4b}$	$I^C \cup I^F$	I^F
	$P^{S>0,C>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{4c}$	$I^S \cup I^C$	I^S, I^C
	$P^{S>0,C,F>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}-\delta_{4b}-\delta_{4c}$	I	I^S, I^F
5a.	P^S	π_i	$1-\delta_{2b}$	I^S	
	P^C	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}$	I^C	
	$P^{C,F>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}-\delta_{4b}$	$I^C \cup I^F$	I^F
	$P^{S>0,C>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{4c}-\delta_{5b}$	$I^S \cup I^C$	I^S, I^C
	$P^{S>0,C,F>0}$	π_i	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}-\delta_{4b}-\delta_{4c}-\delta_{5b}$	I	I^S, I^F
5b.	P^S	$\pi_i+\gamma$	$1-\delta_{2b}$	I^S	
	P^C	$\pi_i+\gamma$	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}$	I^C	
	$P^{C,F>0}$	$\pi_i+\gamma$	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}-\delta_{4b}$	$I^C \cup I^F$	I^F
	$P^{S>0,C>0}$	$\pi_i+\gamma$	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{4c}-\delta_{5b}$	$I^S \cup I^C$	I^S, I^C
	$P^{S>0,C,F>0}$	$\pi_i+\gamma$	$1-\delta_{2b}-\delta_{2c}-\delta_{3b}-\delta_{3c}-\delta_{4b}-\delta_{4c}-\delta_{5b}$	I	I^S, I^F

Table 3: Overview of pricing subproblems.

several standard binary 2D-KP that previously pack one item of each set $J \in \mathcal{R}$. Instead of enumerating all combinations, the selection of previously packed items could also be embedded into a branch-and-bound algorithm. However, we formulate and solve this subproblem as a variant of the shortest-path problem with resource constraints (see next paragraph).

For the first splitting variant that minimizes the total number of splits, i.e., objective 5a., the function f must count the number of products the constructed pattern contains. This is done with

$$f(\boldsymbol{y}) = |\{i \in I : y_i > 0\}|.$$

For the second splitting variant that minimizes the number of split products, i.e., objective 5b., the function f must count the (negative) number of integrally packed products the constructed pattern contains. The definition

$$f(\boldsymbol{y}) = -|\{i \in I : y_i = q_i\}|$$

gives the correct contribution. As an example, the two 2D-KPs arising at Stage 2. are specified in the following example.

Example 2. *At Stage 2., we consider two 2D-KPs that generate patterns of the subsets P^S and $P^{S,C\cup F>0}$. Recall that δ_{2b} and δ_{2c} are the dual prices of constraints (2b) and (2c), respectively. Both 2D-KPs have $\sigma_i = \pi_i$ for all $i \in I$ and $f \equiv 0$. The first 2D-KP generates patterns of set P^S and is defined by $\delta = 1 - \delta_{2b}$, $\mathcal{I} = I^S$, and $\mathcal{R} = \{I^S\}$, while the second generates patterns of the set $P^{S,C\cup F>0}$ and is defined by $\delta = 1 - \delta_{2b} - \delta_{2c}$, $\mathcal{I} = I$, and $\mathcal{R} = \{I^C \cup I^F\}$.*

SPPRC-based solution. The different problems KP$(\mathcal{I}, \sigma_i, \delta, \mathcal{R})$ can be formulated as *shortest path problems with resource constraints* (SPPRCs, Irnich and Desaulniers, 2005). SPPRCs can model intricate relationships between attributes and many variants of the SPPRC can be effectively solved by dynamic-programming labeling algorithms. Such an approach has been successfully chosen for the one-dimensional BP by Wei *et al.* (2019) and for the VPP by Heßler *et al.* (2018). The advantage of a labeling-based approach is that also subsets $\mathcal{R}$ of required items and the objectives 5a. and 5b. defining the function f can be considered as well as involved branching decisions (discussed later in Section 4.4).

For the sake of simplicity, we start with the case that any item can be contained in a pattern, i.e., $\mathcal{I} = I = \{1, 2, \ldots, m\}$. The underlying digraph $G = (V, E)$ then consists of $m + 1$ vertices $V = \{0, \ldots, m\}$. The arc set E consists of $m + \sum_{i \in I} u_i = \sum_{i \in I}(u_i + 1)$ arcs partitioned into m subsets $E(i)$ associated with item $i \in I$. The set $E(i)$ contains $u_i + 1$ parallel arcs connecting vertex $i - 1$ with vertex i. Thus, we have an arc set $E(i)$ consisting of arcs $e_j(i)$ with $j \in \{0, 1, \ldots, u_i + 1\}$.

Each arc $e_j(i)$ has three attributes, two for the weights and one for the profit. For the j-th arc $e_j(i)$ of $E(i)$, these are defined as $w(e_j(i)) = j \cdot w_i$, $b(e_j(i)) = j \cdot b_i$, and $\sigma(e_j(i)) = j \cdot \sigma_i$, respectively.

An example of the digraph is sketched in Figure 2. Each 0-m-path $(0, e_{j_1}(1), 1, e_{j_2}(2), 2, \ldots, m - 1, e_{j_m}(m), m)$ uniquely defines a pattern (a_i) with $a_i = j_i$. Necessary conditions for its feasibility are that $\sum_{i=1}^{m} w(e_{j_i}(i)) \leq W$ and $\sum_{i=1}^{m} b(e_{j_i}(i)) \leq B$ holds. Partial paths that do not fulfill these conditions can be directly discarded.

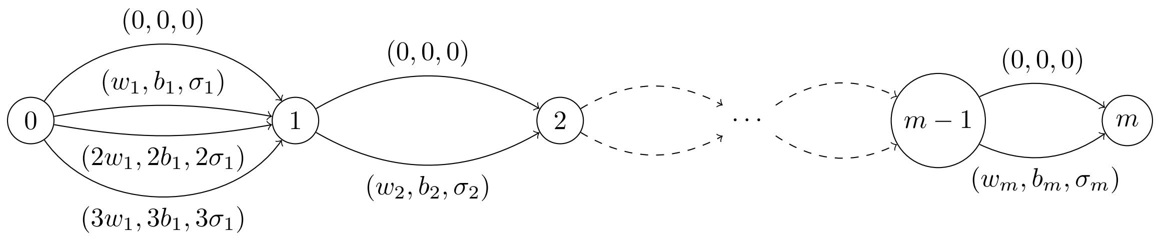

Figure 2: SPPRC digraph with items having $u_1 = 3, u_2 = 1$, and $u_m = 1$. The arcs $e \in E$ are shown with their weights and profit $(w(e), b(e), \sigma(e))$.

If $\mathcal{I} \subsetneq I$, i.e., some categories of items are not in $\mathcal{I}$, the corresponding digraph can be obtained by reducing the arc set so that for each $i \in I \setminus \mathcal{I}$ only the null-arc $(0, 0, 0)$ is in $E(i)$. Subsequently, the digraph can be shrunk by merging vertices $i - 1$ and i for all $i \in I \setminus \mathcal{I}$.

At Stages 2.–5., three additional boolean attributes $\boldsymbol{v} = (v_S, v_C, v_F)$ for the three categories standard, cooled, and frozen are added to indicate whether a specific product category has been packed. For an item $i \in I$, we define $\boldsymbol{v}(e_j(i)) = (1, 0, 0)$, $(0, 1, 0)$, or $(0, 0, 1)$ for all j according to the category product/item i belongs to.

The SPPRC is solved by a forward dynamic-programming labeling algorithm that works as follows: The starting point is the trivial partial path (0) with the initial label $(0, 0, 0)$. All labels at a vertex $i - 1$ are extended to vertex i iteratively for $i = 1, 2, \ldots, m$. A forward extension along an arc $e(i) \in E(i)$ of label $L(F) = (w, b, \boldsymbol{v}, \sigma)$ (omitting the index j) produces the label $(w + w(e(i)), b + b(e(i)), \max\{\boldsymbol{v}, \boldsymbol{v}(e(i))\}, \sigma + \sigma(e(i)))$, where the maximum operator is applied component-wise. The labels at m and the associated 0-m-paths have to be filtered at the end: Only labels $(w, b, \boldsymbol{v}, \sigma)$ with correct $\boldsymbol{v}$-values provide feasible patterns, e.g., $v_C + v_F \geq 1$ for the pattern set $P^{S, C \cup F > 0}$ used at Stage 2. in the second subcase. Moreover, the reduced cost of the associated pattern is $\delta - \sigma$.

The work of Heßler *et al.* (2018) has pointed out that different dominance principles between labels are possible and worth to be investigated. There clearly exists a tradeoff between the strength of a dominance rule and its computational burden resulting from the pairwise comparison of labels. A strong dominance rule can help to drastically reduce the overall number of labels that remain at termination of the labeling algorithm. The result is also fewer labels at intermediate vertices, which reduces the computational effort associated with the label extension step and later dominance comparisons. On the downside, the application of the following strong dominance rule may require the pairwise comparison of a possibly huge number of labels.

Rule 1. *(Strong Dominance) A forward label $L(F) = (w, b, \boldsymbol{v}, \sigma)$ of a forward partial path F dominates another forward label $L(F') = (w', b', \boldsymbol{v}', \sigma')$ of a forward partial path F' ending at the same vertex if $w \leq w'$, $b \leq b'$, $\boldsymbol{v} \geq \boldsymbol{v}'$, and $\sigma \geq \sigma'$.*

The following weaker dominance rule allows a direct $\mathcal{O}(1)$ comparison of labels, immediately at the time when a label is created, if labels are stored in a lookup table.

Rule 2. *(Weak Dominance) A forward label $L(F) = (w, b, \boldsymbol{v}, \sigma)$ of a forward partial path F dominates another forward label $L(F') = (w', b', \boldsymbol{v}', \sigma')$ of a forward partial path F' ending at the same vertex if $w = w'$, $b = b'$, $\boldsymbol{v} = \boldsymbol{v}'$, and $\sigma \geq \sigma'$.*

As in Heßler *et al.* (2018), also our pretests have confirmed that the weak dominance Rule 2 performs better than the strong Rule 1 for most benchmark instance used in the computational experiments. Therefore, we do not use Rule 1 and also omit its proof. The validity of the weak dominance Rule 2 is obvious.

Finally, we discuss the incorporation of the functions f at Stage 5. into the SPPRC modelling and solution approach. For the first objective 5a., the function f counts the number of products the pattern contains, i.e., $f(\boldsymbol{y}) = |\{i \in I : y_i > 0\}|$. Consequently, the profit attribute on the arcs of the SPPRC digraph must consider the number of different products in the resulting pattern. Thus, a cost of 1 is subtracted for every item that is packed, i.e., every non-zero arc. An example is given in Figure 3.

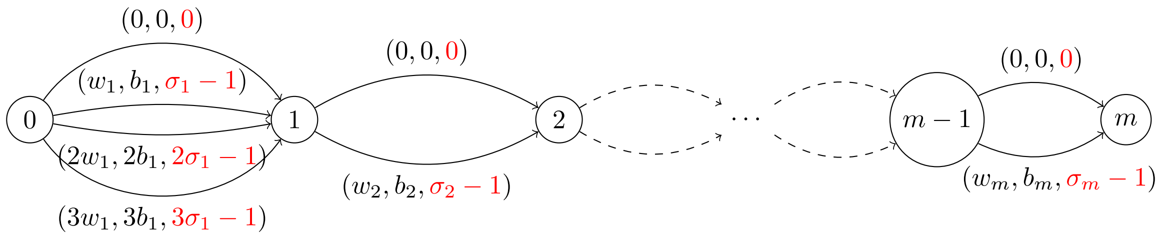
$(0,0,0)$
$(w_1, b_1, \sigma_1 - 1)$
$(2w_1, 2b_1, 2\sigma_1 - 1)$
$(3w_1, 3b_1, 3\sigma_1 - 1)$
$(0,0,0)$
$(w_2, b_2, \sigma_2 - 1)$
$(0,0,0)$
$(w_m, b_m, \sigma_m - 1)$
0
1
2
$\cdots$
$m-1$
m

Figure 3: SPPRC digraph for objective 5a. with otherwise identical attributes as depicted in Figure 2. The arcs $e \in E$ are shown with their weights and profit $(w(e), b(e), \sigma(e))$.

For the second objective 5b., the function f counts the (negative) number of integrally packed products the pattern contains, i.e., $f(\boldsymbol{y}) = -|\{i \in I : y_i = q_i\}|$. Hence, arc profits are modified if an item i is packed completely, i.e., the additional profit of 1 is added to the q_ith arc of item i (present only if $q_i \leq u_i$). An example is given in Figure 4. Note that the dual price γ of constraint (5d) is already included in the definition of σ_i for all $i \in I$.

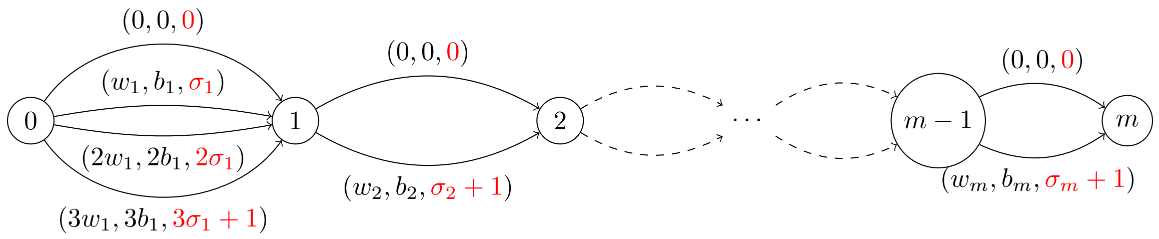
$(0,0,0)$
(w_1, b_1, σ_1)
$(2w_1, 2b_1, 2\sigma_1)$
$(3w_1, 3b_1, 3\sigma_1 + 1)$
$(0,0,0)$
$(w_2, b_2, \sigma_2 + 1)$
$(0,0,0)$
$(w_m, b_m, \sigma_m + 1)$
0
1
2
$\cdots$
$m-1$
m

Figure 4: SPPRC digraph for objective 5b. with otherwise identical attributes as depicted in Figure 2. The arcs $e \in E$ are shown with their weights and profit $(w(e), b(e), \sigma(e))$.

4.3. Stabilization

The CG process can be stabilized by using dual inequalities (Valério de Carvalho, 2005; Ben Amor *et al.*, 2006; Gschwind and Irnich, 2016). In the following, we restrict ourselves to inequalities that are formulated only in the dual variables π_i that correspond to the demand fulfillment constraints (1b) of the model solved at Stage 1. (for the subsequent stages we consider the projection onto the m-dimensional space defined by the π-variables). Let D^* be the set of dual optimal solutions to the linear relaxation. For $\boldsymbol{t} \in \mathbb{Z}^m$ and $t \in \mathbb{Z}$, the *dual inequality* (DI) $\boldsymbol{t}^\top \pi \leq t$ is a *dual-optimal inequality* (DOI) if $D^* \subseteq \{\pi : \boldsymbol{t}^\top \pi \leq t\}$. A set of DIs is a set of *deep dual-optimal inequalities* (DDOIs) if at least one optimal dual solution $\pi^* \in D^*$ fulfills *all* inequalities of this set.

We first focus on Stage 1. and the B-VPP and 01-VPP. According to Heßler *et al.* (2018), *pair inequalities* (PIs) $\pi_h \geq \pi_i$ are DDOIs for all $h, i \in I$ with $w_h \geq w_i$ and $b_h \geq b_i$. In the primal formulation (1), these additional columns stand for the exchange of the larger item h by the smaller item i, i.e., a zero-cost column in the RMP with entries -1 and 1 for h and i, respectively. Moreover, *subset inequalities* (SIs) $-\pi_h + \sum_{i\in S} \pi_i \leq 0$ are defined for any item $h \in I$ and subset $S \subseteq I \setminus \{h\}$ with $\sum_{i\in S} w_i \leq w_h$ and $\sum_{i\in S} b_i \leq b_h$. We refer to a SI as (S, h). In general, SIs are not necessarily DOIs or DDOIs for the B-VPP and the 01-VPP. Nevertheless, their associated primal columns can be added to stabilize the RMP at the risk of a possible over-stabilization. Over-stabilization can be restored with a recovery procedure of low computational effort as described in detail in Gschwind and Irnich (2016).

At Stages 2.–4., the use of DIs is still possible when categories of items are kept separate, i.e., a SI for (S, h) can only be used if all items $i \in S$ and also item h all belong to the same, stage-dependent *subset of exchangeable items*. An overview of the applicable DIs at the different stages is provided in Table 4. For example, at Stage 2., the PIs and SIs exclusively exchange either items of standard products I^S or of refrigerated products $I^C \cup I^F$. At Stages 3. and 4., exchanges are only allowed among items of the same category.

Stage/ objective	Subsets of exchangeable items	RHS of a SI defined by (S, h), i.e., cost c of the associated primal column
1.	I	0
2.	$I^S, I^C \cup I^F$	0
3. and 4.	I^S, I^C, I^F	0
5a.	I^S, I^C, I^F	$\lvert\{i \in I : t_i > 0\}\rvert - \chi_1(u_h)$
5b.	I^S, I^C, I^F	1

Table 4: Overview of DIs.

At Stage 5., we have to take into account the following speciality. The transformation of a solution with packing and DI columns into a pure packing column solution may change the number of packed products in a pattern (to be taken into account at Stage 5a.). Likewise, an integrally packed product may be transformed into a split product and vice versa (to be taken into account at Stage 5b.). In order to ensure a valid objective value after the replacement, we have to modify the right-hand side of a DI from 0 to some integer value c, i.e., add a cost c to the corresponding primal DI column. Table 4 summarizes the different cases. For example, let $p \in P$ be a pattern and let $\sum_{i\in I} t_i \pi_i \leq c$ be a DI. If at Stage 5a. in the primal model both corresponding variables are positive, the following replacement can be performed: Exchange item h by items $\{i \in I : t_i > 0\}$ in pattern p with original associated cost s_p (i.e., the number of packed products) yields a new pattern p' with associated cost $s_{p'} \leq s_p + |\{i \in I : t_i > 0\}| - \chi_1(u_h)$, where χ is the indicator function. The new pattern p' might consist of added items $\{i \in I : t_i > 0\}$ compared to pattern p and if $u_h = 1$ then item h vanishes in pattern p'.

Similarly, at Stage 5b., the (negative) number of integrally packed items for the new pattern p' can be estimated by $d_{p'} \leq d_p + 1$ because item h might change from an integrally packed item to a split item.

As discussed in Gschwind and Irnich (2016) in more detail, one may experiment with different strategies which DIs and at what point in time the DI columns are added to the RMP. On the one hand, DI columns can be added at the very beginning to the initial RMP and stay there during the entire CG process (*static*). On the other hand, only violated DIs can be added during the CG process (*dynamic*). We combine these two strategies which yields a *mixed* approach that adds some DI columns at the start and adds violated DIs later on.

We employ this mixed strategy in the following form: For the initialization of the first RMP, for each item i, the PI $\pi_i \geq \pi_j$ with $j = \arg\min_{k \in I \setminus \{i\}} \{|w_i - w_k| + |b_i - b_k| : w_i \geq w_k, b_i \geq b_k\}$ is added and all SIs with $|S| = 2$ are added. While solving the root node, violated PIs are added dynamically. At Stages 2.–5., we only keep DIs belonging to the subset of exchangeable items according to Table 4 (and remove the others). In the same spirit, we only dynamically add violated PIs belonging to these subsets.

4.4. Branching

We apply a single or a two-level branching scheme depending on the stage. At Stage 1., there is only one level where we apply the branching scheme suggested by Vanderbeck (1999). The idea is to formulate the 2D-VPP as a pure binary problem so that all pattern coefficients are also binary. Branching is then performed choosing two disjoint subsets I^0, I^1 of items. In any fractional solution there exists a pair (I^0, I^1) for which the number of patterns having coefficients a_i equal to 0 (1) for all $i \in I^0$ $(\in I^1)$ is fractional. For a detailed explanation and examples, we refer to Heßler *et al.* (2018).

At Stages 2.–4., there are two branching levels. First, we branch on the number of refrigerated trucks (2c), the number of frozen trucks (3c), and the number of mixed trucks (4c), respectively. Second, we apply the above described branching rules of Vanderbeck (1999).

At Stage 5., there are also two branching levels. First, we branch on the number of integrally packed products, i.e., when the value

$$\sum_{p=(a_i)\in P : a_i^p = q_i} a_i^p x_p \tag{7}$$

is fractional for an item $i \in I$. Note that this is a special case of Vanderbeck branching, which we apply subsequently at the second level.

At all levels, the branching variable/term is selected as one with fractional part closest to 0.5. Ties are resolved randomly.

Note that all first-level and second-level branching decisions can be implemented by either adding some linear constraint(s) to the RMP or by removing certain pattern subsets. Since the branching rule of Vanderbeck (1999) ensures integrality, the overall branching scheme is complete for all stages.

Finally, we use a depth-first tree exploration strategy. The intention is to find an integer solution as soon as possible. Since the lower bounds of the linear relaxations of formulations (1a)-(5) are often tight, we quickly obtain good and sometimes optimal solutions.

4.5. Acceleration techniques

When applying the policy splitting allowed, patterns with fewer split products are preferred over patterns with more split products. Therefore, it is beneficial to already generate patterns with very few split products (if any) at the Stages 1.–4. so that the initial RMP at Stage 5. mainly consists of variables representing patterns with many integrally packed products. To foster that predominantly patterns with only integrally packed products are generated, the SP is always first solved heuristically over a reduced SPPRC digraph. In this reduced digraph, each item $i \in I$ has only two associated parallel arcs $(0, 0, 0)$ and $(u_i w_i, u_i b_i, u_i \pi_i)$ so that for $u_i = q_i$ the item is either integrally packed or not packed at all. An example of a reduced graph is given in Figure 5. Only if no pattern with negative reduced cost is found in the reduced digraph, SPPRC SPs are solved again using the complete digraph.

This approach has two advantages: First, the 2D-KPs at the Stages 1.–4. are solved faster because in most iterations the reduced network provides negative reduced cost patterns. Secondly, Stage 5. is solved faster because the RMP already consists of beneficial columns. Computational results show that this approach significantly accelerates the CG process. For a computational analysis, we refer to Section 6.4.

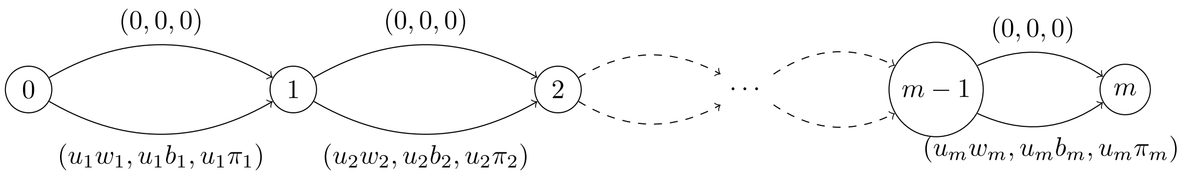

Figure 5: Reduced SPPRC digraph for objective 1. with otherwise identical attributes as depicted in Figure 2. The arcs $e \in E$ are shown with their weights and profit $(w(e), b(e), \sigma(e))$.

5. Heuristics

For real-world applications, user acceptance is a crucial success factor, especially for constructive heuristics, where the key users want to get some insights into the decision mechanism. Therefore, we developed various heuristic ideas and presented their underlying rationals as well as their strengths and weaknesses to the decision makers of the affected company. For comparison, we also re-implemented the strategy previously employed by the dispatchers, the so-called *Business Today* logic.

For our own development, we started with the well-known BFD, FFD, and WFD heuristics, all of them employing the concept of surrogate weight (Section 5.1). Then five additional heuristic approaches were developed in accordance to the real-world situation of our industrial partner (Section 5.2). Comparisons to results of the Business Today logic can be found in Section 6.

5.1. Basic single-class heuristics

At the beginning, we will introduce a number of heuristics which consider only a single class of items and do not distinguish between products of different categories. Also splitting products is not taken into account. At the end of this section, we will have obtained one combined heuristic which consists of running all heuristics described so far and taking the best solution. This combined heuristic will serve as building block for a more complicated heuristic framework described in Section 5.4.

As pointed out in the introduction, we are facing a highly heterogeneous product range where the unit weight ρ_k of product $k \in K$ varies considerably. It is obvious that truck loads containing mainly heavy goods ($\rho_k \gg W/B$) will leave excess pallet spaces even if the weight capacity is exhausted. On the other hand, truck loads containing mainly lightweight goods ($\rho_k \ll W/B$) still provide excess weight capacity even if all pallet spaces are used. Thus, a "good" loading pattern should utilize both capacity dimensions by mixing heavy and lightweight products in a suitable way.

Based on this simple observation the transport planners previously created loading patterns filling one truck after the other in a FF manner. Each truck is manually assigned its load following a strict rule that alternately packs the heaviest and the lightest unpacked item as described in the following *Business Today* Algorithm 1.

Practical experience showed: If each product fills only one pallet, i.e., $b_i = 1$ for all $i \in I$, then this procedure amounts to a pairwise combination of items with large and small unit weight ρ_i and shows reasonably good results. However, this is a rare case, and for all other, more general instances, inefficient solutions may occur. In the worst case, many large items (high w_i and b_i) with medium density ρ_i remain until the end, which leads to unnecessary additional trucks.

Naturally, Algorithm 1 suffers from neglecting the absolute weights of items and from the FF strategy of handling trucks. However, sorting items by decreasing weight w_i or decreasing number of pallets b_i would only be useful for instances with a clear dominance in one dimension. Our real-world test instances showed that both constraints, weight and pallet capacity, can become active, i.e., there is no clear dominance of one of the two dimensions.

To overcome this dilemma, we suggest to apply the concept of a *surrogate weight* proposed by Caprara and Toth (2001). The surrogate weight of an item $i \in I$ constitutes a convex combination of relative weight

Algorithm 1 Business Today

```
Sort item set $I$ by increasing $\rho_i$.
Open truck 1 and set $t \leftarrow 1$.
while $I \neq \emptyset$ do
    Take the last (first) item $i$ from $I$ in an odd (even) iteration.
    if $i$ fits into truck $t$ then
        Insert $i$ into truck $t$.
    else
        Open a new truck $t+1$ and insert $i$.
        $t \leftarrow t+1$.
    end if
    Remove $i$ from $I$.
end while
```

and relative pallet number. For each $i \in I$, it is defined as

$$s_i = \lambda \frac{w_i}{W} + (1-\lambda)\frac{b_i}{B}, \quad \text{where } \lambda = \frac{\sum_{i \in I} \frac{w_i}{W}}{\sum_{i \in I} \left(\frac{w_i}{W} + \frac{b_i}{B} \right)}.$$

Based on these surrogate weights, we can (almost) transform our packing problem into a classical one-dimensional bin packing problem and apply well-known standard heuristics. In particular, we take FFD, BFD, and WFD, and adapt them for the solution of our truck loading problem by using the surrogate weight s_i. In the following, we present further heuristic approaches based on the concept of a surrogate weight.

5.2. Block-based single-class heuristics

In this section, we introduce three heuristics called A, B, and C, which are all based on a fixed pattern of assigning blocks of items to trucks. Therefore, the lower bound of the number of trucks LB (see Section 3) is computed and LB trucks are opened. Then, the item set is partitioned into blocks of equal size LB. In each iteration, one block is selected and matched to the set of trucks, i.e., every item of the block is assigned to one of the trucks. The selection of blocks and the matching patterns differ between the three heuristics. If an item does not fit into the preferred truck, the item is moved to the next truck in the sequence (following a FF logic) or, if necessary, a new truck is opened, see Algorithm 2. Note that any new trucks opened during the execution of the heuristic only serve as a backup for loading items exceeding the capacity, while the matching of subsets remains restricted to the originally opened LB trucks.

All three heuristics are based on sorting the items by increasing surrogate weights s_i. Their details are informally described below followed by pseudocode. An illustrative example is given in Table 5 further below.

Heuristic A assigns the LB items with largest surrogate weights to the LB trucks and then adds the LB items with smallest surrogate weights in a reversed order, such that the items with $s_{\max}$ and $s_{\min}$ end up in the same truck number 1. These two steps are repeated iteratively for the remaining items as described in Algorithm 3.

Heuristic B also starts by assigning the LB items with largest surrogate weights to the LB trucks, but then proceeds by assigning the next largest (w.r.t. s_i) LB items to the trucks in a mirrored order such that items with LB*th* and (LB + 1)-largest surrogate weight end up being assigned to the same truck number LB, see Algorithm 4.

Finally, Heuristic C repeats the first step of Heuristic A and iteratively assigns the items with largest surrogate weights to the LB trucks, always maintaining the same order of trucks, see Algorithm 5.

To illustrate the different heuristics, an example with 20 items is given in Table 5. Obviously, Heuristic A exhibits more similarities to the Business Today logic, including the feature of not keeping the items with lowest s_i until the end. Heuristics B and C overcome that issue. Besides that, Heuristic B aims at uniformly

Algorithm 2 Subprocedure **Assign item i to truck t**

```
Try to insert $i$ into an open truck by a FF strategy,
i.e., try all trucks $t, t+1, \dots, \mathrm{LB}$ and then trucks $1, 2, \dots, t-1$ until item $i$ fits.
If no such truck is found, insert $i$ into a truck $\mathrm{LB}+1, \mathrm{LB}+2, \dots$ by a FF strategy, opening a new truck
if necessary.
```

Algorithm 3 Heuristic A

```
Calculate lower bound LB and open LB trucks.
Sort item set $I$ by increasing $s_i$.
Partition set $I$ into $H := \lceil m/\mathrm{LB} \rceil$ subsets $I_j$ of cardinality LB according to the sorting. {For simplicity
of notation we assume that $H$ is even.}
$j \leftarrow 1$, $h \leftarrow H$
while $j < h$ do
   for $i \leftarrow$ LB downto 1 do
      Assign item $i$ in set $I_h$ to truck $\mathrm{LB} - i + 1$.
   end for
   $h \leftarrow h - 1$
   for $i \leftarrow 1$ to LB do
      Assign item $i$ in set $I_j$ to truck $i$.
   end for
   $j \leftarrow j + 1$
end while
```

Algorithm 4 Heuristic B

```
Calculate lower bound LB and open LB trucks.
Sort item set $I$ by increasing $s_i$.
Partition set $I$ into $H := \lceil m/\mathrm{LB} \rceil$ subsets $I_j$ of cardinality LB according to the sorting. {For simplicity
of notation we assume that $H$ is even.}
$j \leftarrow H$
while $j \geq 1$ do
   for $i \leftarrow$ LB downto 1 do
      Assign item $i$ in set $I_j$ to truck $\mathrm{LB} - i + 1$.
   end for
   $j \leftarrow j - 1$
   for $i \leftarrow 1$ to LB do
      Assign item $i$ in set $I_j$ to truck $i$.
   end for
   $j \leftarrow j - 1$
end while
```

Algorithm 5 Heuristic C

```
Calculate lower bound LB and open LB trucks.
Sort item set $I$ by increasing $s_i$.
Partition set $I$ into $H := \lceil m/\mathrm{LB} \rceil$ subsets $I_j$ of cardinality LB according to the sorting.
for $j \leftarrow H$ downto 1 do
   for $i \leftarrow$ LB downto 1 do
      Assign item $i$ in set $I_j$ to truck $\mathrm{LB} - i + 1$.
   end for
end for
```

	Business Today					Heuristic A				Heuristic B				Heuristic C			
truck $\setminus$ iter.						1	2	3	4	1	2	3	4	1	2	3	4
$t = 1$	20	1	19	2	18	20	1	15	6	20	11	10	1	20	15	10	5
$t = 2$	3	17	4	16		19	2	14	7	19	12	9	2	19	14	9	4
$t = 3$	5	15	6	14		18	3	13	8	18	13	8	3	18	13	8	3
$t = 4$	7	13	8	12	9	17	4	12	9	17	14	7	4	17	12	7	2
$t = 5$	11	10				16	5	11	10	16	15	6	5	16	11	6	1

Table 5: Comparison of solutions produced by the Business Today algorithm and Heuristics A, B, and C. The example has 20 items $I = \{1, 2, \ldots, 19, 20\}$ and LB = 5. Items are numbered according to their surrogate weight in ascending order, i.e., item 1 is the smallest item and item 20 is the largest item.

filling all trucks. According to the basic intuition of how good solutions might look like, this should reduce the probability that any smaller item does not fit into its *preferred truck*. On the other hand, Heuristic C provides gaps of different sizes in the LB different trucks, which may offer more room to maneuver if an item does not fit into its assigned truck.

5.3. *Density-based single-class heuristics*

The third group of heuristics for items of the same class is based on the unit weight ρ considerations discussed at the beginning of Section 5.1. Recall that "good" loading patterns, in general, utilize both capacity dimensions and do not leave large excess capacity of pallet space if weight capacity is used to the limit, and vice versa. Therefore, we can define a desired *average density* for an ideal load of a truck, namely the ratio $optavg_\rho = W/B$. Clearly, a loading pattern P_t of a truck t with an average density $avgr_t := \sum_{j \in P_t} w_j / \sum_{j \in P_t} b_j$ close to $optavg_\rho$ and total weight close to the capacity W also fills the available pallet space close to the limit B, and vice versa.

Based on this simple observation, we introduce two heuristics D and E. In contrast to heuristics A, B, and C, they both consider trucks one after the other and for each considered truck t they add the item yielding a new average density $avgr_t$ as close as possible to the target value $optavg_\rho$. Of course, the item also has to fit into the truck at hand. Ties are broken by choosing the item with the largest weight, because smaller items can be expected to fit more easily in later iterations.

Heuristics D and E differ from each other only by the weight measure for the initial sorting step. Note that the sorting only determines the first item put into every truck. Heuristic D sorts by the surrogate weight s_i ensuring that "large" items are assigned first and will not give rise to trucks with low utilization towards the end of the packing procedure. On the contrary, heuristic E focuses on outliers with an unusual density that are potentially hard to assign. Therefore, items are sorted in decreasing order of unit weight ρ_i. By the same rationale also the reverse ordering could be a reasonable choice, but for our test data, pretests showed that this ordering is less effective.

Heuristics D and E could be immediately extended to consider all trucks instead of only the current truck t for inserting items. In that case, the computation of i' would have to be extended to taking the arg min over all currently open trucks, which means that an item is inserted into a truck where the minimal deviation from $optavg_\rho$ can be realized. Indeed, we also implemented this extension with the initialization that the first LB items (according to the given ordering) are packed on LB trucks. However, since improvements were only marginal and occurred only for a small subset of instances, we decided to stick to the simpler standard version described above.

5.4. *Multi-class heuristic scheme of policy splitting forbidden*

All heuristics described in the previous sections focus on creating good patterns utilizing both capacity dimensions, but they take neither items of different product categories packed into different compartments of a truck nor item fragmentation into account, while the latter is considered in the following Section 5.5. Accordingly, we now present a heuristic scheme which is based on the single-class heuristics presented before.

Algorithm 6 Heuristic D (E)

```
Sort items $I$ by decreasing $s_i$ ($\rho_i$).
Let $P_t \leftarrow \emptyset$ be the loading of all trucks $t$.
$t \leftarrow 0$
while $I \neq \emptyset$ do
    $t \leftarrow t + 1$
    Take the first $i$ from $I$ and insert $i$ into $P_t$.
    Set $avgr_t = \rho_i$ and remove $i$ from $I$.
    while items exist that fit into $P_t$ do
        $i' \leftarrow \arg\min_{i \in I} \left\{ \left| \frac{w_i + \sum_{j \in P_t} w_j}{b_i + \sum_{j \in P_t} b_j} - optavg_\rho \right| : i \text{ fits into } P_t \right\}$
        Insert $i'$ into $P_t$ and remove $i'$ from $I$.
    end while
end while
```

These are embedded into a more complicated algorithmic framework consisting of various decomposition and re-merging steps. Since the running times of the single-class heuristics are rather low, we

perform the task of solving a single-class problem for a certain subset of item $\mathcal{I} \subseteq I$ as a subroutine by running all nine heuristics from Sections 5.1, 5.2, and 5.3 and taking the best solution found. More precisely, we run the Business Today algorithm, FFD, BFD, WFD, and Heuristics A to E. The result (=number of trucks) derived by this combined single-class heuristic is denoted by $z_{\text{heu}}(\mathcal{I})$. Additionally, this combined heuristic is executed with a given starting solution of prepacked trucks, i.e., some trucks have reduced residual capacities. We refrain from listing all the details of adapting the single-class heuristics to this situation.

In Section 3, the lower bounds $\text{LB}, \text{LB}^S, \text{LB}^{C,F}$, and LB^F for the corresponding subsets of items are introduced. Applying a single-class heuristic only for standard items and then separately only for refrigerated items yielding heuristic solution values z^S_{heu} and $z^{C,F}_{\text{heu}}$, respectively, we can conclude the following: If $z^{1.}_{\text{sep}} := z^S_{\text{heu}} + z^{C,F}_{\text{heu}} = LB$, then we have reached an optimal solution w.r.t. the objectives of Stages 1. and 2. with $z^{1.} = z^{1.}_{\text{sep}}$ and $z^{4.} = 0$.

Otherwise, if $z^S_{\text{heu}} + z^{C,F}_{\text{heu}} > LB$, we cannot guarantee optimality of the merged solution without mixed trucks. In this case, it makes sense to gradually introduce mixed trucks while trying to reduce the total number of trucks. As already used in Section 4, constraint (4c), the lower bound for mixed trucks ($z^{4.}$) is given by $\text{LB}^{\text{mixed}} = \text{LB}^S + \text{LB}^{C,F} - \text{LB}$ for solutions that are optimal w.r.t. Stages 1. to 3.. Therefore, $\text{LB}^{\text{mixed}} = 0$, if $\text{LB}^S + \text{LB}^{C,F} = \text{LB}$ and $\text{LB}^{\text{mixed}} > 0$, otherwise.

The heuristic scheme striving for a good solution of the lexicographic optimization problem defined by Stages 1.–4. with splitting forbidden works in three phases which are described in Algorithm 7. Phase 1 builds up a solution starting with frozen products. After packing I^F by the combined single-class heuristic (following the Stage 4. objective), the cooled items are added which might require additional trucks (Stage 3. objective). Then a separate solution for the standard item set I^S is determined. Taking the union of the two sets of trucks gives a first solution, where standard and refrigerated items are packed in separate trucks (Stage 2. objective). If the resulting number of trucks z_{best} equals the overall lower bound LB we stop (**STOP 1**), since we have found a solution which is optimal w.r.t. Stages 1. and 2.

Otherwise, we try to improve the currently best solution in Phase 2: Therefore, we allow mixed trucks and pack standard products also into refrigerated trucks. To reach a better starting position for adding also large standard items to the refrigerated trucks given from Phase 1, but keeping the number of mixed trucks small, the load of the refrigerated trucks is *regrouped* at the beginning of Phase 2 with the goal of obtaining some refrigerated trucks with a low load and others which are almost fully packed with cooled products. This is done by subprocedure **Regroup**, which tries to empty the least loaded refrigerated truck by moving its items (in decreasing order of surrogate weight) to other refrigerated trucks selected by a BF strategy. If the number of frozen products is small, we try to keep them together in the regrouping in the spirit of the Stage 3. objective. Therefore, all frozen products contained in the current truck are considered as a single product and moved to a new truck together. Only if this artificial product does not fit, the frozen products are reassigned individually. This regrouping is applied to all trucks in an increasing order of surrogate load. Then the standard items I^S are added to the regrouped refrigerated trucks by the combined single-class heuristic. In this way, we aim to find a packing with a lower number of trucks (Stage 1.), but without increasing the number of refrigerated trucks (Stage 2.). Again, we stop if the new solution matches the lower bound LB^I (**STOP 1**).

A further reduction of the total number of trucks is sought in Phase 3 at the cost of increasing the number of refrigerated trucks. Again we try to improve the possibility of adding standard products to refrigerated trucks. Therefore, we take the refrigerated truck with lowest load w.r.t. the surrogate weight and split its content on two empty trucks, thereby incrementing the number of refrigerated trucks by one. The splitting is performed in a balanced way by assigning the items by a WFD strategy (w.r.t. surrogate weights). Again, all frozen items are treated as one product, if the truck contains both frozen and cooled products. Then the standard items are again added by the single-class heuristic. If the total number of trucks reaches the lower bound LB we stop the procedure. Otherwise, we unpack all standard items again and iterate, taking the refrigerated truck with lowest load that was not split before. If all refrigerated trucks given at the beginning of Phase 3 were split, another iteration is started permitting a second split operation for each refrigerated truck given at the end of the first iteration. Further iterations along this rule are possible, until at some point the number of refrigerated trucks exceeds the lower bound for all items (**STOP 2**). In this case, we also run the single-class heuristic for all items I to reach a possible improvement of the Stage 1. objective.

5.5. Multi-class heuristic scheme for policy splitting allowed

If the multi-class heuristic of Section 5.4 reaches a solution where all lower bounds for the objectives of Stages 1.–4. are reached, obviously the result cannot be improved by splitting products. However, if this is not the case, item fragmentation offers potential for improvements.

In the following, we consider the two different splitting objectives introduced in the Introduction.

5.5.1. Stage 5a.: Minimizing the number of splits

In this setting it is not a-priori clear which items should be split and in which way to gain the largest improvement of the solution. We have chosen to incorporate item splitting as a post-processing step applied to a solution derived from the multi-class heuristic of Section 5.4. In every iteration of the main loop of our Algorithm 8 we try to empty the truck with lowest surrogate load and thus reduce the total number of trucks by one. If this turns out to be impossible, the procedure is stopped. Otherwise we proceed to the next iteration, unless we have reached a solution with LB_1 trucks, which proves optimality of objective 1. and also lets us stop the procedure.

Every iteration for the truck t with minimal surrogate weight consists of two parts: At first (**for-loop**) we try to further reduce the load of truck t by moving items (without splitting them) to other trucks by a BF strategy. Then (in the **while-loop**) we iteratively take the largest (w.r.t. surrogate weight) remaining item i' in truck t and split it, i.e., we pack the largest possible fractional part of it, expressed as number of pallets, to one of the other trucks. Formally, the largest pallet number h' is calculated, such that h' pallets out of the $b_{i'}$ pallets constituting i' can be loaded on some truck t'. Ties among trucks are broken by a BF rule. The remaining part of item i' consisting of $b_{i'} - h'$ pallets is treated like an individual item in t. By moving the largest possible number of pallets, it can be expected that the total number of splits remains small. If no single pallet can be moved from truck t (i.e., $h' = 0$), the whole procedure is stopped. Otherwise, truck t is completely empty and can be removed.

5.5.2. Stage 5b.: Minimizing the number of split products

In this case it does not matter whether a large or small item i is split and whether i is split into two fragments or the maximum number of b_i fragments. It is intuitively clear that the largest potential for improvement is given by splitting large products completely into separate pallets. Therefore, in our Algorithm 9 we iterate over all items in decreasing order of b_i. Every considered item is split into separate

Algorithm 7 Multi-class heuristic

```
Compute LB.
**Phase 1:** {*pack refrigerated and standard items separately*}
Compute $z^F_{\text{heu}}$
Compute $z_{\text{refrig}} \leftarrow z^C_{\text{heu}}$ starting from the packing implied by $z^F_{\text{heu}}$
Compute $z^S_{\text{heu}}$    {*for empty trucks*}
$z_{\text{best}} \leftarrow z_{\text{refrig}} + z^S_{\text{heu}}$    {*current best solution*}
**if** $z_{\text{best}} = \text{LB}$ **then**
  **STOP 1**.
**end if**
**Phase 2:** {*reshuffle the refrigerated items and then add standard items*}
**Regroup**(all $z_{\text{refrig}}$ refrigerated trucks)
Compute $z^S_{\text{heu}}$ starting from the packing of the $z_{\text{refrig}}$ refrigerated trucks returned by **Regroup**.
$z_{\text{best}} \leftarrow \min\{z_{\text{best}}, z^S_{\text{heu}}\}$
**if** $z_{\text{best}} = \text{LB}$ **then**
  **STOP 1**.
**end if**
**Phase 3:** {*Iteration: expand the number of refrigerated trucks by distributing the load of one truck on two trucks and add standard items*}
Remove all standard items $I^S$ from the $z_{\text{refrig}}$ refrigerated trucks returned by **Regroup**.
**repeat**
  $z_{\text{current}} \leftarrow z_{\text{refrig}}$
  **for** all $z_{\text{current}}$ refrigerated trucks $t$ opened so far in increasing order of load w.r.t. surrogate weights **do**
    Open a new (empty) truck $z_{\text{refrig}} + 1$
    $z_{\text{refrig}} \leftarrow z_{\text{refrig}} + 1$
    Take all (refrigerated) items currently loaded on truck $t$ and pack them on the two trucks $t$ and $z_{\text{refrig}}$ by a WFD strategy.    {*gives a balanced bipartition*}
    Compute $z^S_{\text{heu}}$ starting from the current packing of the $z_{\text{refrig}}$ refrigerated trucks.
    $z_{\text{best}} \leftarrow \min\{z_{\text{best}}, z^S_{\text{heu}}\}$
    **if** $z_{\text{best}} = \text{LB}$ **then**
      **STOP 1**.
    **end if**
    **if** $z_{\text{refrig}} \geq \text{LB}$ **then**
      Compute $z_{\text{heu}}$    {*for empty trucks*}
      $z_{\text{best}} \leftarrow \min\{z_{\text{best}}, z_{\text{heu}}\}$
      **STOP 2**.
    **end if**
    Remove all standard items $I^S$ from the trucks
  **end for**
**until** false

Subprocedure **Regroup**($P_1, \ldots, P_T$)
Sort the $T$ trucks in increasing order of load w.r.t. surrogate weights
**for** $t = 1$ **to** $T$ **do**
  **for** all items $i$ in $P_t$ in decreasing order of surrogate weights **do**
    Try to pack item $i$ in trucks $\{P_{t+1}, \ldots, P_T\}$ by a BF strategy (w.r.t. surrogate weights)
  **end for**
**end for**
```

Algorithm 8 Multi-class heuristic: Stage 5a., minimizing the number of splits

```
Compute $\mathrm{LB}_1$
Execute Algorithm **Multi-class heuristic** returning $T := z_{\text{best}}$ trucks with loads $P_1, \ldots, P_T$
Sort the $T$ trucks in increasing order of load w.r.t. surrogate weights
$t \leftarrow 1$
**repeat**
  **for** all items $i$ in $P_t$ in decreasing order of surrogate weights **do**
    Try to pack item $i$ into another truck by a BF strategy (w.r.t. surrogate weights)
  **end for**
  **while** $P_t \neq \emptyset$ **do**
    Let $i'$ be the item in $P_t$ with largest surrogate weight
    {*find the truck $t'$ where the largest number of pallets of $i'$ can be added*}
    $h' \leftarrow \max_\tau\{h : h \cdot w_{i'}/b_{i'} + \sum_{j \in P_\tau} w_j \leq W,\ h + \sum_{j \in P_\tau} b_j \leq B, \tau \in \{1, \ldots, T\} \setminus \{t\}\}$ attained for $t'$
    **if** $h' = 0$ **then**
      {*Truck $t$ cannot be emptied*}
      Return the solution with $z_{\text{best}}$ trucks (as given at the start of the current iteration of **repeat**)
      **STOP**.
    **end if**
    Move $h'$ pallets of product $i'$ from $P_t$ to $P_{t'}$
  **end while**
  Remove $P_t$
  $z_{\text{best}} \leftarrow z_{\text{best}} - 1$
  **if** $z_{\text{best}} = \mathrm{LB}_1^I$ **then**
    Return current solution with $z_{\text{best}}$ trucks
    **STOP**.
  **end if**
  $t \leftarrow t + 1$
**until** false
```

Algorithm 9 Multi-class heuristic: Stage 5b., minimizing the number of split products

```
Compute $\mathrm{LB}_1$, $\mathrm{LB}_1^S$, $\mathrm{LB}_1^{C,F}$ and $\mathrm{LB}_1^{\text{mixed}}$
Sort all items $i \in I$ in decreasing order of $b_i$
$i \leftarrow 0$
**repeat**
  $i \leftarrow i + 1$
  Split item $i$ into $b_i$ separate items
  Execute Algorithm **Multi-class heuristic** returning $z_{\text{best}}$
**until** all lower bounds $\mathrm{LB}_1$, $\mathrm{LB}_1^S$, $\mathrm{LB}_1^{C,F}$ and $\mathrm{LB}_1^{\text{mixed}}$ are reached or $b_{i+1} = 1$ or $i = m$
Return $z_{\text{best}}$
```

pallets and then the multi-class heuristic from Section 5.4 is executed. The iterative process is stopped in any of the following three cases: (i) the multi-class heuristic stops with a solution matching all lower bounds for the objectives of Stages 1.–4.. Since we consider split products, we can only utilize the volume based lower bound LB_1; (ii) all products consisting of more than one pallet were split; (iii) all products were considered and split.

6. Computational results

In this section we will report computational results for all algorithms described before. The experiments are based on real-world instances and on more difficult instances generated from these by reducing their data to more difficult item sets (see Section 6.2). We will first report in Section 6.3 results for the heuristic approaches described above in Section 5. To analyze the performance of the algorithm we compare the heuristic with the exact solutions in detail. Then we will illustrate the performance of the exact BaP algorithm in Section 6.4. Furthermore, we also analyze in that section the impact of adding the heuristic solutions as columns to the initial RMP.

6.1. Details of the implementation

All heuristic algorithms presented in Section 5 are implemented and tested in Python 3.3.7 on a standard PC equipped with an Intel® Core^{TM} i5-3210M processor with 2.5 GHz and 4 GB of RAM. Note that the heuristics are also used in the daily planning task of our industrial partner. For this application they were originally implemented within the SAP ERP system of the company in the SAP-internal programming language ABAP. However, for test purposes we re-implemented the heuristics in Python.

The BaP algorithm described in Section 4 is implemented in C++ and compiled in release mode into a single-thread code under MS Visual Studio 2015. The experiments are conducted on a standard PC with an Intel® Core^{TM} i7-5930k processor clocked at 3.5 GHz and 64 GB of RAM. The RMP is solved at each column-generation iteration by means of CPLEX 12.9. Moreover, CPLEX is called after the solution of each branch-and-bound node as a primal MIP-based heuristic solver using the so far generated columns. CPLEX's default values are kept for all parameters. The time limit to solve each stage is set to 10 minutes (600 seconds). In case a stage cannot be solved within the time limit, the computation is stopped and following stages are not considered.

6.2. Benchmark instances

We consider 80 real-world instances with 35 to 430 items provided by our partner from the European food and beverage industry. The cardinality bound of each truck is 33 pallets (a common industry standard) and the weight capacity is 22.8 or 24.5 tons. The weight of each item is rounded up with an accuracy of 10 kg. Preliminary tests show that both constraints, weight and cardinality, can become active, i.e., there is no clear dominance of one of the two dimensions. Not all instances contain all types of products. Especially, frozen products often appear in rather small quantities or even miss completely. A detailed summary of our test instances can be found in Table 6.

Instance type	real-world	generated
number of instances	80	30
average number of items	144	153.9
average weight of items (in 10 kg)	93.7	114.6
average pallet number of items	3.8	6.7
number of instances with standard products	80	30
number of instances with cooled products	73	30
number of instances with frozen products	31	16

Table 6: Overview of benchmark instances.

During the computational study it turned out that for most of the instances the optimal solutions for Stages 1.–4. do not require any splitting and thus the objective of Stage 5. would be optimized by default. The reason for this behavior is that the instances also contain rather small items which fill up trucks without being split. Therefore, we distilled more difficult new instances from the given real-world instances avoiding this effect in the following way. The 80 real-world instances are divided into 10 groups each with 8 instances. For each group, we selected 5, 10 or 15 items with the highest weight and/or the highest number of used pallets of each instance forming a new instance, respectively. In total, we generated 30 new instances with up to 80, 160 or 240 items, respectively. All instances are available on the website `http://logistik.bwl.uni-mainz.de/benchmarks.php`.

6.3. *Results of the heuristic approach*

In this section, we report results of the heuristic approach. Table 7 shows the key results for the pure heuristic approach. For 56 out of the 80 real-world instances, all lower bounds for objectives 1.–4. are already achieved without splitting. The 24 remaining instances provide at least a theoretical improvement potential, which is exploited in 18 cases. For 13 of them, even the lower bounds for objectives 1.–4. are achieved if items are allowed to be split. While for the real-world instances exactly 70 % could be solved to the lower bounds for objectives 1.–4., this was possible only for 10 % (3 out of 30) of the generated instances, all without splitting.

However, the share of solutions that could be improved by applying splitting is far higher (25 out of 27) and the same is valid for the number of instances where that improvements even led to the lower bounds for objectives 1.–4. This higher improvement is not surprising, as the average pallet size of items is more than 75 % larger for generated instances and thus the effect of splitting items into separate pallets is much stronger. The higher difficulty of generated instances is also reflected in the running times of the heuristics.

For none of the 110 instances, the original Business Today logic outperformed the heuristic approach, but for 89 instances, the heuristics performed better. For the other 21 (all real-world) instances, both the heuristics and the Business Today logic reached the lower bounds.

Instance type	real-world	generated
number of instances	80	30
Business Today logic achieved *LB*	21	0
heuristics outperform Business Today logic	59	30
heuristics achieve *LB* without split	56	3
splitting improved heuristic solution	18	25
improvements led to reaching *LB*	13	21
no improvement by splitting	6	2
heuristic runtime $<$ 1 sec.	46	3
heuristic runtime 1-10 sec.	27	8
heuristic runtime 10-30 sec.	4	7
heuristic runtime $>$ 30 sec.	3	12
average number of trucks Business Today	23.1	45.8
average number of trucks heuristic unsplit	22.2	42.2
average number of trucks heuristic split	22.1	40.6

Table 7: Results of the heuristic approach.

For a further performance analysis, we compare the objective values of the heuristic approach with the optimal objective values computed by the BaP approach. Detailed results can be found in Table 8. The table entries have the following meaning:

#opt:	number of instances solved to proven optimality at the specific stage;
$z_H = z^*$**:**	number of instances in which the heuristic approach finds the optimal solution; instances with non-optimal heuristic solutions in former stages are not considered;
$z_H > z^*$**:**	number of instances in which the heuristic approach does not find the optimal solution; instances with non-optimal heuristic solutions in former stages are not considered;
gap$_H$**:**	average gap between heuristic and exact solution, we use the gap $z_H - z^*$ instead of the relative gap because the optimal solution can be $z^* = 0$ in Stages 4.–5.; recall that solution values for Stages 1.–4. represent number of trucks, but split parameters for Stage 5.
#implicit:	number of instances for which the heuristic and optimal solution implicitly coincide, or where the current stage is not active (e.g., no Stage 3. for instances without any frozen items)
Σ**-**$z_H = z^*$**:**	total number of instances in which the heuristic approach finds the optimal solution up to the specific stage: $(\Sigma\text{-}z_H = z^*) := (z_H = z^*)+(\#\text{implicit})$.

For a better understanding, we describe the entries in the second row of Table 8 in detail. At Stage 2., the branch-and-price algorithm solves 64 instances to proven optimality. For 60 (3) instances, the heuristic approach could (not) find the optimal solution. For one instance, we cannot compare the heuristic and exact approach at Stage 2. because the solution at Stage 1. differs for this instance. For 6 instances without any refrigerated items, the heuristic approach finds the optimal solution at Stage 1. so that Stage 2. is solved implicitly. Including instances without refrigerated products, the heuristic and exact solution coincide for in total 66 instances.

splitting	class	Stage	#opt	$z_H = z^*$	$z_H > z^*$	gap$_H$	#implicit	Σ-$z_H = z^*$
forbidden	real-world	1	76	71	5	0.1	0	71
		2	64	60	3	<0.1	6	66
		3	26	24	1	<0.1	41	65
		4	61	52	4	0.3	6	58
	generated	1	30	15	15	0.6	0	15
		2	22	9	5	0.9	0	9
		3	11	4	0	<0.1	5	9
		4	18	6	2	1.3	0	6
allowed	real-world	1	73	73	0	0.0	0	73
		2	64	64	0	0.0	7	71
		3	27	27	0	0.0	44	71
		4	60	60	0	0.0	7	67
5a.		5	62	50	6	98.3	0	50
5b.		5	63	48	6	3.9	0	48
	generated	1	28	28	0	0.0	0	28
		2	27	25	2	0.2	0	25
		3	15	15	0	0.0	10	25
		4	25	24	0	<0.1	0	24
5a.		5	7	3	4	80.8	0	3
5b.		5	8	3	5	32.2	0	3
	Total		767	661	58	9.5	126	

Table 8: Comparison between the heuristic and the exact solution.

For the policy splitting forbidden, the heuristic solution is optimal for 58+6 of 68+18 instances. Interestingly, in most of the cases, the heuristic approach fails to find the optimal solution at the first stage. Especially for the generated instances, only 15 of 30 instances are solved to optimality. For the policy splitting allowed, the heuristic approach performs well with 67+24 of 67+25 instances solved optimally up to Stage 4. The absolute deviance gap_H is low up to Stage 4. with a maximum of 0.2, but the values are big for Stage 5. With 6/6+4/5 of 62/63+7/8 non-optimal instances, the heuristics fail for most of the instances at Stage 5.

6.4. Results of the branch-and-price algorithm

In this section, we report the results of our BaP algorithm. In general, reaching a stage of the lexicographic optimization task triggers the execution of a BaP algorithm solving the respective model (1)–(4), and possibly (5a)–(5b) or (5c)–(5d). However, for some instances, not all stages become *active*, e.g., if there are no frozen products in the instance or if a stage is solved implicitly during the preceding stage. Therefore, the total number of instances per stage to be reported in the computational experiments may be different for every stage. Moreover, the different BaP algorithms of the stages may have different success rates and thus will lead to a different number of instances remaining for the successive stages. This makes the comparison of different algorithmic approaches a delicate task. The following analyses may, therefore, seem somewhat less transparent but our focus is on the correct interpretation of the results we obtained.

The construction of the initial RMP works as follows. As the first stage is a vector packing problem without additional constraints, we start with the same initial RMP as described in (Heßler *et al.*, 2018) by adding unit vectors, columns with only one non-negative coefficient $a_i^p = u_i$, and 200 additional columns resulting from variants of the FF and BF heuristics. At Stage 2., we add other 200 columns resulting from the FF and BF heuristics that first assign items to bins that already contain the same product type. Moreover, the same FF and BF variants used in Stage 1. are performed on restricted item sets by considering standard or refrigerated products separately. Analogously, Stage 3. applies the FF and BF variants for cooled or frozen products separately. At Stage 4.–5., no additional columns are added initially to the RMPs.

For each stage, the computation time is restricted to a maximum of 10 minutes (600 seconds). If a stage cannot be solved within the time limit, we cannot solve the instance exactly. Therefore, the computation is stopped at this stage and the following stages are not considered. If an instance has no frozen or no refrigerated products at all, Stage 3. or Stages 2.–4. are skipped, respectively.

At first, the BaP algorithm is tested employing the results of the heuristic approach, i.e., the corresponding columns are added to the initial RMP. We refer to this variant as `base`. To analyze the impact of the heuristic solution, we compare the `base` variant to a variant without using the heuristic solution. Moreover, the impact of the lower bounds enforced by constraints (2c), (3c), and (4c), and the graph reduction presented in Section 4.5 are tested. We refer to these variants as `without heuristic solution`, `without lb`, and `without reduced graph`, respectively. The results are summarized in Table 9. The table entries have the following meaning:

class:	class of instances; real-world or self-generated; for details see Section 6.2;
splitting:	allowed or forbidden; in case of splitting allowed, either the number of splits (objective 5a.) or the number of split products (objective 5b.) is minimized, see Section 4.1;
#inst:	number of active instances;
#inst_cons:	number of active instances considered at the current stage; instances that are not solved to proven optimality in a previous stage are not considered;
#opt:	number of instances (out of #inst_cons) solved to proven optimality at the specific stage within 10 minutes (600 seconds);
#implicit:	number of instances for which the current stage is already implicitly solved by the result of a previous stage, or where the current stage is not active (e.g., at Stage 3. for instances without any frozen items);
Σ-#opt:	number of instances solved to proven optimality at the current stage: Σ-#opt:= #opt+#implicit
$\bar{T}$**:**	average computation time in seconds over #inst_cons instances; unsolved instances are taken into account with the time limit of 10 minutes (600 seconds);
$\bar{T}_{LP}$**:**	average computation time of the linear relaxation in seconds; unsolved linear relaxations are taken into account with the time limit of 10 minutes (600 seconds);
gap:	average gap between the upper and lower bound; we use the gap $UB - LB$ instead of the relative gap because the objective value can be zero at Stages 4.–5.

We summarize and interpret the results as follows: For the policy splitting forbidden, 68 of 80 real-world and 18 of 30 self-generated instances are solved to optimality. In particular, the algorithm performs well at Stage 3. with an average computation time of around one minute. In comparison, the policy splitting allowed is more difficult than splitting forbidden. The performance at Stage 1. is similar for both policies with 101 solved instances (splitting allowed) instead of 106 (splitting forbidden). Up to Stage 4., 67 of the 80 real-world and 25 of the 30 generated instances are solved which is $(-1) + 3$ instances more compared to the policy splitting forbidden. The performance of the algorithm at Stage 5., where 62/63+7/8 instances are solved, is almost the same for both objectives 5a. and 5b. with similar average solution time. As can be expected, Stage 5. is particularly difficult for the generated instances. Especially, the average deviation (gap) at Stage 5a. is very high because in many cases the root node cannot be solved or the solution of the preceding stage is poor.

To analyze the impact of the heuristic solution, we compare the variants `base` and `without heuristic solution`. For the policy splitting forbidden, the results hardly change when the initial columns from the heuristic are omitted. In total, only one instance less can be solved and the average running times differ by less than 5 seconds. Therefore, the use of the heuristic solution provides only a small advantage for the policy splitting forbidden. In contrast, for the policy splitting allowed, the effect of adding the heuristic solution is more significant, as it helps to solve 6/7+2/3 additional instances, respectively.

Adding the additional constraints (2c), (3c), and (4c) to ensure lower bounds on the number of refrigerated, frozen, and mixed trucks, respectively, has a strong impact on the performance of the BaP algorithm. Without adding these lower bounds, only 45+13 instances of the policy splitting forbidden and 38/39+3 instance of the policy splitting allowed can be solved. In total, around one-third of the instances cannot be solved without these bounds. Moreover, the running time increases significantly for all stages and problem variants.

For the policy splitting allowed, using a reduced graph as described in Section 4.5 has a positive impact for the Stages 1.–4. Already at Stage 1., 10+9 instances less are solved if the graph reduction is omitted. Overall, 8/9+1/2 instances less are solved.

				base							without heuristic solution							without lb							without reduced graph						
splitting	class	Stage	#inst	#inst_cons	#opt	#implicit	Σ-#opt	$\bar{T}$	$\bar{T}_{LP}$	gap	#inst_cons	#opt	#implicit	Σ-#opt	$\bar{T}$	$\bar{T}_{LP}$	gap	#inst_cons	#opt	#implicit	Σ-#opt	$\bar{T}$	$\bar{T}_{LP}$	gap	#inst_cons	#opt	#implicit	Σ-#opt	$\bar{T}$	$\bar{T}_{LP}$	gap
forbidden	real-world	1	80	80	76	0	76	65.7	30.1	0.1	80	**77**	0	**77**	59.1	32.6	<0.1	80	75	0	75	91.4	58.9	0.1							
		2	73	69	**64**	7	**71**	79.8	21.3	0.9	70	62	7	69	99.9	21.3	1.6	68	63	7	70	136.2	88.8	0.6							
		3	31	26	**26**	45	**71**	61.4	40.0	0.0	25	25	44	69	58.3	38.0	0.0	26	25	44	69	101.9	68.4	0.3							
		4	73	64	**61**	7	**68**	65.3	32.6	0.1	62	59	7	66	69.6	23.4	<0.1	62	38	7	45	261.3	34.5	0.4							
	generated	1	30	30	**30**	0	**30**	47.2	40.2	0.0	30	**30**	0	**30**	49.5	39.9	0.0	30	**30**	0	**30**	80.2	73.6	0.0							
		2	30	30	22	0	22	192.4	33.7	3.3	30	**23**	0	**23**	186.4	33.1	2.5	30	**23**	0	**23**	187.2	79.9	4.5							
		3	16	11	11	11	22	22.3	13.1	0.0	12	**12**	11	**23**	45.6	16.6	0.0	11	11	12	**23**	28.0	20.0	0.0							
		4	30	22	18	0	18	132.6	18.0	0.2	23	**19**	0	**19**	130.9	18.8	0.2	23	13	0	13	283.4	29.3	0.5							
allowed	real-world	1	80	80	**73**	0	**73**	131.4	107.9	1.4	80	71	0	71	139.7	106.5	1.4	80	**73**	0	**73**	131.3	107.9	1.4	80	63	0	63	235.8	225.0	6.5
		2	73	66	**64**	7	**71**	114.8	75.8	0.5	64	61	7	68	117.8	75.7	1.2	66	61	7	68	179.8	147.0	0.9	56	55	7	62	94.3	57.3	0.1
		3	31	27	**27**	44	**71**	170.5	139.5	0.0	25	21	43	64	227.2	137.9	0.4	24	19	44	63	230.6	213.5	0.5	23	22	39	61	141.3	119.6	0.2
		4	73	64	**60**	7	**67**	140.3	95.5	0.2	57	52	7	59	144.6	81.8	0.1	56	34	7	41	285.0	61.1	0.4	54	51	7	58	107.9	69.5	0.2
	5a.	5	80	67	**62**	0	**62**	157.3	91.2	5.8	59	56	0	56	140.4	79.1	8.4	41	38	0	38	140.9	68.7	7.3	58	54	0	54	137.2	74.2	12.6
	5b.	5	80	67	**63**	0	**63**	148.3	88.2	1.4	59	56	0	56	139.1	87.1	4.0	41	39	0	39	122.4	64.3	1.7	58	54	0	54	121.8	58.9	1.6
	generated	1	30	30	**28**	0	**28**	181.6	159.2	5.4	30	22	0	22	226.9	175.9	9.0	30	29	0	29	169.8	146.0	3.0	30	19	0	19	289.4	282.3	19.8
		2	30	28	**27**	0	**27**	192.5	143.1	0.8	22	11	0	11	355.2	139.2	10.2	29	24	0	24	312.1	276.1	2.2	19	17	0	17	197.6	162.3	1.9
		3	16	15	**15**	12	**27**	240.3	201.6	0.0	6	5	5	10	231.1	121.0	0.2	13	11	11	22	280.2	245.7	0.4	9	9	8	17	201.0	172.4	0.0
		4	30	27	**25**	0	**25**	229.4	175.9	0.4	10	10	0	10	124.2	112.8	0.0	22	13	0	13	339.3	153.2	0.6	17	16	0	16	190.2	157.6	0.2
	5a.	5	80	25	**7**	0	**7**	465.8	317.3	347.3	10	5	0	5	360.3	249.2	177.6	13	3	0	3	497.6	366.9	275.7	16	6	0	6	413.2	382.4	336.9
	5b.	5	80	25	**8**	0	**8**	459.2	320.5	32.0	10	5	0	5	399.1	251.9	39.1	13	3	0	3	498.0	366.9	23.5	16	6	0	6	441.8	384.9	19.1
	Total		1046	853	**767**	140	**907**	127.9	78.5	12.3	764	682	131	813	133.6	73.0	5.0	758	625	139	764	188.8	106.9	6.4	436	372	61	433	182.1	145.6	17.6

Table 9: Results of the exact BaP-based approach.

7. Conclusion

Packing pallets of products from industrial producers into trucks for shipment to wholesalers is an important daily task for supply chain management. Reducing the number of trucks, even if only by one, or rearranging the package plan to diminish the number of the more costly trucks carrying cooled or frozen products, gives a valuable impact on costs as well as emissions, since the supply operation is usually performed daily or several times a week. Thus, the effort spent on optimizing the packing operation yields savings that will add up over a year to numbers of considerable economic relevance.

In this contribution, we consider a real-word packing problem where products should be packed into trucks such that a five-level lexicographic objective function is minimized. Feasibility is defined by a weight and a volume (=number of pallets) constraint for every truck. This special variant of a multi-objective optimization problem considers the total number of required trucks as main objective dominating all other goals due to the high cost of labor. However, after fixing the overall number of trucks (and thus the number of required personal) secondary objectives come into play, which concern the cost consuming operations of cooling and freezing devices of a truck. Finally, for practical handling it is clearly convenient for the customer to receive all pallets carrying the same product in the same truck. But if the total number of trucks could be reduced, the customer will agree to a splitting of products onto several trucks. Nevertheless, such a splitting of products should be kept as low as possible, which opens the question how to measure the inconvenience of splitting. We believe that such a lexicographic setting has not be considered for one- or two-dimensional bin-packing problems before.

The first part of this paper gives a branch-and-price framework for computing exact solutions of our problem. While the upper-level problem of minimizing the number of trucks can be modelled as a two-dimensional vector packing problem, with a special structure in one dimension, the lexicographic objective requires five iterative steps of successive optimization problems, in each of which columns are generated by solving appropriate instances of a shortest path problem with resource constraints (SPPRC). It is a major contribution of the paper to put the five different problems arising for the different levels into a uniform framework of column generation. Furthermore, advanced concepts of stabilization and some acceleration techniques are employed.

In the second part, we describe the heuristic solution method previously employed in practice and then develop a number of new constructive heuristics for solving a single level of the packing problem. These try to balance the two constraints and generate solutions with a favorable mix of weight and volume. Then, the single-level heuristics are integrated into a more complex heuristic framework for building solutions of high quality with respect to the multi-level lexicographic objective function.

Computational experiments for real-world benchmark instances as well as structurally more difficult instances extracted from these show that the branch-and-price algorithm is highly effective in computing optimal solutions. Within 10 minutes of computation time it reaches proven optimality for 68 out of 80 real-world scenarios without splitting products. Allowing splitting leads to more difficult problems, but still 62 of 80 are solved to optimality.

It also turns out that the heuristic framework performs much better than the approach previously applied in practice and matches the optimal solutions in 86 % of all subproblems. Note that it is also very helpful to use the heuristic solution as a starting column for the branch-and-price algorithm. Based on this evaluation, our heuristic framework has been successfully incorporated into the SAP ERP software system of our industrial partner and is currently used for the daily planning process.

In the future, it would be interesting to extend our single-source single-sink shipping problem to a more complex supply network serving multiple customers. This would open up the possibility of delivering less-than-truckload amounts to one customer and using the residual capacity of a truck for serving a second (or third) customer, if the routing costs support such a decision.

Acknowledgements

We would like to thank *scc EDV-Beratung AG, Vienna,* and its industrial partner for the fruitful collaboration and for providing all the practical insights.

Ulrich Pferschy was supported by the Field of Excellence “COLIBRI” at the University of Graz.

References

Aringhieri, R., Duma, D., Grosso, A., and Hosteins, P. (2018). Simple but effective heuristics for the 2-constraint bin packing problem. *Journal of Heuristics*, **24**(3), 345–357.

Bansal, N., Eliáš, M., and Khan, A. (2016). Improved approximation for vector bin packing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 3, pages 1561–1579. SIAM.

Ben Amor, H., Desrosiers, J., and Valério de Carvalho, J. M. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, **54**(3), 454–463.

Bertazzi, L., Golden, B., and Wang, X. (2019). The bin packing problem with item fragmentation: A worst-case analysis. *Discrete Applied Mathematics*, **261**, 63–77.

Brandão, F. and Pedroso, J. P. (2016). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, **69**, 56–67.

Caprara, A. (1998). Properties of some ILP formulations of a class of partitioning problems. *Discrete Applied Mathematics*, **87**(1-3), 11–23.

Caprara, A. and Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, **111**(3), 231–262.

Caprara, A., Kellerer, H., and Pferschy, U. (2003). Approximation schemes for ordered vector packing problems. *Naval Research Logistics*, **50**(1), 58–69.

Casazza, M. (2019). New formulations for variable cost and size bin packing problems with item fragmentation. *Optimization Letters*, **13**(2), 379–398.

Casazza, M. and Ceselli, A. (2016). Exactly solving packing problems with fragmentation. *Computers & Operations Research*, **75**, 202–213.

Christensen, H. I., Khan, A., Pokutta, S., and Tetali, P. (2017). Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, **24**, 63–79.

Coffman Jr, E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. (2013). Bin packing approximation algorithms: survey and classification. *Handbook of combinatorial optimization*, pages 455–531.

Côté, J.-F. and Iori, M. (2018). The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing*, **30**(4), 646–661.

Delorme, M. and Iori, M. (2020). Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, **32**(1), 101–119.

Delorme, M., Iori, M., and Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, **255**(1), 1–20.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Garey, M. R., Graham, R. L., and Ullman, J. D. (1972). Worst-case analysis of memory allocation algorithms. In *Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 143–150. ACM.

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

Gschwind, T. and Irnich, S. (2016). Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, **28**(1), 175–194.

Henke, T. (2018). *Multi-compartment vehicle routing problems in the context of glass waste collection*. Ph.D. thesis, Otto-von-Guericke University Magdeburg, Magdeburg, Germany.

Heßler, K., Gschwind, T., and Irnich, S. (2018). Stabilized branch-and-price algorithms for vector packing problems. *European Journal of Operational Research*, **271**(2), 401–419.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, chapter 2, pages 33–65. Springer.

Johnson, D. S. (1973). *Near-Optimal Bin Packing Algorithms*. Ph.D. thesis, Massachusetts Institute of Technology.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer, Berlin.

LeCun, B., Mautor, T., Quessette, F., and Weisser, M.-A. (2015). Bin packing with fragmentable items: Presentation and approximations. *Theoretical Computer Science*, **602**, 50–59.

Lodi, A., Martello, S., and Vigo, D. (2002). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, **123**(1-3), 379–396.

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

Malaguti, E., Monaci, M., Paronuzzi, P., and Pferschy, U. (2019). Integer optimization with penalized fractional values: The Knapsack case. *European Journal of Operational Research*, **273**(3), 874–888.

Mandal, C. A., Chakrabarti, P. P., and Ghose, S. (1998). Complexity of fragmentable object bin packing and an application. *Computers & Mathematics with Applications*, **35**(11), 91–97.

Martello, S. and Toth, P. (1990). Bin-packing problem. In *Knapsack problems: Algorithms and Computer Implementations*, Wiley Series in Discrete Mathematics and Optimization, pages 221–245. Wiley.

Martello, S. and Toth, P. (2003). An exact algorithm for the two-constraint 0–1 knapsack problem. *Operations Research*, **51**(5), 826–835.

Pollaris, H., Braekers, K., Caris, A., Janssens, G. K., and Limbourg, S. (2014). Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum*, **37**(2), 297–330.

Spieksma, F. C. (1994). A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers & Operations Research*, **21**(1), 19–25.

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

Valério de Carvalho, J. M. (2005). Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, **17**(2), 175–182.

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

Wei, L., Luo, Z., Baldacci, R., and Lim, A. (2019). A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*. https://doi.org/10.1287/ijoc.2018.0867.

Wei, L., Lai, M., Lim, A., and Hu, Q. (2020). A branch-and-price algorithm for the two-dimensional vector packing problem. *European Journal of Operational Research*, **281**(1), 25–35.