Omega 125 (2024) 103039

Contents lists available at ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

Exact decomposition approaches for a single container loading problem with stacking constraints and medium-sized weakly heterogeneous items☆

Maxence Delorme *, Joris Wagenaar

Department of Econometrics and Operations Research, Tilburg University, The Netherlands

ARTICLE INFO

Keywords:
Container loading problem
Operations research applications
Exact algorithms
Integer linear programming
Benders' decomposition

ABSTRACT

We consider a real-world three-dimensional container loading problem in which the objective is to maximize the volume of the items packed into a single vehicle. While container loading problems have been extensively studied in the literature, our case study displays a set of item features (rotation, medium-sized dimensions, stackability, weak heterogeneity) that was not often considered together in previous research papers. In fact, we show that some of these features can be exploited in an innovative way to derive more effective exact algorithms. We first describe a compact integer programming model to solve the problem exactly together with a number of ad hoc reduction procedures and modeling tricks to enhance the empirical performance of the model. We then present a sequential approach where one generates item columns in a first stage and then solves a two-dimensional knapsack problem afterwards and show that each of the two components is $\mathcal{NP}$-hard. Thereafter, we introduce a set of exact algorithms based on Benders' decomposition. We identify three ways to split the problem decisions (deciding if an item should be packed, in which column, in which x-coordinate, and in which y-coordinate) into the classical master-subproblem framework and we observe trough an extensive set of computational experiments on both real and randomly generated instances that not all decomposition methods are as competitive as the others. We conclude our work by showing how relevant extensions of the problem related to item fragility and customer visit order can be taken into account. Overall, this work aims at establishing a bridge between the theoretical field of two-dimensional packing problems and the more practical field of container loading problems.

1. Introduction

Third-party logistic service providers are responsible for door-to-door deliveries of goods using vehicles. Each day, there is a set of goods that needs to be transported from the service provider to one or several customers. In this work, we are interested in a practical case originating from a Dutch company, Nunner Logistics, in which the packing layout of a single truck needs to be optimized.

The problem faced by Nunner Logistics can be summarized as follows. In the early morning, the company is given a set of pallets (also called "items" in the following) that needs to be delivered to customers with a single vehicle (also called "truck" in the following). As there is only one delivery per day, the objective is to maximize the space occupied by the pallets in the daily packing layout. In practice, it often happens that every pallet fits within the truck. As far as packing layout is concerned, Nunner first stacks the pallets on top of each other to create item columns that are then positioned into the truck before the vehicle starts its daily delivery route.

This problem, which is known as the single container loading problem in the literature, has been extensively studied by the research community. Nevertheless, as observed by Bischoff and Ratcliff [1], existing approaches to container loading problems cannot always be modified to take into account ad hoc constraints. The authors also outlined that there were still "scenarios" (i.e., real-world problems) for which no solution methods were proposed. Whereas the work of Bischoff and Ratcliff [1] is now almost thirty years old, the observations made by the authors are still relevant today, especially when it comes to exact algorithms: those are very scarce in the container loading literature and the few existing approaches cannot easily be extended to Nunner's case.

Among the features that are specific to Nunner, we mention that: (i) the items can be rotated by 90° (i.e., the width and the length of an item are interchangeable, but not the height), (ii) the items are relatively large when compared to the vehicle (the truck can accommodate around 15 × 3 columns of items), (iii) the items are weakly

☆ Area - Production Management, Scheduling and Logistics. This manuscript was processed by Associate Editor Yagiura.

* Corresponding author.
E-mail address: m.delorme@tilburguniversity.edu (M. Delorme).

https://doi.org/10.1016/j.omega.2024.103039
Received 24 August 2023; Accepted 19 January 2024
Available online 1 February 2024

heterogeneous on some dimensions (an instance typically contains 5 or 6 distinct width–length couples) but not on the others (the item heights and weights are more heterogeneous), (iv) only one item per layer is allowed in a column, and (v) the item stackability (i.e., whether an item can be stacked on top of another) can be fully captured in a compatibility graph. In this work, we show how some of these features can be exploited in order to derive exact algorithms able to solve the daily instances faced by the company.

The main contributions of this paper are the following:

- We describe the optimization problem faced by Nunner Logistics, formalize it with a mathematical model, and provide a number of reduction procedures and modeling tricks to enhance the empirical performance of the model. In particular, we show how to exploit the fact that many items share the same widths and lengths and we introduce a stronger preprocessing based on integer programming aimed at simultaneously enlarging the item dimensions and reducing the truck size. The latter strategy works particularly well when the maximum number of items that can fit side by side into the truck is relatively small.
- We describe a sequential approach where the first phase consists in creating columns of items and the second phase consists in loading the columns into the truck. We demonstrate that the decision version of the two problems are $\mathcal{NP}$-complete and outline the resemblance between the first phase and the well-known kidney exchange problem.
- We introduce three Benders' decomposition approaches in which the master problem creates a set of columns with maximum utilized space whereas the subproblem checks whether the selected columns fit within the container. The difference between the three approaches lies in the way in which the problem decisions (i.e., deciding if an item should be packed, in which column, in which x-coordinate, and in which y-coordinate) are split between the two problems.
- We empirically evaluate each of the proposed methods on both real instances provided by the company and randomly generated instances based on similar real-world characteristics.
- We show how supplementary real-world features such as customer multiplicity and another type of fragility constraints can be integrated into the proposed approaches.

This work aims at extending the successful approaches recently proposed for theoretical two-dimensional packing problems [2–4] to the more practical field of container loading problems.

The rest of the paper is organized as follows. Section 2 reviews the relevant literature and Section 3 formally defines the problem and provides an initial mathematical formulation. Section 4 introduces reduction procedures and modeling tricks to decrease the size of the mathematical model and Section 5 describes a sequential approach inspired by the approach used by the company and outlines the resemblance of one of its components with the kidney exchange problem. Section 6 introduces three Benders' decomposition methods aimed at solving the problem exactly and Section 7 gives a summary of the results obtained by extensive computational experiments on real and random instances in addition to showing how supplementary real-world features could be taken into account. Finally, concluding remarks are provided in Section 8.

2. Literature review

The container loading problem is one of the cutting and packing (C&P) problems that has been the most extensively studied in the literature. We refer the reader to the works of Silva et al. [5] and Zhao et al. [6] for two recent surveys on the topic. In a standard container loading problem, one wants to pack three-dimensional items (such as a product or a pallet) into three-dimensional rectangular containers (such as a truck or a ship) in order to optimize an objective function (such as minimizing the unused container space or maximizing the profit of the items packed) while satisfying a given set of constraints. Some of these constraints are shared by most (if not all) container loading problems: this is the case of non-overlapping constraints (two items cannot occupy the same space within the container) and capacity constraints (the items must fit entirely within the container). There are also problem-specific constraints (see [7] for a comprehensive review on the topic) such as stacking constraints (a heavy item cannot be packed on top of a fragile item) and conflict constraints (some items cannot be packed together in the same container). In the following, we review the C&P literature that is the most relevant to our study.

When only non-overlapping and capacity constraints are taken into account, the container loading problem is similar to the well-known three-dimensional bin packing problem (or three-dimensional knapsack problem, depending on the problem characteristics). We refer the reader to the recent work of Silva et al. [8] for a comparative study of exact approaches on three-dimensions packing problems. The algorithms evaluated in [8] cannot always be extended to incorporate problem-specific constraints, and even when they can, the size of the instances encountered in real-world cases is often unsuitable for exact methods. This is why older [9–11] and more recent approaches [12–14] aimed at solving real-world container loading problems belong to the category of (meta)heuristics. Nevertheless, a few research papers [15, 16] proposed exact approaches for solving targeted container loading problems.

Even though adding a practical constraint to a C&P problem such as enforcing the packing to be stable or balanced [16,17] usually makes the problem more difficult, there are some cases in which requiring the packing to have certain properties allows for more effective solution methods (e.g., enforcing the cutting patterns to be guillotinable [18]). At Nunner logistics, items are first stacked into columns before being positioned into the truck and every layer in a column is composed of exactly one item. Such a requirement (that shares some similarities with one-stage guillotinability) can be exploited in order to derive simpler optimization algorithms. One common strategy when dealing with container loading problems with stacked columns is to decompose the problem into two components and solve them sequentially. For example, Haessler and Talbot [19] developed a heuristic procedure that was implemented at a large U.S.-based company in which the items where first stacked into columns and then loaded into the truck. Gehring and Bortfeldt [20] used a similar strategy in their genetic algorithm and enriched the loading component by including a number of practical constraints. Toffolo et al. [21] solved a more complex version of the problem in which several items could be packed into the same layer. The authors generated a set of initial solutions through heuristics and improved the resulting packing layouts with a local search algorithm.

Whereas such a sequential strategy cannot return a certificate of optimality (even if each component is solved exactly), there exists more advanced frameworks such as Benders' decomposition that can. To the best of our knowledge, Benders' decomposition has not yet been used to solve the container loading problem with stacked columns, but it was used to solve other container loading problems. For example, in the recent work of Do Nascimento et al. [22], the first component decides the set of items that should be packed into the container and the second component checks whether or not there exists a feasible packing of the selected items that satisfies the loading constraints. Polyakovskiy and M'Hallah [23] used a similar strategy to solve an extension of the two-dimensional bin packing problem.

Decomposition methods were also proven to be very effective when solving two-dimensional C&P problems. We mention in particular the works of Côté et al. [2] and Delorme et al. [4] who used such methods to solve the strip packing problem, where the first component decides the strip height and a suitable x-coordinate for every item and the second component checks whether or not the packing can be completed with a suitable y-coordinate for every item. We also mention the relevant work of Côté et al. [3] who used a similar strategy to solve the two-dimensional bin packing problem. In this work, we show that the

aforementioned Benders' decomposition approaches can be extended to tackle the three-dimensional packing problem faced by Nunner by intelligently modeling the stacking component of the problem and exploiting the item multiplicity.

3. Problem definition

We are given a set of n three-dimensional items to be delivered to a unique customer using one truck. The truck has width W, length L, height H, and capacity K (the maximum weight the truck can handle). Every item i ($i = 1 \dots, n$) has width w_i, length l_i, height h_i, weight k_i, and profit p_i. Without loss of generality, we assume that w_i, l_i, h_i ($i = 1 \dots, n$), W, L, and H are integers. Note that, if this is not the case, one can multiply each dimension by a sufficiently large coefficient. In our application, the profit of an item is equal to its area (i.e., $p_i = w_i \cdot l_i$). As items can be rotated by 90 degrees (i.e., the width and the length of an item are interchangeable), each of the n items is duplicated once to represent its rotated version. Therefore, item $n+i$ is the rotated version of item i ($i = 1 \dots, n$).

We say that an item is *packed* in coordinate (w, l) if the bottom left corner of the item is located at abscissa w and ordinate l of the truck. All the points (w, l) in which item i ($i = 1 \dots, 2n$) can be packed are gathered in set $\mathcal{P}_i$. A trivial definition of $\mathcal{P}_i$ includes every point (w, l) such that $w = 0, \dots, W - w_i$ and $l = 0, \dots, L - l_i$. We say that item i ($i = 1 \dots, 2n$) *occupies* (w, l) if a portion of the item fills the unit length square with bottom left corner (w, l), or in other words, if the item is packed in point (w', l') such that $0 \leq w - w_i + 1 \leq w' \leq w$ and $0 \leq l - l_i + 1 \leq l' \leq l$. All the points (w', l') for which item i would occupy (w, l) if the item was packed in (w', l') are gathered in set $\mathcal{P}_i^{(w,l)}$.

As some items can be *stacked* on top of other items, we also consider a compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where vertex set $\mathcal{V} = \{1, \dots, 2n\}$ contains one node per item and where arc set $\mathcal{A}$ contains arc (i, i') if item i' can be stacked on top of item i. In practice, item i' can be stacked on top of i ($i = 1, \dots, 2n$, $i' = 1, \dots, 2n$) if the six following conditions are met: (i) the width of i is larger than or equal to the width of i', (ii) the length of i is larger than or equal to the length of i', (iii) the weight of i is larger than or equal to the weight of i', (iv) both i and i' are stackable (a binary information related to the items that is only used for the purpose of stackability), (v) the surface of i is *flat* (another binary information related to the items that is also only used for the purpose of stackability), and (vi) i' is neither i nor the rotated version of i. One can observe that due to such requirements, there is a form of symmetry in the compatibility graph that makes the information about the item rotation unnecessary (indeed, if item i' can be stacked on top of item i, then the rotated version of item i' can be stacked on top of the rotated version of item i). We can therefore merge nodes i and $n + i$ ($i = 1, \dots, n$) in graph $\mathcal{G}$. One may reduce symmetry even further by not allowing any cycle in the compatibility graph. This can easily be achieved when constructing $\mathcal{A}$ by not adding arc (i', i) if the arc set already contains arc (i, i'). For modeling purposes, we define parameter $c_{ii'}$ equals to 1 if $(i, i') \in \mathcal{A}$ and 0 otherwise.

A set of items packed on top of each other is called a *stacked column*. An item located at the bottom of a stacked column is called a *bottom item*. We define the index of a column by the index of its bottom item. We say that an item is packed in *position* s of stacked column i if it is stacked on top of $s - 1$ items in the column originating from item i. Note that if stacked column i exists, then item i is packed in position 1 of the stacked column. An item packed in position $s \geq 2$ of a stacked column is called a *stacked item*. For practical purposes, at most S items can belong to the same stacked column. The mathematical notation is summarized in Table 1.

By introducing binary decision variables ξ_i^P that take value 1 if item i ($i = 1, \dots, n$) is a bottom item and 0 otherwise, ξ_i^S that take value 1 if item i ($i = 1, \dots, n$) is a stacked item and 0 otherwise, binary decision variables x_{iwl} that take value 1 if the bottom left corner of item i ($i = 1, \dots, 2n$) is packed at abscissa w and ordinate l $\left((w, l) \in \mathcal{P}_i\right)$, and 0 otherwise, and binary decision variables $y_{ii's}$ that take value 1 if item i' ($i' = 1, \dots, n$) is packed in position s ($s = 2, \dots, S$) in stacked column i ($i = 1, \dots, n$), and 0 otherwise, the problem can be modeled as follows:

$$\max \quad \sum_{i=1}^{n} p_i \xi_i^P + \sum_{i=1}^{n} p_i \xi_i^S \tag{1}$$

$$\text{s.t.} \quad \xi_i^P = \sum_{(w,l) \in P_i} x_{iwl} + \sum_{(w,l) \in P_{n+i}} x_{n+i,w,l} \qquad i = 1, \dots, n, \tag{2}$$

$$\xi_i^S = \sum_{i'=1}^{n} \sum_{s=2}^{S} y_{i'is} \qquad i = 1, \dots, n, \tag{3}$$

$$\xi_i^P + \xi_i^S \leq 1 \qquad i = 1, \dots, n, \tag{4}$$

$$\sum_{i=1}^{n} k_i \xi_i^P + \sum_{i=1}^{n} k_i \xi_i^S \leq K, \tag{5}$$

$$\sum_{i'=1}^{n} \sum_{s=2}^{S} h_{i'} y_{ii's} \leq (H - h_i) \xi_i^P \qquad i = 1, \dots, n, \tag{6}$$

$$y_{ii'2} \leq c_{ii'} \xi_i^P \qquad i = 1, \dots, n, i' = 1, \dots, n, \tag{7}$$

$$y_{ii's} \leq \sum_{i''=1}^{n} c_{i''i'} y_{i,i'',s-1} \qquad i = 1, \dots, n, i' = 1, \dots, n, s = 3, \dots, S, \tag{8}$$

$$\sum_{i'=1}^{n} y_{ii's} \leq 1 \qquad i = 1, \dots, n, s = 2, \dots, S, \tag{9}$$

$$\sum_{i=1}^{2n} \sum_{(w',l') \in \mathcal{P}_i^{(w,l)}} x_{iw'l'} \leq 1 \qquad w = 0, \dots, W-1, l = 0, \dots, L-1, \tag{10}$$

$$\xi_i^P, \xi_i^S \in \{0, 1\} \qquad i = 1, \dots, n, \tag{11}$$

$$x_{iwl} \in \{0, 1\} \qquad i = 1, \dots, 2n, (w, l) \in \mathcal{P}_i, \tag{12}$$

$$y_{ii's} \in \{0, 1\} \qquad i = 1, \dots, n, i' = 1, \dots, n, s = 2, \dots, S. \tag{13}$$

Objective function (1) maximizes the value of the items inserted in the truck (i.e., the value of the bottom items plus the value of the stacked items). Constraints (2) and (3) link variables ξ_i^P and ξ_i^S with variables x_{iwl} and $y_{ii's}$. Constraints (4) ensure that an item can only be packed once, either at the bottom of the truck or in a stacked column. Constraint (5) makes sure that the load capacity of the truck is respected. Constraints (6) ensure that the height of every stacked column is at most H if it exists, and 0 otherwise. Constraints (7) ensure consistency between x_{ikl} and $y_{ii's}$ variables. They also ensure compatibility between every item packed in position 2 of a stacked column with the item placed right below it, in position 1. Constraints (8) ensure compatibility between every item packed in position 3 or above of a stacked column with the item placed right below it. Constraints (9) prevent two items to be packed in the same position of a given stacked column. Finally, constraints (10) ensure that items do not overlap by checking that no unit length square is occupied by more than one item. Model (1)–(13) is referred to as **COMPACT** hereafter.

We end this section by outlining that **COMPACT** requires $O(Sn^2)$ variables and $O(Sn^2 + WL)$ constraints, which quickly become large as W and L increase. We point out that there exists another stream of mathematical models in the C&P literature that uses a set of continuous decision variables to represent the packing coordinates of every item in each dimension instead of discretizing the packing region [24,25]. Even though the size of such models solely depends on the number of items n, they still require a set of $O(n^2)$ binary variables and a set of $O(n^2)$ big-M constraints to model the non-overlap requirements. Besides the fact that most techniques introduced hereafter cannot be directly applied to those models, preliminary experiments indicated that their empirical performance is significantly worse than the one displayed by **COMPACT**.

4. Preprocessing techniques

To improve the performance of an ILP model (i.e., the average computing time required by a state-of-the-art ILP solver to solve a given

Table 1
Mathematical notation.

Notation	Definition
W, L, H, K	Width, length, height, and capacity of the truck
S	Maximum number of layers per column
n	Number of items
$n+i$	Index of the rotated version of item i
w_i, l_i, h_i, k_i, p_i	Width, length, height, weight, and profit of item i
(w, l)	Point located in abscissa w and ordinate l
$\mathcal{P}_i$	Set containing all the points in which item i can be packed
$\mathcal{P}_i^{(w,l)}$	Set containing all the points in which item i can be packed such that i occupies (w, l)
$c_{ii'}$	Coefficient taking value 1 if item i' can be stacked on top of item i

instance of the model to optimality), it is common to first apply a set of preprocessing techniques on the instance one wants to solve [2–4]. Such techniques can be used, for example, to reduce the size of the model (i.e., the number of variables, the number of constraints, or the number of non-zero elements in the coefficient matrix), to strengthen the model continuous relaxation bound, or to remove some symmetry in the model solution space. In the following, we outline three sets of preprocessing techniques we used in our experiments: the first one is aimed at adjusting the item and truck dimensions, the second one is aimed at reducing the number of coordinates at which the items can be packed, and the third one is aimed at exploiting the fact that many items have the same *packing dimensions* (i.e., the same width and the same length).

4.1. Adjusting the item and truck dimensions

Following the idea introduced by Alvarez-Valdés et al. [26], one can reduce the truck height H by one unit as long as there are no combinations of item heights whose sum is equal to H. In practice, this can be done by solving the following knapsack problem:

$$\max \sum_{i=1}^{n} z_i h_i \ : \ \sum_{i=1}^{n} z_i h_i \leq H, z_i \in \{0,1\} \ (i = 1, \dots, n),$$

and by setting H to the optimal solution value of the problem. The same idea can be used to reduce the truck width W (or length L) after making the necessary adjustments to take the item rotation into account:

$$\max \sum_{i=1}^{2n} z_i w_i \ : \ \sum_{i=1}^{2n} z_i w_i \leq W, z_i + z_{n+i} \leq 1 \ (i = 1, \dots, n),$$
$$z_i \in \{0,1\} \ (i = 1, \dots, 2n).$$

As shown by Boschetti et al. [27], one can also increase the height of an item i' by one unit as long as there are no combinations of (other) item heights whose sum is equal to $H - h_{i'}$. In practice, this can also be done by solving a knapsack problem for every item i':

$$\max \sum_{i=1}^{n} z_i h_i \ : \ \sum_{i=1}^{n} z_i h_i \leq H - h_{i'}, z_{i'} = 0, z_i \in \{0,1\} \ (i = 1, \dots, n),$$

and by setting $h_{i'}$ to the height of the truck minus the optimal solution value of the problem. Once more, the same idea can be used to increase the width of an item i' (or its length) after making the necessary adjustments to take the item rotation into account:

$$\max \sum_{i=1}^{2n} z_i w_i \ : \ \sum_{i=1}^{2n} z_i w_i \leq W - w_{i'}, z_{i'} + z_{n+i'} = 0, z_i + z_{n+i} \leq 1 \ (i = 1, \dots, n),$$
$$z_i \in \{0,1\} \ (i = 1, \dots, 2n).$$

Solving these knapsack problems can be done with dynamic programming in negligible time for reasonable truck dimensions. However, such a sequential process does not necessarily lead to the best results. We propose to solve instead an ILP model aimed at simultaneously adjusting all the item and truck measures for a given dimension. For adjusting H and h_i $(i = 1, \dots, n)$, such an ILP model is as follows:

$$\min \quad n\bar{H} - \sum_{i=1}^{n} \bar{h}_i \tag{14}$$

$$\text{s.t.} \quad \sum_{i \in B} \bar{h}_i \leq \bar{H} \qquad B \in \mathcal{B}^H, \tag{15}$$

$$\sum_{i \in B} \bar{h}_i \geq \bar{H} + 1 \qquad B \in \not{\mathcal{B}}^H, \tag{16}$$

$$\bar{H} \in \mathbb{N}_0, \tag{17}$$

$$\bar{h}_i \in \mathbb{N}_0 \qquad i = 1, \dots, n, \tag{18}$$

where decision variable $\bar{H}$ represents the adjusted truck height, decision variables $\bar{h}_i$ represent the adjusted height of item i, $\mathcal{B}^H$ contains every *feasible set* of items (i.e., every set of items B such that $\sum_{i \in B} h_i \leq H$), and $\not{\mathcal{B}}^H$ contains every *infeasible set* of items (i.e., every set of items B such that $\sum_{i \in B} h_i \geq H+1$). While solving such an ILP model is usually impractical because the size of sets $\mathcal{B}^H$ and $\not{\mathcal{B}}^H$ grows exponentially with n, this strategy works very well when only few items can fit side by side in the truck (which is our case for the width and the height dimensions). In Example 4.1, we provide an instance for which the preprocessing obtained by solving model (14)–(18) is better than the preprocessing obtained by solving a sequence of knapsack problems.

Example 4.1. Let us consider the following instance with $n = 9$ items with $h_1 = h_2 = 4, h_3 = h_4 = h_5 = h_6 = 5, h_7 = h_8 = h_9 = 6$, and $H = 20$. One can observe that no adjustment occurs after applying the standard preprocessing as $h_1 + h_3 + h_4 + h_7 = 4 + 5 + 5 + 6 = 20 = H$. However, using model (14)–(18) on the instance results in adjusted item heights $\bar{h}_1 = \bar{h}_2 = 2, \bar{h}_3 = \bar{h}_4 = \bar{h}_5 = \bar{h}_6 = 3, \bar{h}_7 = \bar{h}_8 = \bar{h}_9 = 4$, and adjusted $\bar{H} = 12$. We point out that every feasible set of items with the original heights remains feasible with the adjusted heights, and that every infeasible set of items with the original heights remains infeasible with the adjusted heights. □

While we are not aware of such a preprocessing being used in the literature to solve packing problems, we point out that the idea relies on the work of Kartak et al. [28] who showed that one *"can decompose the infinite set of one-dimensional cutting stock problem instances, with a fixed number of demanded pieces, into a finite number of equivalence classes"*. The equivalence class of a given instance is determined after building sets $\mathcal{B}^H$ and $\not{\mathcal{B}}^H$ whereas model (14)–(18) computes the instance with the lowest $n\bar{H} - \sum_{i=1}^{n} \bar{h}_i$ value within that equivalence class.

4.2. Reducing the number of packing coordinates

Following the seminal work of Christofides and Whitlock [29], we use the concept of *normal patterns* to generate the set of packing coordinates $\mathcal{P}_i$ instead of including every point (w, l) such that $w = 0, \dots, W - w_i$ and $l = 0, \dots, L - l_i$. The authors outlined that *"if a rectangle [...] is to be cut [...] at some position x, then [...] there must be some combination of the lengths l_i of the available pieces [...] whose sum must be exactly x"*. As a result, the set of packing coordinates $\mathcal{P}_{i'}$ can be

redefined for every item i' so as to contain each point (w,l) satisfying the two non-related conditions:

$$w = \sum_{i=1}^{2n} z_i w_i \,:\, w \le W - w_{i'}, z_{i'} + z_{n+i'} = 0, z_i + z_{n+i} \le 1 \ (i = 1,\dots,n),$$
$$z_i \in \{0,1\} \ (i = 1,\dots,2n),$$
$$l = \sum_{i=1}^{2n} z_i l_i \,:\, l \le L - l_{i'}, z_{i'} + z_{n+i'} = 0, z_i + z_{n+i} \le 1 \ (i = 1,\dots,n),$$
$$z_i \in \{0,1\} \ (i = 1,\dots,2n).$$

Normal patterns can be computed using dynamic programming.

Two immediate supplementary reduction procedures consist in: (i) removing from $\mathcal{P}_i$ all packing coordinates (w,l) such that $0 < W - (w + w_i) < w_{\min}$ (where $w_{\min}$ is the width of the smallest item) and replacing them by $(W - w_i, l)$ and (ii) removing from $\mathcal{P}_i$ all packing coordinates (w,l) such that $0 < L - (l + l_i) < l_{\min}$ (where $l_{\min}$ is the length of the smallest item) and replacing them by $(w, L - l_i)$. Indeed, if packing an item i at a certain coordinate (w,l) would not leave space for any other item in a given dimension (i.e., if either $W - (w + w_i) < w_{\min}$ or $L - (l + l_i) < l_{\min}$), then item i can be pushed to the edge of the truck on that dimension (i.e., item i should be packed in $(W - w_i, l)$ or in $(w, L - l_i)$). Note that recent works showed that smaller subsets could theoretically be obtained by using the so-called "meet-in-the-middle" patterns [30], but we did not observe any empirical improvement in our computational experiments by doing so.

4.3. Exploiting the item packing dimension multiplicity

Even after implementing the aforementioned preprocessing and variable reduction techniques, model **COMPACT** still struggles to solve some trivial realistic instances because of its large size. While it is known that solving such a model with an ILP solver does not generally produce the best empirical results, one can use the fact that the item widths and lengths are very homogeneous to significantly reduce the model size and enhance its performance. From a packing perspective, this can be seen as merging several items in a bin packing problem to solve instead a cutting stock problem. The packing component of the model (which only involves the width and the length of the items) will use variables related to the *item type* whereas the rest of the model (which involves more heterogeneous features such as weight, profit, and stackability) will use variables related to the *item index*.

We create one item type $j \in \mathcal{T}$ per distinct couple $(\max\{w_i, l_i\}, \min\{w_i, l_i\})$ $(i = 1,\dots,n)$. We call m the number of item types (i.e., $m = |\mathcal{T}|$) and we use set $\mathcal{T}(j)$ to gather the index of the items with type j $(j = 1,\dots,m)$. Each of the m item types is duplicated once to represent its rotated version. We redefine sets $\mathcal{P}_i$, $\mathcal{P}_i^{(w,l)}$, sizes w_i and l_i, and binary decision variables x_{iwl} (which were originally created for every item index) such that they are now associated with the item types. Those now appear as $\mathcal{P}_j, \mathcal{P}_j^{(w,l)}, w_j, l_j$, and x_{jwl} in the models. We also introduce new integer decision variables $\xi_j^{P\text{-}T}$ $(j = 1 \dots m)$ that indicate the number of times item type j is a bottom item and link them with variables ξ_i^P $(i = 1 \dots n)$ as follows:

$$\sum_{i \in \mathcal{T}(j)} \xi_i^P = \xi_j^{P\text{-}T} \quad j = 1,\dots,m.$$

We update constraints (2) and (10) to now use the item type instead of the item index:

$$\xi_j^{P\text{-}T} = \sum_{(w,l)\in\mathcal{P}_j} x_{jwl} + \sum_{(w,l)\in\mathcal{P}_{m+j}} x_{m+j,w,l} \qquad j = 1,\dots,m,$$
$$\sum_{i=1}^{2m} \sum_{(w',l')\in\mathcal{P}_j^{(w,l)}} x_{jw'l'} \le 1 \qquad w = 0,\dots,W-1, l = 0,\dots,L-1.$$

We finish by updating the domain constraints:

$$\xi_j^{P\text{-}T} \in \mathbb{N}_0 \qquad j = 1,\dots,m,$$
$$x_{jwl} \in \{0,1\} \quad j = 1,\dots,2m, (w,l) \in \mathcal{P}_j.$$

Note that the objective function and constraints (3)–(9), (11), and (13) remain the same. This updated model is referred to as **COMPACT-CSP** in the next sections. As it will be shown in our computational experiments, model **COMPACT-CSP** performs better than model **COMPACT** because of its reduced size. Indeed, as $m \ll n$ in our realistic instances, the number of variables x_{jwl} is greatly reduced and the updated constraints (10) involve significantly fewer non-zero coefficients.

5. Sequential approaches

Our first solution method, which is inspired by the one currently used by the company, consists in solving the problem in two phases: first, one groups the items together in order to form good quality columns (called "column creation problem" afterwards) and second, one selects the best set of columns that can fit into the truck while also determining the truck layout (called "column selection and positioning problem" afterwards). Whereas the company solves each phase heuristically, we follow a different strategy in which those are solved to optimality. In the following, we provide a mathematical model for each phase, we show that both the column creation problem and the column selection and positioning problem are $\mathcal{NP}$-hard, and we provide a discussion as to why such a sequential approach may sometimes lead to poor quality solutions overall.

5.1. Column creation problem (CCP)

In the CCP, one groups the items together in order to form the columns that will be used in the second phase. While several definitions could be relevant as to what constitutes the best set of columns (e.g., the set with minimum cardinality or the set with the maximum number of "big" columns), we looked for the set of columns with minimum total area (the area of a column is equal to the area of the lowest item in the column). This decision is motivated by the data sets provided by the company: as all items can fit into the truck in most instances (i.e., it is very likely that all columns will be selected in the second phase of the problem), we want to find a column set that is as easy to position as possible. Model **COMPACT-CSP** can easily be adapted to solve the CCP as follows:

$$\min \quad \sum_{j=1}^{m} w_j l_j \xi_j^{P\text{-}T} \tag{19}$$
$$\text{s.t.} \quad \sum_{i\in\mathcal{T}(j)} \xi_i^P = \xi_j^{P\text{-}T} \qquad j = 1,\dots,m, \tag{20}$$
$$\xi_i^P + \xi_i^S = 1, \tag{21}$$
$$(3), (6), (7), (8), (9), (11), (13),$$
$$\xi_j^{P\text{-}T} \in \mathbb{N}_0 \qquad j = 1,\dots,m, \tag{22}$$

where availability constraints (4) are replaced by demand constraints (21), and weight constraint (5) and non-overlapping constraints (10) are removed. This model is called **SA-CCP** thereafter.

We can easily show that CCP-D, the decision version of the CCP where one asks whether there is a solution with value z or less, for a given $z \in \mathbb{Z}_0^+$, is $\mathcal{NP}$-complete as there is a polynomial reduction from the well-known ($\mathcal{NP}$-complete) Partition problem to CCP-D. Let us consider an instance I' of Partition in which we are given a set of n' positive integers $\mathcal{N}' = \{w'_1,\dots,w'_n\}$ and where the objective is to determine whether $\mathcal{N}'$ can be partitioned into two subsets $\mathcal{N}'_1$ and $\mathcal{N}'_2$ such that $\mathcal{N}'_1 \cup \mathcal{N}'_2 = \mathcal{N}'$, $\mathcal{N}'_1 \cap \mathcal{N}'_2 = \emptyset$, and $\sum_{w'\in\mathcal{N}'_1} w' = \sum_{w'\in\mathcal{N}'_2} w'$. We show how to create in polynomial time an instance I of CCP-D such that, if I has answer Yes, then I' has answer Yes, while if I has answer No, then I' has answer No. To do so, for each integer w'_i in

$\mathcal{N}'$, we generate a flat and stackable item with height $2w'_i$, length 1, width 1, and weight 1 (resulting in a complete compatibility graph $\mathcal{G}$). We also set S to n', H to $\sum_{w' \in \mathcal{N}'} w'$, and z to 2. If the answer to the CCP-D instance is YES, then there exists a feasible solution using exactly two columns with height $\sum_{w' \in \mathcal{N}'} w'$ where the items stacked in each column correspond to the integers in each subset in the PARTITION solution.

A link with the kidney exchange problem

It is interesting to notice that if one relaxes constraints (6) (i.e., the constraints limiting the column heights), then the CCP becomes (a special version of) the *Kidney Exchange Problem* (KEP), an extensively studied combinatorial optimization problem for which effective mathematical models have been proposed in the literature [31–33]. In the KEP, one looks for a vertex-disjoint set of cycles and chains in a graph given a limit on the maximum cycle size and chain length. In the compatibility graph of the KEP, a node represents a recipient/donor pair while an arc between two nodes represents a compatibility relationship between the donor of the first pair and the recipient of the second pair. An item i in the CCP would therefore correspond to a recipient/donor pair i in the KEP, an arc (i, i') in the CCP compatibility graph $\mathcal{G}$ indicating that item i' can be stacked on top of item i could be seen as a compatibility relationship between the donor of pair i and the recipient of pair i', a stacked column in a CCP solution would be a chain in a KEP solution (while cycles would not exist), and the limit S on the maximum number of items per stacked column in the CCP would correspond to the maximum chain length in the KEP. As a result, even though the KEP is $\mathcal{NP}$-hard [34], one could still adapt the most effective KEP model to solve the CCP: the *Position-Indexed Chain-Edge Formulation*, also known as PICEF [32].

In our PICEF adaptation (called **PICEF-CCP** afterwards), binary decision variables $y_{ii's}$ takes value 1 if item i' ($i' = 1, \dots, n$) is packed on top of item i ($i = 1, \dots, n$) in position s ($s = 2, \dots, S$), and 0 otherwise. In other words, one only needs to keep track of the predecessor of an item (i.e., the item below it) instead of the index of the column in which the item is packed as done in model **SA-CCP**. **PICEF-CCP** is as follows:

$$\min \quad \sum_{j=1}^{m} w_j l_j \xi_j^{P\text{-}T} \tag{23}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{T}(j)} \xi_i^P = \xi_j^{P\text{-}T} \qquad j = 1, \dots, m, \tag{24}$$

$$\xi_i^P + \xi_i^S = 1, \tag{25}$$

$$\xi_{i'}^S = \sum_{(i,i') \in \mathcal{A}} \sum_{s=2}^{S} y_{ii's} \qquad i' = 1, \dots, n, \tag{26}$$

$$\sum_{(i,i') \in \mathcal{A}} y_{ii'2} \le \xi_i^P \qquad i = 1, \dots, n, \tag{27}$$

$$\sum_{(i,i') \in \mathcal{A}} y_{ii's} \le \sum_{(i',i) \in \mathcal{A}} y_{i',i,s-1} \qquad i = 1, \dots, n, s = 3, \dots, S, \tag{28}$$

$$\xi_i^P, \xi_i^S \in \{0, 1\} \qquad i = 1, \dots, n, \tag{29}$$

$$\xi_j^{P\text{-}T} \in \mathbb{N}_0 \qquad j = 1, \dots, m, \tag{30}$$

$$y_{ii's} \in \{0, 1\} \qquad (i, i') \in \mathcal{A}, s = 2, \dots, S. \tag{31}$$

We point out that model **SA-CCP** has $O(Sn^2)$ constraints and $O(Sn^2)$ variables whereas **PICEF-CCP** has $O(Sn)$ constraints and $O(Sn^2)$ variables. Note that one could still find an optimal solution for the CCP by using **PICEF-CCP** if the model is solved within a constraint generation framework such that no-good cuts

$$\xi_i^P + \sum_{(i,i',s) \in \ell} y_{ii's} \le |\ell|,$$

are added on the fly when a column starting from item i and composed of arcs $\ell = \{(i, i', 2), (i', i'', 3), \dots\}$ that does not respect the height constraint is found in a solution. One could therefore favor **PICEF-CCP** over **SA-CCP** in case the constraints limiting the column heights are rarely binding. Experiments outlining the computational behavior of such an approach are provided in Section 7.

5.2. Column selection and positioning problem (CSPP)

In the CSPP, one selects the best set of columns that can fit into the truck while also determining the truck layout. For each column i ($i = 1, \dots, n'$) returned by the CCP (i.e., for each item i such that $\xi_i^P = 1$), we determine its width w_i and its length l_i (i.e., the width and the length of the first item in the column) and its profit p_i (i.e., the sum of the price of every item composing the column). The CSPP therefore reduces to the very well-known *two-dimensional knapsack problem* (2D-KP) which has been extensively studied in the literature (see [35] for a recent survey on the topic). By re-using the principles and notations defined for **COMPACT-CSP** where n' now indicates the number of columns, and m' the number of column types, the CSPP can be modeled as follows:

$$\max \quad \sum_{i=1}^{n'} p_i \xi_i^P \tag{32}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{T}(j)} \xi_i^P = \xi_j^{P\text{-}T} \qquad j = 1, \dots, m', \tag{33}$$

$$\xi_j^{P\text{-}T} = \sum_{(w,l) \in \mathcal{P}_j} x_{jwl} + \sum_{(w,l) \in \mathcal{P}_{m'+j}} x_{m'+j,w,l} \qquad j = 1, \dots, m', \tag{34}$$

$$\sum_{j=1}^{2m'} \sum_{(w',l') \in \mathcal{P}_j^{(w,l)}} x_{jw'l'} \le 1 \qquad w = 0, \dots, W-1, l = 0, \dots, L-1, \tag{35}$$

$$\xi_i^P \in \{0, 1\} \qquad i = 1, \dots, n', \tag{36}$$

$$\xi_j^{P\text{-}T} \in \mathbb{N}_0 \qquad j = 1, \dots, m', \tag{37}$$

$$x_{jwl} \in \{0, 1\} \qquad j = 1, \dots, 2m', (w, l) \in \mathcal{P}_j. \tag{38}$$

We point out that since the 2D-KP is $\mathcal{NP}$-hard [35], then it means that the CSPP is also $\mathcal{NP}$-hard.

5.3. Relevant considerations

While it is possible that the solution obtained after solving the CCP and the CSPP sequentially is also optimal for our truck loading problem overall, we provide in Example 5.1 an instance for which this is not the case.

Example 5.1. Let us consider the following instance as described in Fig. 1 with $n = 5$ items A, B, C, D, and E:

An optimal CCP solution with area $12\times12+9\times9 = 225$ packs the five items in two columns: a first column composed of items A, B, C, and E, and a second column solely composed of item D. However, since the two columns cannot be packed simultaneously in the truck, the resulting CSPP solution consists in only packing the first column for a value of $144 + 96 + 96 + 64 = 400$, which is not optimal for the overall problem. An optimal solution for the overall problem has value 481 and can be obtained by using, for example, a column composed of items A and D and a column composed of items B, C, and E. Note that a CSPP solution with value 481 can only be reached by using a set of columns with total area greater than or equal to 240 (i.e., which is suboptimal for the CCP). □

We finish this section by pointing out that even though alternative strategies for the CCP can be considered (e.g., finding the column set with minimum cardinality or the column set with the maximum number of "big" columns), one can always find an ad hoc instance such that the set of columns selected in the first phase cannot lead to an optimal solution for the overall problem. In other words, even if sequential approaches may work very well in practice (as demonstrated in Section 7), there is no guarantee that such methods terminate with an optimal solution. To alleviate this issue, we introduce three decomposition-based exact algorithms in the next section.

	width	length	height	weight
truck	20	20	5	5
A	12	12	1	1
B	12	8	1	1
C	12	8	1	1
D	9	9	1	1
E	8	8	1	1

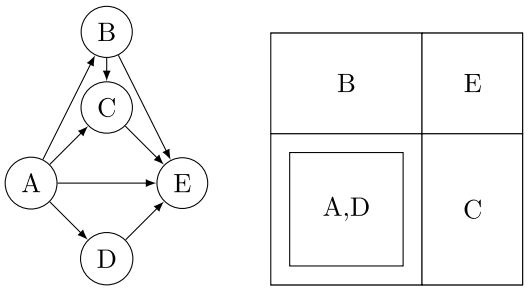


Fig. 1. An instance and its optimal solution.

6. Integrated approaches

We identified three distinct components in our problem: (i) selecting a subset of items and grouping that subset into columns, (ii) finding a suitable abscissa for every column, and (iii) finding a suitable ordinate for every column. Each of the proposed decomposition methods handles these three components in their own way, as summarized in Fig. 2:

- The first approach (on the left of the figure) is a Benders' decomposition in which component (i) is dealt with in the master problem while components (ii) and (iii) are determined simultaneously in the subproblem;
- The second approach (in the middle of the figure) is also a Benders' decomposition in which components (i) and (ii) are dealt with simultaneously in the master problem while component (iii) is determined in the subproblem;
- The last approach (on the right of the figure) relies on two embedded Benders' decompositions in which component (i) is dealt with in the master problem of the first decomposition, component (ii) is dealt with in the master problem of the second decomposition, and component (iii) is determined in the subproblem of the second decomposition. Note that the subproblem of the first decomposition is formed by the second decomposition itself.

In the remainder of this section, we provide further details about each decomposition.

6.1. Decomposition 1

In our first Benders' decomposition, we used a strategy that is inspired by the approach proposed in [22] for the *three-dimensional knapsack problem*, where the authors determined in the master problem a set of items with maximum value and checked in the subproblem whether or not the selected items could fit into the container. We modified that approach so that it is able to take into account our practical constraints by letting the master problem determine the stacked columns instead of the selected items and by considering only two dimensions in the subproblem.

Master problem (MP1). **MP1** determines a set of feasible stacked columns with maximum value such that the sum of their areas (the area of a column is equal to the area of the lowest item in the column) is smaller than or equal to the area of the truck. The model is as follows:

$$\max \quad \sum_{i=1}^{n} p_i \xi_i^P + \sum_{i=1}^{n} p_i \xi_i^S \tag{39}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{T}(j)} \xi_i^P = \xi_j^{P\text{-}T} \qquad j = 1, \dots, m, \tag{40}$$

(3) – (9), (11), (13),

$$\sum_{j=1}^{m} w_j l_j \xi_j^{P\text{-}T} \leq WL, \tag{41}$$

$$\xi_j^{P\text{-}T} \in \mathbb{N}_0 \qquad j = 1, \dots, m, \tag{42}$$

and is a straightforward adaptation of model **COMPACT-CSP** where the packing variables x_{jwl} $(j = 1, \dots, 2m, (w,l) \in \mathcal{P}_j)$ are not considered and where non-overlapping constraints (8) are relaxed and replaced by constraint (41).

Subproblem (SP1). **SP1** determines whether it is possible to find feasible values for packing variables x_{jwl} $(j = 1, \dots, 2m, (w,l) \in \mathcal{P}_j)$ satisfying demand and non-overlapping constraints given a solution $\{\bar{\xi}_i^P, \bar{\xi}_j^{P\text{-}T}, \bar{\xi}_i^S, \bar{y}_{ii's}\}$ returned by **MP1**. As only the widths and the lengths of the bottom items (i.e., the $\bar{\xi}_j^{P\text{-}T}$) determine whether or not such a solution exists, **SP1** reduces to a simple *orthogonal packing problem* (see [2,36] for two recent research papers related to the topic). **SP1** can therefore be modeled as follows:

$$\max \quad 0 \tag{43}$$

$$\text{s.t.} \quad \bar{\xi}_j^{P\text{-}T} = \sum_{(w,l) \in \mathcal{P}_j} x_{jwl} + \sum_{(w,l) \in \mathcal{P}_{m+j}} x_{m+j,w,l} \qquad j = 1, \dots, m, \tag{44}$$

$$\sum_{j=1}^{2m} \sum_{(w',l') \in \mathcal{P}_j^{(w,l)}} x_{jw'l'} \leq 1 \qquad w = 0, \dots, W-1, l = 0, \dots, L-1, \tag{45}$$

$$x_{jwl} \in \{0,1\} \qquad j = 1, \dots, 2m, (w,l) \in \mathcal{P}_j. \tag{46}$$

If **SP1** turns out to be feasible, then an optimal solution for the problem has been found. Otherwise, one should solve again **MP1** with a supplementary *no-good cut*.

No-good cut (NGC1). Using ξ_i^P variables in **NGC1** would not be expedient since any solution with different item indices, but with the exact same item types could be returned by **MP1**. This is not desirable since the feasibility of **SP1** solely depends on the types of the bottom items, not their indices. Therefore, $\xi_j^{P\text{-}T}$ variables should ideally be used in **NGC1**, but this is not possible with the current version of **MP1** as $\xi_j^{P\text{-}T}$ are integer variables whereas no-good cuts require binary variables. As a consequence, we used the concept of a *binary expansion* in **MP1** and replaced every integer variable $\xi_j^{P\text{-}T}$ $(j = 1, \dots, m)$ by a set of binary variables $\xi_{jg}^{P\text{-}T\text{-}B}$ $(j = 1, \dots, m, g = 0, \dots, \lfloor \log_2(d_j) \rfloor)$ such that:

$$\xi_j^{P\text{-}T} = \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} 2^g \xi_{jg}^{P\text{-}T\text{-}B} \qquad j = 1, \dots, m,$$

where d_j indicates the total number of items with type j (i.e., $d_j = |\mathcal{T}_j|$). **NGC1** can now be defined as follows:

$$\sum_{j=1}^{m} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} \bar{\xi}_{jg}^{P\text{-}T\text{-}B} \xi_{jg}^{P\text{-}T\text{-}B} \leq \left(\sum_{j=1}^{m} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} \bar{\xi}_{jg}^{P\text{-}T\text{-}B} \right) - 1, \tag{47}$$

which prevents any infeasible **MP1** solution from being generated again. We point out that, in general, no-good cuts can be strengthened to obtain the so-called "combinatorial Benders' cuts" [2]. To do so, one should look for a minimum cardinality subset of items among those selected in an infeasible **MP1** solution for which **SP1** remains infeasible. Whereas using combinatorial Benders' cuts may drastically reduce the number of constraints that needs to be added to the master problem in order to solve an instance to optimality [2,4,37], the fact that finding a minimum infeasible subset requires solving multiple subproblems

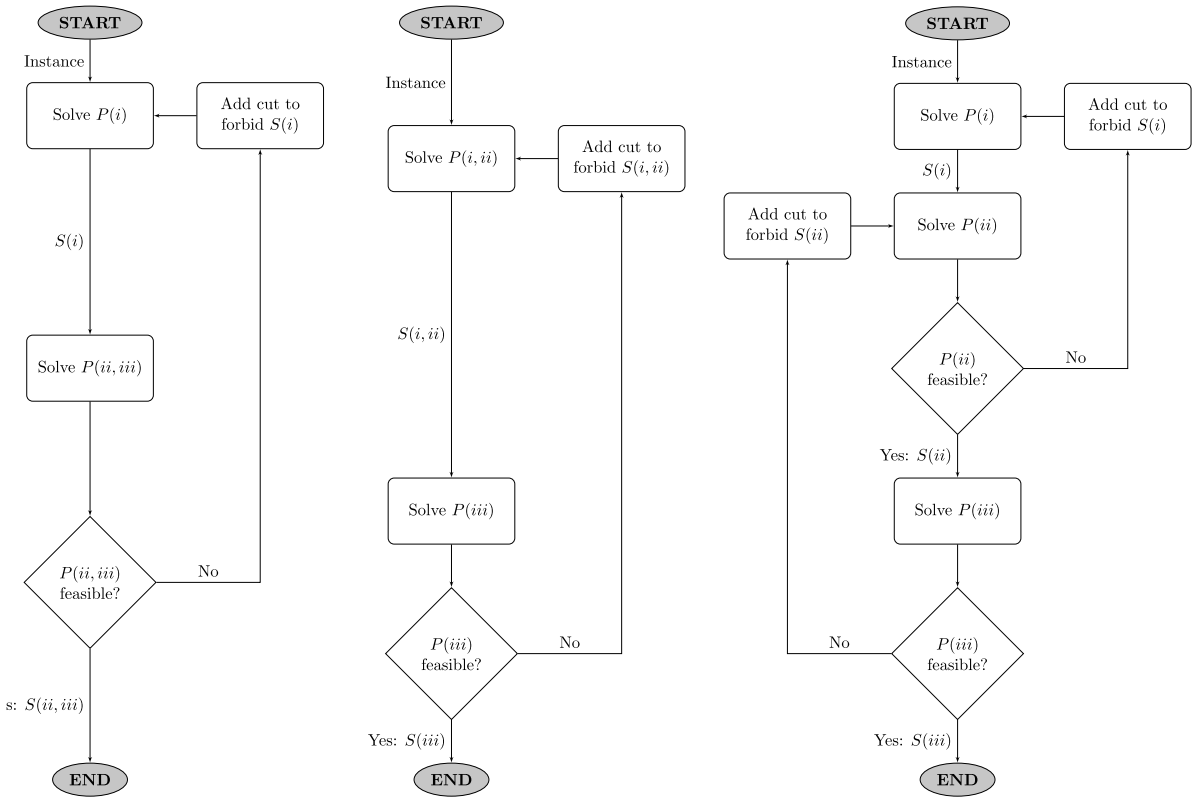


Fig. 2. Flowchart of the three Benders' decomposition approaches.

does not always make such a strategy expedient. Besides, it is not immediately clear how such cuts can efficiently be implemented in the context of a binary expansion.

6.2. Decomposition 2

In our second Benders' decomposition, we used a strategy that is inspired by the approach proposed in [2,4] for the *two-dimensional strip packing problem*, where the authors determined in the master problem the minimum strip height and a suitable x-coordinate for every item and checked in the subproblem whether or not the packing could be completed by finding a suitable y-coordinate for every item. We modified that approach so that it is able to take into account our practical constraints by letting the master problem determine the selected subset of items, the stacked columns, and the orientation and abscissa of every bottom item (or column).

Master problem (MP2). **MP2** determines both a set of feasible stacked columns with maximum value and their abscissa such that, for every abscissa w, the sum of the lengths of the bottom items occupying w does not exceed the truck length L. We say that a bottom item with type j ($j = 1\dots,2m$) occupies w if a portion of the item overlaps with the unit-width slice starting in w and ending in $w+1$, or in other words, if the item is packed in abscissa w' such that $0 \le w - w_j + 1 \le w' \le w$. Reusing the notation introduced in the previous models, $\mathcal{P}_j$ contains all the abscissae w in which a bottom item with type j ($j = 1,\dots,2m$) can be packed, $\mathcal{P}_j^w$ contains all the abscissae w' for which a bottom item with type j ($j = 1,\dots,2m$) would occupy w if the item was packed in w' ($w' \in \mathcal{P}_j$), and we replace two-dimensional (binary) packing variables x_{jwl} by their one-dimensional (integer) counterpart $x_{jw}(j = 1,\dots,2m, w \in \mathcal{P}_j)$, to which binary expansion is applied, resulting in (binary) variables $x^B_{jwg}(j = 1,\dots,2m, w \in \mathcal{P}_j, g = 0,\dots,\lfloor\log_2(d_j)\rfloor)$. **MP2** is defined as follows:

$$\max \quad \sum_{i=1}^{n} p_i \xi_i^P + \sum_{i=1}^{n} p_i \xi_i^S \tag{48}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{T}(j)} \xi_i^P = \xi_j^{P\text{-}T} \qquad j = 1,\dots,m, \tag{49}$$

$$\xi_j^{P\text{-}T} = \sum_{w=0}^{W-w_j} \sum_{g=0}^{\lfloor\log_2(d_j)\rfloor} 2^g x^B_{jwg} + \sum_{w=0}^{W-w_{j+m}} \sum_{g=0}^{\lfloor\log_2(d_j)\rfloor} 2^g x^B_{j+m,w,g} \qquad j = 1,\dots,m, \tag{50}$$

(3) – (9), (11), (13),

$$\sum_{j=1}^{2m} \sum_{w' \in \mathcal{P}_j^w} \sum_{g=0}^{\lfloor\log_2(d_j)\rfloor} 2^g l_j x^B_{jw'g} \le L \qquad w = 0,\dots,W-1, \tag{51}$$

$$\xi_j^{P\text{-}T} \in \mathbb{N}_0 \qquad j = 1,\dots,m, \tag{52}$$

$$x^B_{jwg} \in \{0,1\} \qquad j = 1,\dots,2m, w \in \mathcal{P}_j, g = 0,\dots,\lfloor\log_2(d_j)\rfloor, \tag{53}$$

and is a straightforward adaptation of model **COMPACT-CSP** where linking constraints (2) are replaced by (50) to take into account the updated x^B_{jwg} variables and where non-overlapping constraints (10) are relaxed and replaced by capacity constraints (51).

Subproblem (SP2). **SP2** determines whether it is possible to find a feasible y-coordinate for every bottom item satisfying demand and non-overlapping constraints given an **MP2** solution. Note that the feasibility of **SP2** only depends on the x^B_{jwg} variables. **SP2** is an $\mathcal{NP}$-complete scheduling problem with non-overlapping constraints for which exact approaches were developed in the literature [2,4]. We solved **SP2** with the following Constraint Programming (CP) approach, which was originally proposed by [4], as it performs very well in practice while being easy to reimplement. Given an MP2 **solution** $\{\bar{\xi}_i^P, \bar{\xi}_j^{P\text{-}T}, \bar{\xi}_i^S, \bar{y}_{ii's}, \bar{x}^B_{jwg}\}$ returned by **MP2**, we create, for each item type j and for each abscissa w, exactly $\sum_{g=0}^{\lfloor\log_2(d_j)\rfloor} 2^g \bar{x}^B_{jwg}$ tasks with processing time l_j occupying abscissae $w, w+1,\dots,w+w_j-1$. All the tasks are gathered in set $\mathcal{I}$ and

all the tasks occupying abscissa w are gathered in set $\mathcal{I}_w$. We associate an interval variable ivl_i to each task i ($i \in \mathcal{I}$) and solve the following CP model:

$$\max \quad 0 \tag{54}$$

$$\text{s.t.} \quad \texttt{endof}(ivl_i) \leq L \qquad i \in \mathcal{I}, \tag{55}$$

$$\texttt{nonoverlap}\left\{ivl_i, i \in \mathcal{I}_w\right\} \qquad w = 0, \dots, W-1, \tag{56}$$

where the starting time of a task i corresponds to the y-coordinate of the associated item. Constraints (55) state that every task needs to be completed by time L and constraints (56) make sure that the tasks occupying the same abscissa do not overlap in time (or in other words, that the items overlapping on the x-axis do no overlap on the y-axis). If **SP2** turns out to be feasible, then an optimal solution for the problem has been found. Otherwise, one should solve again **MP2** with a supplementary no-good cut.

No-good cut (NGC2). Following the idea developed for **NGC1**, **NGC2** can be defined as:

$$\sum_{j=1}^{2m} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} \bar{x}^B_{jwg} x^B_{jwg} \leq \left(\sum_{j=1}^{2m} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} \bar{x}^B_{jwg} \right) - 1. \tag{57}$$

6.3. Decomposition 3

In our last Benders' decomposition, we use a strategy that is inspired by the approach proposed in [3] for the *two-dimensional bin packing problem*, where the authors determined in the master problem the minimum number of bins required to pack all the items together with the item subset assigned to each bin and checked in the subproblem whether or not each of the resulting item allocations was feasible. The subproblem itself was solved through the Benders' decomposition proposed in [2]. We modified that approach so that it is able to take into account our practical constraints by letting the master problem determine the stacked columns instead of the item allocation.

First level master problem (MP3-1). **MP3-1**, which is identical to **MP1**, determines a set of feasible stacked columns with maximum value such that the sum of their areas (the area of column is equal to the area of the lowest item in the column) is smaller than or equal to the area of the truck.

Second level master problem (MP3-2). **MP3-2**, which is a simplified version of **MP2**, determines whether it is possible to find a feasible x-coordinate for every bottom item satisfying demand and capacity constraints given an **MP3-1** solution. **MP3-2** can be defined as follows:

$$\max \quad 0 \tag{58}$$

$$\text{s.t.} \quad \bar{\xi}^{P\text{-}T}_j = \sum_{w=0}^{W-w_j} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} 2^g x^B_{jwg} + \sum_{w=0}^{W-w_{j+m}} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} 2^g x^B_{j+m,w,g} \qquad j = 1, \dots, m, \tag{59}$$

$$\sum_{j=1}^{2m} \sum_{w' \in \mathcal{P}^w_j} \sum_{g=0}^{\lfloor \log_2(d_j) \rfloor} 2^g l_j x^B_{jw'g} \leq L \qquad w = 0, \dots, W-1, \tag{60}$$

$$x^B_{jwg} \in \{0,1\} \qquad j = 1, \dots, 2m, w \in \mathcal{P}_j, g = 0, \dots, \lfloor \log_2(d_j) \rfloor. \tag{61}$$

Second level subproblem (SP3-2). **SP3-2**, which is identical to **SP2**, determines whether it is possible to find a feasible y-coordinate for every bottom item satisfying demand and non-overlapping constraints given an **MP3-2** solution.

First and second level no-good cuts (NGC3-1 and NGC3-2). First level no-good cut **NGC3-1** is identical to **NGC1** whereas second level no-good cut **NGC3-2** is identical to **NGC2**.

7. Computational experiments

We empirically investigated the performance of each of the proposed approaches, namely:

- **COMPACT-CSP**, model (1)–(13) introduced in Section 3 with all the preprocessing techniques introduced in Sections Section 4.1, 4.2, and 4.3. For the sake of completeness, we also tested the model without the preprocessing related to item multiplicity (**COMPACT**) and without any preprocessing at all (**COMPACT-NOP**);
- **SEQ**, the sequential approach introduced in Section 5 composed of **SA-CCP** for the first phase and model (32)–(38) solved through a decomposition algorithm similar to the one proposed in [2] for the second phase. We also tested **SEQ-KEP**, a sequential approach in which **SA-CCP** is replaced by **PICEF-CCP**;
- **DEC1-CB**, the first decomposition approach introduced in Section 6.1 in which **SP1** is embedded in a callback function invoked when an integer solution for **MP1** is found. We also tested **DEC1**, the same decomposition without callbacks. To improve the performance of **DEC1**, we refined the objective function of **MP1** to incorporate a hierarchical component such that, if more than one optimal solution exists (i.e., if there are at least two sets of feasible stacked columns with maximum value whose sum of areas is smaller than or equal to the area of the truck), then the solution with minimum sum of areas is favored;
- **DEC2-CB**, the second decomposition approach introduced in Section 6.2 in which **SP2** is embedded in a callback function invoked when an integer solution for **MP2** is found. We also tested **DEC2**, the same decomposition without callbacks;
- **DEC3-CB**, the third decomposition approach introduced in Section 6.3 in which **MP3-2** is embedded in a callback function invoked when an integer solution for **MP3-1** is found and in which **SP3-2** is embedded in a callback function invoked when an integer solution for **MP3-2** is found. We also tested **DEC3**, the same decomposition without callbacks;

Our algorithms were coded in `C++`, the ILP models were solved with `Gurobi 10.0.1` whereas the CP components were solved with `IBM ILOG CPLEX CP Optimizer 22.11` (we remind the reader that Gurobi does not offer a CP solver at the moment). The tests were executed on a single thread of a virtual machine `AMD EPYC-Rome Processor` with 2.00 GHz and 64 GB of RAM memory, running under `Ubuntu 20`. For every run, a time limit of 3600 s was imposed. We first study the computational behavior of the proposed methods on real instances provided by the company, we then use randomly generated instances to assess how this behavior evolves when key instance features such as the number of item types or the truck layout change, and we conclude by demonstrating how supplementary features such as customer multiplicity and generalized fragility constraints can also be taken into account in our models. All our computer codes and instances can be downloaded from the online repository https://github.com/mdelorme2/Single_Container_Loading_Problem_With_Stacking_Constraints.

7.1. Real instances

The company provided 10 real instances with the following features: the number of items n was between 28 and 49, the number of item types m was between 3 and 8, the truck dimensions (in cm) were $W \times L \times H = 248 \times 1362 \times 270$, the smallest item dimensions (in cm) were $(w_{\min}, l_{\min}, h_{\min}) = (40, 60, 30)$ whereas the largest were $(w_{\max}, l_{\max}, h_{\max}) = (125, 240, 260)$, the smallest item packing area (in cm^2) was $w_i \times l_i = 2400$ whereas the biggest was 28 800, the item weight was never binding (i.e., $\sum_{i=1}^{n} k_i \leq K$), 68% of the items were stackable, 71% of the stackable items had a flat surface, and the maximum number of items allowed per column S was 3.

We report in Tables 2 and 3 the detailed results of our 11 approaches for each of the 10 instances. The first two columns identify the index and the number of items for every instance. The following columns indicate, for each method, whether a proven optimal solution was found ("# opt"), the CPU time in seconds ("T (s)") required by the

Table 2
Performance of the compact model and the sequential approaches on the real instances.

Idx	n	COMPACT-NOP		COMPACT		COMPACT-CSP		SEQ		SEQ-KEP		
		# opt	T (s)	# opt	T (s)	# opt	T (s)	# opt	T (s)	# opt	T (s)	# C
1	38	0	M.L.	1	59	1	5	1	3	1	5	801
2	28	0	M.L.	1	4	1	1	1	1	1	1	6
3	35	0	M.L.	1	77	1	5	1	3	1	5	234
4	36	0	M.L.	0	M.L.	1	218	1	1	1	1	26
5	32	0	M.L.	0	M.L.	1	129	1	1	1	1	3
6	34	0	M.L.	1	207	1	10	1	2	1	3	226
7	38	0	M.L.	1	115	1	15	1	4	1	4	239
8	31	0	M.L.	1	5	1	3	1	2	1	2	0
9	49	0	M.L.	0	T.L.	0	T.L.	1	8	1	8	20
10	34	0	M.L.	1	87	1	9	1	1	1	1	7
Total		0	3600	7	1136	9	399	10	3	10	3	156.2

Table 3
Performance of the decomposition approaches on the real instances.

Idx	n	DEC1-CB			DEC1			DEC2-CB			DEC2			DEC3-CB				DEC3			
		# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C1	# C2	# opt	T (s)	# C1	# C2
1	38	1	89	0	1	15	0	1	3	0	1	3	0	1	3	0	0	1	3	0	0
2	28	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	0	1	1	0	0
3	35	1	3	0	1	3	0	1	3	0	1	3	0	1	3	0	0	1	3	0	0
4	36	0	T.L.	1	1	111	0	1	2	0	1	1	0	1	2	5	0	1	2	11	0
5	32	1	3107	2	1	151	0	1	1	0	1	1	0	1	1	2	0	1	1	1	0
6	34	1	19	0	1	2	0	1	2	0	1	2	0	1	3	0	0	1	2	0	0
7	38	0	T.L.	3	1	4	0	1	4	0	1	4	0	1	4	4	0	1	4	1	0
8	31	1	3	0	1	3	0	1	3	0	1	2	0	1	3	0	0	1	3	0	0
9	49	0	T.L.	20	1	1847	24	1	331	0	1	49	0	1	10	34	0	1	17	24	0
10	34	1	27	0	1	2	0	1	1	0	1	1	0	1	1	0	0	1	1	0	0
Total		7	1405	2.6	10	214	2.4	10	35	0	10	7	0	10	3	4.5	0	10	4	3.7	0

instance run ("T.L." means that the run was terminated because of the time limit and "M.L." indicates that the run was aborted because of memory limitation), and, when relevant, the number of no-good cuts required ("# C"). The last line in the table provides a summary of the results obtained by each method: the total number of instances solved to optimality, the average computation time (M.L. and T.L. are counted as unsolved in 3600s), and the average number of no-good cuts required.

The results in Table 2 outline the importance of using the preprocessing techniques on the compact model as **COMPACT-NOP** was not able to solve any instance to optimality because of memory limitation and **COMPACT** solved two instances less than **COMPACT-CSP** while also using three times more computing time. The two sequential approaches obtained a solution for every instance in less than 10 s, which turned out to be optimal. Note that a proof of optimality cannot be provided by the sequential approaches unless a solution packing every item is found, which was the case for all real instances except I9. We also observe that even though **SEQ-KEP** required many no-good cuts in order to forbid the columns exceeding the truck height, the approach remains competitive.

The results in Table 3 clearly show that the decomposition approaches do not all display the same computational behavior when solving our loading problem. The worst approach, **DEC1-CB**, failed to solve three real instances. Further analyses showed that the **MP1** solutions heuristically found by the solver took a long time to be verified by **SP1** while also being of very poor quality (e.g., each column in the heuristic solution was composed of only one item). That problematic behavior appeared less frequently in **DEC1** as **SP1** was only called to verify optimal **MP1** solutions. Nevertheless, undesirable **DEC1** behaviors were still observed for the instances in which every item could be packed because there is no incentive in **MP1** to find an "easy to pack" solution when more than one optimal solution exists. This is why we introduced an ad hoc **DEC1** refinement that incorporates a hierarchical component in **MP1** in order to select the solution that is the most likely to be feasible for **SP1** (we opted for the one with minimum sum of bottom item areas) among all the optimal **MP1** solutions. As far as **DEC2-CB** and **DEC2** are concerned, both methods could solve all instances to optimality. We point out that **DEC2-CB** took more time than **DEC2** to solve I9. This was not due to supplementary **SP2** calls to verify heuristic **MP2** solutions as the time required by **SP2** was negligible. Such a time difference between the two methods could be due, for example, to solver randomness or to the fact that key features are deactivated in the solver when callbacks are activated. We investigated further the topic by testing a version of **DEC2** that uses a dummy callback. The fact that such a version took 330 s to solve I9, just like **DEC2-CB**, goes in favor of the second hypothesis, even though one should be cautious when deriving general conclusions based on such limited experiments. Finally, we observe that both **DEC3** and **DEC3-CB** could solve all ten instances to optimality in a very short time. We notice that, in both methods, several **NGC3-1** were required whereas no **NGC3-2** was necessary.

7.2. Random instances

As the company instances were relatively easy to solve for the best decomposition approaches, we designed four alternative sets of instances with the goal of assessing how the computational behavior of the proposed methods evolves when key instance features change. Even though these four datasets were randomly generated, they were created in order to closely mimic the most difficult real instances: our "reference group" is composed of ten instances with $n = 50$ items of $m = 5$ types, the truck dimensions were $W \times L \times H = 250 \times 1375 \times 275$, the width and length (also called packing dimensions) of the items were uniformly distributed in the range $[\frac{W}{4}; \frac{W}{2}]$, the height of the items was uniformly distributed in the range $[\frac{H}{4}; H]$, the item weight was never binding, 80% of the items were stackable, 67% of the stackable items had a flat surface, and the maximum number of items allowed per column S was 3. Between 80% and 100% of the items were included in an optimal solution, meaning that the knapsack component of the problem did not always matter. For each randomly generated instance set, we created four subsets of ten instances each where one

Table 4
Performance of the compact model and the sequential approaches on random instances varying $W \times L \times H$.

$W \times L \times H$	COMPACT-NOP		COMPACT		COMPACT-CSP		SEQ		SEQ-KEP		
	# opt	T (s)	# opt	T (s)	# opt	T (s)	# opt	T (s)	# opt	T (s)	# C
$10 \times 55 \times 11$	9	372	8	725	10	1	10	1	10	7	219.9
$20 \times 110 \times 22$	8	1039	7	1238	9	500	10	53	10	19	99
$50 \times 275 \times 55$	0	3600	4	2595	4	2191	10	396	10	380	52.5
$150 \times 825 \times 165$	0	3600	0	3600	1	3271	10	1327	10	1717	50.7
$*250 \times 1375 \times 275$	0	3600	0	3600	0	3600	9	1153	9	1135	69.2
Total	17	2442	19	2352	24	1912	49	586	49	651	98.3

Table 5
Performance of the decomposition approaches on random instances varying $W \times L \times H$.

$W \times L \times H$	DEC1-CB			DEC1			DEC2-CB			DEC2			DEC3-CB				DEC3			
	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C1	# C2	# opt	T (s)	# C1	# C2
$10 \times 55 \times 11$	10	22	4.1	10	0	2	10	1	0	10	0	0	10	0	4.1	0	10	0	2.2	0
$20 \times 110 \times 22$	9	381	4.2	9	369	14.8	10	111	0	10	4	0	10	2	19.6	0	10	9	18.1	0
$50 \times 275 \times 55$	4	2387	1.3	5	1851	21.9	9	462	0.1	10	10	0	10	21	91.7	0.3	10	29	87.4	0.1
$150 \times 825 \times 165$	3	2617	1.7	3	2537	6.8	6	1551	0.6	8	795	0.3	10	480	317.1	1.9	10	416	326.2	0.8
$*250 \times 1375 \times 275$	0	3600	7.9	2	3093	8.6	6	1479	2.5	9	472	0.3	9	879	508.4	1.5	9	634	392.6	0.1
Total	26	1801	3.8	29	1570	10.8	41	721	0.6	47	257	0.1	49	276	188.2	0.7	49	218	165.3	0.2

key parameter varied each time. We then compared the results obtained by the proposed methods with the results obtained on the ten instances of the reference group. The reference group is identified by an asterisk (*) in the tables.

7.2.1. Varying the truck dimensions $W \times L \times H$

Our first set of experiments on random instances aims at evaluating the impact of the truck dimensions on the performance of the proposed methods. We tested four $W \times L \times H$ dimensions $\{(10 \times 55 \times 11), (20 \times 110 \times 22), (50 \times 275 \times 55), (150 \times 825 \times 165)\}$ and reported the results of the 11 methods in Tables 4 and 5. Note that a truck with dimensions around 250 × 1375 × 275 cm is fairly standard in Europe. Smaller truck dimensions were tested in order to evaluate the performance of the proposed algorithms when solving instances for which very strong preprocessing techniques can be applied. For example, if one solves an instance where every item dimension is a multiple of five, then a valid preprocessing technique simply divides all dimensions by five, resulting in a truck with size $\lfloor \frac{250}{5} \rfloor \times \lfloor \frac{1375}{5} \rfloor \times \lfloor \frac{275}{5} \rfloor = 50 \times 275 \times 55$.

Our experiments, whose outcomes are displayed in Table 4, show that small-sized instances do not necessarily require advanced solution methods as all instances but three among the two smallest classes could be solved by **COMPACT-NOP**. Interestingly, we observe that **COMPACT** left five instances unsolved in that group while the model required less variables (85 212 versus 147 273 for class 20 × 110 × 22) and constraints (1554 versus 2674) on average than **COMPACT-NOP**. We attribute this unexpected behavior to solver randomness. The results for larger classes are more in line with our expectations as **COMPACT** was able to solve four instances in class 50 × 275 × 55 whereas **COMPACT-NOP** could not solve any. **COMPACT-CSP** solved 5 instances more than **COMPACT**, which can be explained by a significant decrease in the number of variables (7254 versus 85 212 for class 20 × 110 × 22) and a small decrease in the number of constraints (1512 versus 1554). Once more, the two sequential approaches obtained outstanding results (all instances but one were solved to optimality) even though a certificate of optimality could not be provided. Based on the experiments performed so far, it is clear that method **COMPACT-CSP** outperforms both **COMPACT** and **COMPACT-NOP**, which is why we no longer report the results of the last two models in the next sets of experiments. Similarly, because methods **SEQ** and **SEQ-KEP** display similar (if not identical) performance, we only report the results of the former model in the next sets of experiments.

The results in Table 5 clearly show that, once more, not all decomposition approaches are equally effective. As observed in the experiments for the real instances, **DEC1-CB** is the worst decomposition method since it could only solve 26 out of the 50 tested instances. Further analyses showed (again) that this poor performance could (partially) be attributed to the fact that **SP1** was called to verify several sub-optimal **MP1** solutions, which was not the case in **DEC1**. Methods **DEC3** and **DEC3-CB** could both solve all 50 instances but one. Interestingly, the unsolved instance was not the same for the two methods, meaning that all instances could be solved to optimality by at least one approach. Methods **DEC2** and **DEC2-CB** came third and fourth, respectively. Overall, we observe that no-good cuts of type 1 (i.e., **NGC1** and **NGC3-1**) are more often required than no-good cuts of type 2 (i.e., **NGC2** and **NGC3-2**) on average, which is a phenomenon that is expected and that was already observed in previous research [3]. One would also expect that a decomposition method requires more no-good cuts when callbacks are used, which was indeed the case for both **DEC2-CB** and **DEC3-CB**. The reason why such an observation was not made for **DEC1-CB** was simply because the method was often stopped due to the time limit: for example, for the unsolved 20 × 110 × 22 instance, **DEC1-CB** only had time to call **SP1** 37 times while **DEC1** could call it 148 times, which explains the higher **NGC1** average for the latter approach. Based on these experiments, it is clear that methods **DEC1-CB** and **DEC1** are not competitive, which is why we no longer report their results in the next sets of experiments.

Overall, we observe that even though **DEC2** and **DEC3** (and their respective callback versions) scale relatively well, instances with larger truck dimensions are more difficult to solve than instances with smaller dimensions.

7.2.2. Varying the number of item types m

Our second set of experiments on random instances aims at evaluating the impact of the number of item types on the performance of the selected methods. We tested four number of item types $m \in \{2, 3, 7, 10\}$ and reported the results of the six selected methods in Table 6.

The results reported in Table 6 show that our methods work very well when the number of item types is less than or equal to three. These instances usually allow a very strong preprocessing when calling model (14)–(18) to reduce the item and truck dimensions. As expected, the most difficult instances are the ones with the largest number of item types, probably because the modeling tool exploiting the multiplicity of the item packing dimensions is less effective. Nevertheless, only three instances could not be solved by any of our methods (one with $m = 7$ and two with $m = 10$), which indicates that our algorithms are relatively resilient to an increase in the number of item types. Interestingly, **DEC2** is now the approach that performs best, closely followed by **DEC3-CB** and by **DEC3**. We point out that even though **SEQ** performs relatively well, the method does not seem to offer a competitive advantage compared to **DEC2** as the latter method solves

Table 6
Performance of the selected methods on random instances varying m.

m	COMPACT-CSP		SEQ		DEC2-CB			DEC2			DEC3-CB				DEC3			
	# opt	T (s)	# opt	T (s)	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C1	# C2	# opt	T (s)	# C1	# C2
2	10	116	10	30	10	30	0	10	29	0	10	29	0.8	0	10	29	0.8	0
3	3	2639	10	39	10	31	0	10	24	0	10	25	28.1	0	10	27	27	0
*5	0	3600	9	1153	6	1479	2.5	9	472	0.3	9	879	508.4	1.5	9	634	392.6	0.1
7	0	3600	7	987	8	1134	0.5	9	736	0.2	9	629	543	2	7	1148	540.5	0
10	0	3600	7	2071	6	1601	1.6	8	909	0	6	1646	224.5	6.6	6	1526	184	0.6
Total	13	2711	43	856	40	855	0.9	46	434	0.1	44	642	261	2	42	673	229	0.1

Table 7
Performance of the selected methods on random instances varying n.

n	COMPACT-CSP		SEQ		DEC2-CB			DEC2			DEC3-CB				DEC3			
	# opt	T (s)	# opt	T (s)	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C1	# C2	# opt	T (s)	# C1	# C2
30	6	2005	10	2	10	2	0	10	2	0	10	3	0	0	10	2	0	0
40	1	3278	10	34	10	20	0	10	7	0	10	8	8.3	0	10	7	8.2	0
*50	0	3600	9	1153	6	1479	2.5	9	472	0.3	9	879	508.4	1.5	9	634	392.6	0.1
60	0	3600	6	2566	3	2698	0.6	7	1399	0.3	8	927	313.9	0.1	9	556	307.1	5.2
70	0	3600	7	2330	5	2037	1.9	6	1551	0.4	7	1397	835	1.7	6	1612	350.6	0
Total	7	3217	42	1217	34	1247	1	42	687	0.2	44	643	333.1	0.7	44	562	211.7	1.1

more instances, requires less computation time on average, and does terminate with a certificate of optimality.

7.2.3. Varying the number of items n

Our third set of experiments on random instances aims at evaluating the impact of the number of items on the performance of the selected methods. We tested four number of items $n \in \{30, 40, 60, 70\}$ and reported the results of the six selected methods in Table 7.

Based on the results displayed in Table 7, we can make a number of interesting remarks:

- Instances with as few as 30 items are unsolved by **COMPACT-CSP**. This may come as a surprise since every optimal solution for this instance class packs every item (there even exists optimal solutions where a large portion of the truck remains empty). This indicates that the model size can be a challenge for **COMPACT-CSP**, even for the instances without a knapsack component;
- **DEC3** and **DEC3-CB** solve all but six instances each. Some instances with 70 items can still be solved, which is a good sign for the company as it usually deals with instances between 30 and 50 items;
- Only four instances remain unsolved, one with $n = 60$ items and three with $n = 70$ items;
- The use of callbacks seems very detrimental to **DEC2** whereas it does not appear very impactful for **DEC3**.

7.2.4. Varying the item packing dimensions w_i and l_i range

Our last set of experiments on random instances aims at evaluating the impact of the item packing dimensions w_i and l_i range on the performance of the selected methods. We tested four ranges for the item packing dimensions $w_i, l_i \in \{[\frac{W}{2}, \frac{W}{2}], [\frac{W}{3}, \frac{W}{2}], [\frac{W}{5}, \frac{W}{2}], [\frac{W}{6}, \frac{W}{2}]\}$ and reported the results of the six selected methods in Table 8.

The results displayed in Table 8 outline an interesting behavior as the performance of an approach when solving a given instance I seems to depend both on the packing dimensions of the items in I and on whether or not I has a knapsack component. We remind the reader that smaller items are detrimental for the preprocessing techniques introduced in Section 4, meaning that instances with smaller items usually require mathematical models (for solving the overall problem or for solving its MP and SP components) involving more variables and constraints, all other parameters being equal. We observe that:

- Instances in class $[\frac{W}{2}, \frac{W}{2}]$ are particularly easy for all methods and do not present any computational challenge. We point out that, for this instance class, the computational time was entirely spent in the preprocessing for all methods;
- Instances in class $[\frac{W}{3}, \frac{W}{2}]$ are also relatively easy except for **COMPACT-CSP**. These instances possess a strong knapsack component (i.e., all items cannot be packed in an optimal solution) while being composed of relatively large items;
- Instances in class $[\frac{W}{4}, \frac{W}{2}]$ are more difficult as they are composed of slightly smaller items compared to the previous class while they often still possess a knapsack component;
- Instances in class $[\frac{W}{5}, \frac{W}{2}]$ seem slightly easier. Even if they are composed of smaller items compared to the previous class, only half of these instances possess a knapsack component;
- Instances in class $[\frac{W}{6}, \frac{W}{2}]$ seem to be easy to solve again for all methods except **COMPACT-CSP**. The fact that they are composed of the smallest items seems to be counterbalanced by the fact that none of these instances have a knapsack component (i.e., the items are so small that they can all be packed in the truck);

Overall, the most difficult instances are the ones composed of many small and heterogeneous items that cannot fit all together in the truck.

7.3. Supplementary features

Even though the proposed approaches provide solutions that satisfy Nunner logistics' main criteria, real-world container loading problems often involve problem-specific features that require an additional set of ad hoc constraints to be modeled. We explain in this section how supplementary real-world features which are also relevant for Nunner and other logistic companies can be integrated into our algorithms.

7.3.1. Customer multiplicity

One aspect that Nunner logistics would like to take into account in the packing layout is customer multiplicity. If the company needs to deliver some items to multiple customers in the same day, two supplementary constraints should be taken into account when creating the packing layout: (i) an item can only be stacked on top of another item if both items must be delivered to the same customer and (ii) given a customer ordering (based on the delivery route of the truck), the items that must be delivered to the next customer in the ordering must be "reachable" without having to move the items that belong to one of

Table 8
Performance of the selected methods on random instances varying the range of the item width and length.

w_i and l_i range	COMPACT-CSP		SEQ		DEC2-CB			DEC2			DEC3-CB				DEC3			
	# opt	T (s)	# opt	T (s)	# opt	T (s)	# C	# opt	T (s)	# C	# opt	T (s)	# C1	# C2	# opt	T (s)	# C1	# C2
$[\frac{W}{2}, \frac{W}{2}]$	10	3	10	3	10	3	0	10	3	0	10	2	0	0	10	2	0	0
$[\frac{W}{3}, \frac{W}{2}]$	8	736	7	5	10	8	0	10	3	0	10	3	4.8	0	10	3	4.8	0
$*[\frac{W}{4}, \frac{W}{2}]$	0	3600	9	1153	6	1479	2.5	9	472	0.3	9	879	508.4	1.5	9	634	392.6	0.1
$[\frac{W}{5}, \frac{W}{2}]$	0	3600	9	1075	7	1166	0.1	9	571	0	9	462	298.9	0.5	10	309	332.1	0
$[\frac{W}{6}, \frac{W}{2}]$	0	3600	10	94	10	152	0	10	150	0	10	163	0.3	0.2	10	151	0.3	0
Total	18	2308	45	466	43	561	0.5	48	240	0.1	48	302	162.5	0.4	49	220	146	0

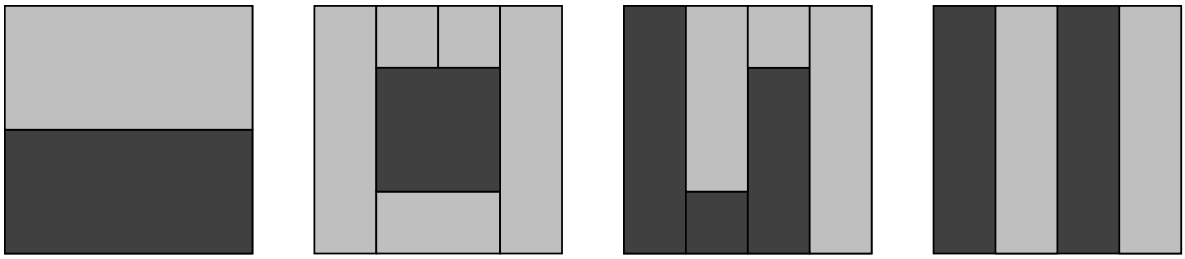


Fig. 3. Four packing configurations for an instance with two customers (2D views seen from above).

the subsequent customers. Such a requirement is known as "unloading constraints" in the literature [36].

Single-customer columns. Forbidding the stacked columns to contain items belonging to two different customers is relatively easy in our problem as one just needs to update the necessary conditions for two items i and i' to be stackable by requiring that, on top of the six conditions introduced in Section 3, both items i and i' should be delivered to the same customer.

Unloading constraints. Making sure that the items belonging to the next customer in the route can all be reached from the back of the truck without having to move any other item is slightly more challenging. First, defining unacceptable packing is not always straightforward. Let us consider, for example, the four packing configurations displayed in Fig. 3 which are 2D views of the truck from above. Assuming that the back of the truck is at the bottom and that the dark gray customer needs to be delivered first, it is clear that the first (most left) configuration is feasible, the second is infeasible, and the third and fourth are both feasible. But if we now consider that the light gray customer needs to be delivered first, while the first configuration is clearly infeasible and the fourth is clearly feasible, there is no obvious answer as far as the second and the third configurations are concerned. Even though the light gray items are all accessible from the back of the truck, one can also argue that the dark gray items may constitute an obstacle.

After some discussions, we adopted the same definition for unloading constraints as the one used by Côté et al. [36], which can be described as follows: a stacked column i (of items) belonging to a customer is allowed to be packed in coordinate (w, l) if every stacked column i' belonging to a subsequent customer is packed in a coordinate (w', l') such that either (i) columns i and i' do not overlap in the x-axis (i.e., either $w + w_i <= w'$ or $w' + w_{i'} <= w$) or (ii) column i is packed closer to the back of the truck than column i' (i.e., $l + l_i \leq l'$). According to such a rule, the two middle configurations are not allowed if the light gray customer must be delivered first. As far as implementation is concerned, such a constraint is easy to implement in **SP2** (and therefore, in **SP3-2**). Recall that, in the two subproblems, all the tasks (i.e., the bottom items) occupying abscissa w are gathered in set $\mathcal{I}_w$ and that an interval variable ivl_i is associated to each task i ($i \in \mathcal{I}$). If we denote by c the customer index (i.e., the position in which a customer is visited in the delivery route) and by c_i the customer index of item i, unloading constraints can be defined as follows:

$$\texttt{endbeforestart}(ivl_i, ivl_{i'}) \qquad i \in \mathcal{I}_w, i' \in \mathcal{I}_w : c_{i'} > c_i, w = 0, \ldots, W-1, \tag{62}$$

which ensures that, given a task i (i.e., a bottom item), every task associated with a subsequent customer that overlaps with i on the x-axis needs to have its starting time (i.e., its y-coordinate) greater than or equal to the starting time of i plus the duration of i.

7.3.2. Generalized fragility constraints

Because Nunner logistics limits the maximum number of items that can be stacked in the same column to $S = 3$, it is possible to incorporate the fragility constraints forbidding a heavy item to be stacked on a light item into the compatibility graph $\mathcal{G}$. By doing so, given the criteria described in Section 3, the sum of the item weights stacked on top of a given bottom item i never exceeds twice the weight of item i (because of the condition stating that the weight of the bottom item must be larger than or equal to the weight of the top item — note that one could also tune that criterion by using an ad hoc coefficient). When one considers larger S values (e.g., 10 or 20), such a strategy does not make sense anymore as an item may be able to accommodate once or twice its weight, but probably not ten or twenty times. These fragility constraints can easily be taken into account in our models by considering a fragility coefficient f_i for each item i ($i = 1, \ldots, n$) indicating the maximum weight allowed to be stacked on top of item i and by adding the following constraints related to the column creation:

$$\sum_{i'=1}^{n} \sum_{s=2}^{S} y_{ii's} k_{i'} \leq \xi_i^P f_i \qquad i = 1, \ldots, n, \tag{63}$$

$$\sum_{i'=1}^{n} \sum_{s'=s+1}^{S} y_{ii's'} k_{i'} \leq \sum_{i'=1}^{n} y_{ii's} f_{i'} \quad i = 1, \ldots, n, s = 2, \ldots, S-2, \tag{64}$$

where constraints (63) make sure that the total weight on top of item i does not exceed its fragility if item i is used as a bottom item and

Table 9
Performance of the DEC3 on the real instances taking into account supplementary features.

Idx	# cust	DEC3-0-0					DEC3-1-0					DEC3-0-1					DEC3-1-1				
		# opt	T (s)	opt val	# C1	# C2	# opt	T (s)	opt val	# C1	# C2	# opt	T (s)	opt val	# C1	# C2	# opt	T (s)	opt val	# C1	# C2
1	2	1	3	401 925	0	0	1	3	401 925	0	1	1	3	401 925	0	0	1	3	401 925	0	1
2	1	1	1	308 400	0	0	1	1	308 400	0	0	1	1	308 400	0	0	1	1	308 400	0	0
3	2	1	3	339 840	0	0	1	3	339 840	0	2	1	3	339 840	0	0	1	3	339 840	0	2
4	4	1	2	396 738	11	0	1	2	396 738	9	21	1	2	396 738	10	0	1	2	396 738	6	21
5	2	1	1	363 394	1	0	1	1	363 394	7	0	1	1	363 394	3	0	1	1	363 394	3	0
6	3	1	2	414 307	0	0	1	2	414 307	0	0	1	2	414 307	0	0	1	2	414 307	0	0
7	4	1	4	375 550	1	0	1	4	375 550	0	27	1	4	375 550	1	0	1	4	375 550	0	27
8	2	1	3	326 700	0	0	1	2	326 700	0	0	1	3	326 700	0	0	1	2	326 700	0	0
9	1	1	17	469 450	24	0	1	17	469 450	24	0	1	45	452 150	77	0	1	46	452 150	77	0
10	2	1	1	379 800	0	0	1	1	379 800	0	0	1	1	379 800	0	0	1	1	379 800	0	0
Total	2.3	10	4	377 610	3.7	0	10	4	377 610	4	5.1	10	6	375 880	9.1	0	10	6	375 880	8.6	5.1

constraints (64) make sure that the sum of the item weights in position $s+1, s+2, \ldots, S$ in column i does not exceed the fragility of the item packed in position s in that same column. We point out that constraints (64) are neither necessary for $s = S-1$ (as those can be included in the compatibility graph $\mathcal{G}$) nor for $s = S$ (as packing an item in position $S+1$ is not allowed).

We report in Table 9 the results of **DEC3** when none (**DEC3-0-0**), one (**DEC3-1-0** for customer multiplicity and **DEC3-0-1** for generalized fragility constraints), and the two features (**DEC3-1-1**) are taken into account on the ten real instances. The number of customers varied between one and four (this information was available in the daily instances provided by Nunner) and values f_i were set to w_i for all items (this information was not available, so we used an approximation). In the table, column "# cust" reports the number of customers in the instance and column "opt val" indicates the optimal solution value of the instance.

Based on the similarity between the results obtained by **DEC3-0-0** and **DEC3-1-0**, we can state that taking customer multiplicity into account in the real instances is not particularly difficult. We observe, however, an increase in the number of **NGC3-2** cuts required (5.1 on average for **DEC3-1-0** versus 0 for **DEC3-0-0**), which does not come as a surprise considering that the newly added unloading constraints belong to **SP3-2**. This increase is more pronounced in the instances with four customers. Interestingly, we did not notice a decrease in the optimal solution value, which is an observation that is probably only valid for the tested real instances. Indeed, for every instance except I9, all items could be packed in an optimal solution. In addition, I9 only has one customer, which explains why **DEC3-0-0** and **DEC3-1-0** obtained identical results for that instance. We tested **DEC3-1-0** on a modified version of I9 in which the existing items were split into two customers and we observed that the instance became much more difficult to solve: it required more than nine hours of computation time, 29 **NGC3-1** cuts, and 11 267 **NGC3-2** cuts. We also observed a decrease in the objective solution value from 469 450 to 467 750.

By comparing the results obtained by **DEC3-0-0** with those obtained by **DEC3-0-1**, we can state that using generalized fragility constraints does not necessarily deteriorate the performance of the model, but it may decrease the value of the optimal solution This does not come as a surprise as **DEC3-1-0** is more constrained than **DEC3-0-0** when f_i is set to w_i. We did not observe any unexpected interactions between customer multiplicity and generalized fragility constraints (i.e., the results obtained by **DEC3-1-1** were in line with those obtained by **DEC3-1-0** and **DEC3-0-1**), but we believe that this is due to the relative "simplicity" of the real instances provided by the company.

To conclude this section, we compare in Fig. 4 the optimal packing layouts obtained by **DEC3-1-1** and by a version of **DEC3-0-0** where multi-customer columns are forbidden but where unloading constraints are not enforced. The figure shows 2D views of the truck from above, the back of the truck is located at the bottom of the figure, and only the bottom items are displayed in the packing layout (but the index of every item composing the column appears in the figure). The item color represents the customer index and lighter colors identify the customers that must be delivered earlier in the route. We can see that the packing layout obtained by **DEC3-0-0** for instances I4, I5, and I7 does not satisfy the unloading constraints, but that **DEC3-1-1** is able to find a packing layout using the same items that does satisfy those constraints. We also observe in I9 solutions that some 3-item columns that are allowed in **DEC3-0-0** are forbidden in **DEC3-1-1** (e.g., the column composed of items 3, 25, and 4 where $k_3 = 82$, $k_{25} = 60$, and $k_4 = 40$) whereas others are feasible in both approaches (e.g., the column composed of items 18, 3, and 4 where $k_{18} = 265$, $k_3 = 82$, and $k_4 = 40$).

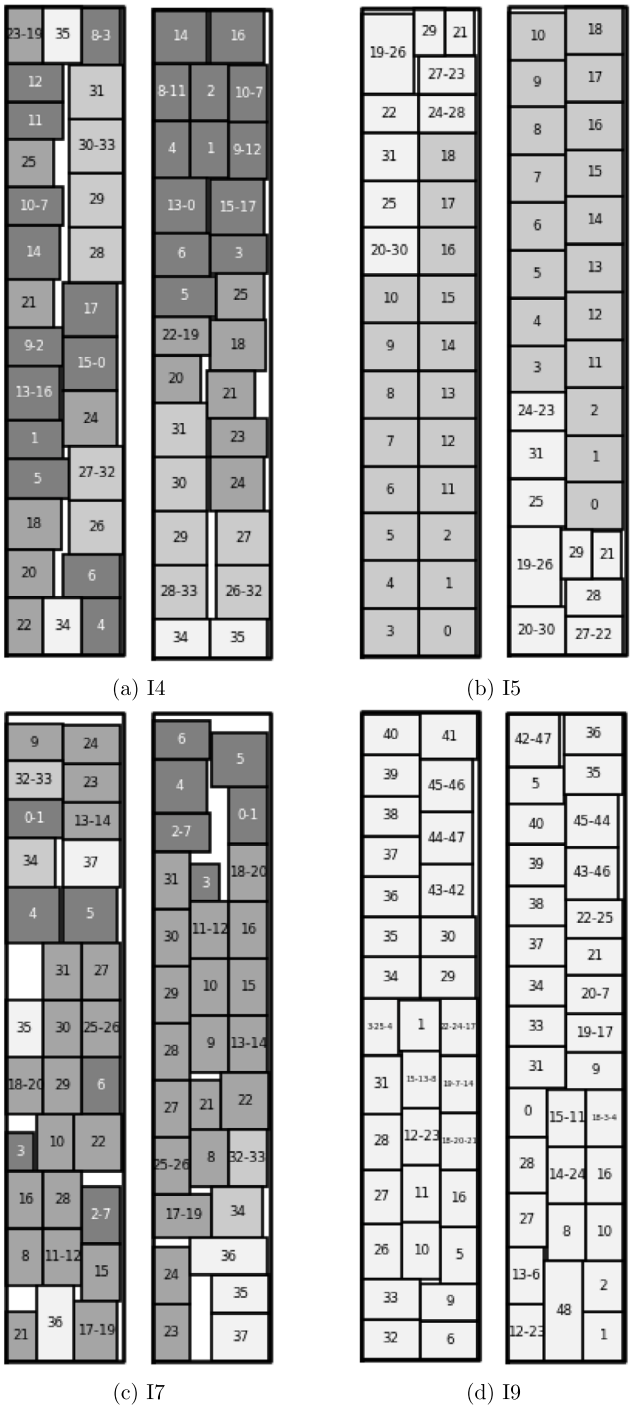
8. Conclusion

We studied various solution methods aimed at solving exactly a real-world single container loading problem with stacking constraints and medium-sized weakly heterogeneous items. After introducing a compact integer programming model together with some innovative ad hoc reduction procedures, we described a sequential approach and three decomposition algorithms to solve the problem. We empirically evaluated each of the proposed methods on a set of real and randomly generated instances and made a number of relevant observations.

From the company point of view, it is interesting to know that (i) the (non-exact) sequential strategy that is currently used often leads to an optimal solution provided that each component of the approach is solved to optimality, (ii) the company daily instances are relatively easy to solve to optimality with an effective decomposition approach, and (iii) customer multiplicity and generalized fragility constraints can easily be taken into account in the company daily instances.

From the optimization point of view, it is interesting to observe that (i) there is a connection between the studied loading problem and the kidney exchange problem, (ii) under certain conditions, there is a better preprocessing for adjusting the item dimensions than the one based on dynamic programming, (iii) item multiplicity can be exploited even if only a subset of the item features (in our case, the packing dimensions) is identical, (iv) there are more than one decomposition method that can be used to solve a loading problem, and some of these decomposition methods are faster than others, and (v) using callbacks does not necessarily speed-up (and can even be detrimental to) a decomposition approach.

Overall, the proposed methods provide satisfactory results for the optimization problem faced by Nunner logistics considering that realistic instances with a common size 250 cm × 1375 cm × 275 cm truck and up to 70 items could be solved to optimality. This is already an achievement compared to existing exact decomposition methods for two-dimensional and three-dimensional packing literature considering that the largest-sized two-dimensional orthogonal packing instance solved by Côté et al. [36] had $W \times L = 14 \times 130$ and 36.9 items on average, the largest-sized two-dimensional strip packing instances solved by Côté et al. [2] and Delorme et al. [4] had either $W = 1000$ and 50 items or $W = 100$ and 100 items, the largest-sized two-dimensional bin packing instances solved by Polyakovskiy and M'Hallah [23] had $W \times L = 100 \times 100$ and 100 items while those solved by Côté et al. [3] had $W \times L = 300 \times 300$ and 100 items, and the largest-sized three-dimensional container loading instances solved by Do Nascimento et al. [22] had either $W \times L \times H = 100 \times 100 \times 100$ and 14 items or $W \times L \times H = 10 \times 10 \times 10$ and 91 items. Note that such comparisons are for indicative purposes only since the optimization problems solved in



(a) I4 (b) I5 (c) I7 (d) I9

Fig. 4. 2D views seen from above of the optimal **DEC3-0-0** (left) and **DEC3-1-1** (right) packing layouts.

each of these papers were not identical to the one we studied, even though they shared some similarities. Indeed, we remind the reader that our approaches exploited some problem features that were specific to the problem encountered by Nunner, especially the fact that the items were relatively large when compared to the vehicle, the fact that the items were weakly heterogeneous on the width and the length dimensions, but not on the height dimension, and the fact that only one item per layer was allowed in a column. If the items were also weakly heterogeneous on the height dimension and if there were small-sized items, then it would make more sense for the company to also consider allowing multiple items per layer, as was done, for example, in the work of Toffolo et al. [21].

As far as future research directions are concerned, we mention in particular (i) the integration of column stability (i.e., making sure that the item columns do not fall when the truck moves), (ii) the search for more effective unloading constraints (an artificial instance with two customers required more than nine hours of computation time to be solved), and (iii) the creation of a hyper-algorithm able to select which method among the ones proposed in this work is better suited to solve a given instance (following the recent trend that aims at incorporating machine learning techniques into optimization algorithms, see [38]).

CRediT authorship contribution statement

Maxence Delorme: Conceptualization, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Joris Wagenaar:** Conceptualization, Data curation, Formal analysis, Validation, Writing – review & editing.

Declaration of competing interest

None.

Data availability

Data will be made available on request.

Acknowledgments

We would like to thank Nunner logistics and Sem Valks for providing us with the problem description and the data used in this research. Thanks are due to SURF for providing access to SURF Research Cloud (the computational resources used in the experiments) through the pilot grant framework. Thanks are also due to the Dutch ministry of education for financing the authors through the starter grant framework. Finally, we would also like to thank the three anonymous reviewers for their valuable comments that have helped to improve the presentation of this paper.

References

[1] Bischoff EE, Ratcliff MSW. Issues in the development of approaches to container loading. Omega 1995;23:377–90.
[2] Côté J-F, Dell'Amico M, Iori M. Combinatorial benders' cuts for the strip packing problem. Oper Res 2014;62:643–61.
[3] Côté J-F, Haouari M, Iori M. Combinatorial benders decomposition for the two-dimensional bin packing problem. INFORMS J Comput 2021;33:963–78.
[4] Delorme M, Iori M, Martello S. Logic based benders' decomposition for orthogonal stock cutting problems. Comput Oper Res 2017;78:290–8.
[5] Silva E, Oliveira JF, Wäscher G. The pallet loading problem: a review of solution methods and computational experiments. Int Trans Oper Res 2016;23:147–72.
[6] Zhao X, Bennell J, Bektaş T, Dowsland K. A comparative review of 3D container loading algorithms. Int Trans Oper Res 2016;23:287–320.
[7] Bortfeldt A, Wäscher G. Constraints in container loading – a state-of-the-art review. European J Oper Res 2013;229:1–20.
[8] Silva EF, Toffolo TAM, Wauters T. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. Comput Oper Res 2019;109:12–27.
[9] Gendreau M, Iori M, Laporte G, Martello S. A tabu search algorithm for a routing and container loading problem. Transp Sci 2006;40:342–50.
[10] Lim A, Rodrigues B, Wang Y. A multi-faced buildup algorithm for three-dimensional packing problems. Omega 2003;31:471–81.
[11] Tarantilis CD, Zachariadis EE, Kiranoudis CT. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. IEEE Trans Intell Transp Syst 2009;10:255–71.
[12] Şafak Ö, Erdoğan G. A large neighbourhood search algorithm for solving container loading problems. Comput Oper Res 2023;154:106199.
[13] Gajda M, Trivella A, Mansini R, Pisinger D. An optimization approach for a complex real-life container loading problem. Omega 2022;107:102559.
[14] Paquay C, Limbourg S, Schyns M, Oliveira JF. MIP-based constructive heuristics for the three-dimensional bin packing problem with transportation constraints. Int J Prod Res 2018;56:1581–92.

[15] Junqueira L, Morabito R, Yamashita DS. Three-dimensional container loading models with cargo stability and load bearing constraints. Comput Oper Res 2012;39:74–85.
[16] Paquay C, Schyns M, Limbourg S. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. Int Trans Oper Res 2016;23:187–213.
[17] Alonso MT, Alvarez-Valdes R, Iori M, Parreño F, Tamarit JM. Mathematical models for multicontainer loading problems. Omega 2017;66:106–17.
[18] Macedo R, Alves C, Valério de Carvalho JM. Arc-flow model for the two-dimensional guillotine cutting stock problem. Comput Oper Res 2010;37:991–1001.
[19] Haessler RW, Talbot FB. Load planning for shipments of low density products. European J Oper Res 1990;44:289–99.
[20] Gehring H, Bortfeldt A. A genetic algorithm for solving the container loading problem. Int Trans Oper Res 1997;4:401–18.
[21] Toffolo TAM, Esprit E, Wauters T, Berghe GV. A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. European J Oper Res 2017;257:526–38.
[22] Do Nascimento OX, de Queiroz TA, Junqueira L. Practical constraints in the container loading problem: Comprehensive formulations and exact algorithm. Comput Oper Res 2021;128:105186.
[23] Polyakovskiy S, M'Hallah R. Just-in-time two-dimensional bin packing. Omega 2021;102:102311.
[24] Chen C-S, Lee S-M, Shen QS. An analytical model for the container loading problem. European J Oper Res 1995;80:68–76.
[25] Kenmochi M, Imamichi T, Nonobe K, Yagiura M, Nagamochi H. Exact algorithms for the two-dimensional strip packing problem with and without rotations. European J Oper Res 2009;198:73–83.
[26] Alvarez-Valdés R, Parreño F, Tamarit J. A branch and bound algorithm for the strip packing problem. OR Spectr 2009;31:431–59.
[27] Boschetti MA, Mingozzi A, Hadjiconstantinou E. Exact solution techniques for two-dimensional cutting and packing. IMA J Manag Math 2002;13:95–119.
[28] Kartak VM, Ripatti AV, Scheithauer G, Kurz S. Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. Discrete Appl Math 2015;187:120–9.
[29] Christofides N, Whitlock C. An algorithm for two-dimensional cutting problems. Oper Res 1977;25:30–44.
[30] Côté J-F, Iori M. The meet-in-the-middle principle for cutting and packing problems. INFORMS J Comput 2018;30:646–61.
[31] Delorme M, Manlove D, Smeets T. Half-cycle: a new formulation for modelling kidney exchange problems. Oper Res Lett 2023;51:234–41.
[32] Dickerson JP, Manlove DF, Plaut B, Sandholm T, Trimble J. Position-indexed formulations for kidney exchange. In: Proceedings of EC '16: the 17th ACM conference on economics and computation. ACM; 2016, p. 25–42.
[33] Mak-Hau VH. On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. J Combinat Optim 2017;33:35–59.
[34] Abraham DJ, Blum A, Sandholm T. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In: Proceedings of EC '07: the 8th ACM conference on electronic commerce. ACM; 2007, p. 295–304.
[35] Iori M, De Lima VL, Martello S, Miyazawa FK, Monaci M. Exact solution techniques for two-dimensional cutting and packing. European J Oper Res 2021;289:399–415.
[36] Côté J-F, Gendreau M, Potvin JY. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. Oper Res 2014;62:1126–41.
[37] Caselli G, Delorme M, Iori M, Magni CA. Exact algorithms for a parallel machine scheduling problem with workforce and contiguity constraints. Comput Oper Res 2023;106484.
[38] Bengio Y, Lodi A, Prouvost A. Machine learning for combinatorial optimization: a methodological tour d'horizon. European J Oper Res 2021;290:405–21.