Mathematical models based on decision hypergraphs for designing a storage cabinet

Luis Marques, François Clautiaux, Aurélien Froger

Mathematical models based on decision hypergraphs for designing a storage cabinet

Luis Marques[1] François Clautiaux[1] Aurélien Froger[1]

[1]Univ. Bordeaux, CNRS, Inria, IMB, UMR 5251, F-33400 Talence, France

Abstract

We study the problem of designing a cabinet made up of a set of shelves that contain compartments whose contents slide forward on opening. Considering a set of items candidate to be stored in the cabinet over a given time horizon, the problem is to design a set of shelves and a set of compartments on each shelf, and select the items to insert into the compartments. The objective is to maximize the sum of the profits of the selected items. We call our problem the *Storage Cabinet Physical Design* (SCPD) problem. The SCPD problem combines a two-staged two-dimensional knapsack problem for designing the shelves and compartments with a set of temporal knapsack problems for selecting and assigning items to compartments. We formalize the SCPD problem and formulate it as a maximum cost flow problem in a decision hypergraph with additional linear constraints. To reduce the size of this model, we break symmetries, generalize graph compression techniques and exploit dominance rules for precomputing subproblem solutions. We also present a set of valid inequalities to improve the linear relaxation of the model. We empirically show that solving the arc-flow model with our enhancements outperforms solving a compact mixed integer linear programming formulation of the SCPD problem.

Keywords— Cutting and Packing, Integer linear programming, Temporal knapsack, Arc-flow models, Decision hypergraphs

1 Introduction

In this paper, we study the strategic problem of the internal physical design of a storage cabinet. We focus on the design of pull-out compartments within a cabinet of a given size. For this purpose, we are given stock entry and exit dates for a set of (representative) three-dimensional rectangular items, which may come from historical or forecast data. We assume the cabinet may be undersized for the items that are candidates for storage over time. Therefore, we must decide the items to be stored to maximize their profit contribution (a measure of their total utility) and, where appropriate, their allocation to a compartment. Every item assigned to a compartment is only present during a certain time interval, and the capacity of every compartment, considered a renewable resource, is not exceeded at every time step. As the compartments' contents are assumed to slide (or be pushed by a mechanism) forward when opened, the items must have the same width as the compartment in which they are stored, and vertical stacking is forbidden.

Our problem is inspired by a real-life application involving inner-city pharmacies that store and retrieve the most common drugs in an under-the-counter cabinet. A related application is in retail stores with highly limited space and little or no shelving in the store, where a storage cabinet not accessible or even visible to customers, is installed under or behind the counter and stores very popular products. Here, we tackle a simplified version

of such problems as a first step towards optimizing such systems. Our main simplification is to consider a deterministic problem, where the departure and arrival dates of the items are known in advance.

From a combinatorial optimization point of view, we study a three-dimensional variant of the temporal knapsack problem that we refer to as the *Storage Cabinet Physical Design* (SCPD) problem. In the SCPD problem, we consider the optimization of a three-dimensional *storage cabinet* or *cabinet* for the rest of the paper. It involves two types of decisions that are made simultaneously: design decisions and assignment decisions. The design decisions horizontally divide the cabinet into *shelves*, which are vertically divided into *compartments*. The assignment decisions correspond to the selection of requests to be satisfied (called *items* in the remainder), and in this case, their allocation to compartments.

The SCPD problem generalizes the 2-staged two-dimensional knapsack (2TDK) (Lodi and Monaci, 2003) and the temporal knapsack problem (TKP) (Bartlett et al., 2005). To some extent, the SCPD problem also shares similarities with retail shelf space planning problems that integrate design decisions with space allocation decisions (Hübner et al., 2021; Gecili and Parikh, 2022). In a recent comprehensive review of retail shelf space planning problems, Bianchi-Aguiar et al. (2021) argue that the study of such problems would be a valuable practical and scientific contribution to the field.

To our knowledge, jointly making design, item selection, and item allocation decisions while explicitly considering the use of space over time has not yet been studied. Our goal through the study of the SCPD problem is to start to fill this gap. In this paper, we focus on the mathematical modeling of the problem via mixed integer linear programming (MILP) formulations, with a particular focus on arc-flow reformulations in *decision hypergraphs* (Martin et al., 1990).

We now outline the main original contributions of this paper:

- We formulate a relaxation of the SCPD problem as a dynamic program. We derive a decision hypergraph from this dynamic program and reformulate the SCPD problem as a maximum cost flow problem with additional constraints to ensure feasibility. This leads us to introduce an arc-flow formulation of the problem.
- We introduce valid inequalities to strengthen the arc-flow formulation.
- To reduce the size of the formulation, we extend existing graph compression techniques to the case of hypergraphs, adapt dominance rules from the one-dimensional and two-dimensional cutting and packing literature, and propose new ones to detect and exploit the presence of subproblems whose solutions can be easily computed in a preprocessing phase.
- We empirically demonstrate the advantages of solving the arc-flow formulation over a compact MILP formulation on randomly generated instances. The results obtained by solving the arc-flow formulation without any graph reduction or valid inequalities are significantly better than those obtained by solving the compact formulation, with 25 more instances optimally solved out of 200 and an average gap reduced from 12.94% to 3.32%. By reducing the size of the decision hypergraph and breaking certain symmetries, we further improve the performance of the arc-flow formulation, optimally solving 15 additional instances and reducing the average gap from 3.32% to 2.68%. Although the valid inequalities slightly improve the value of the linear relaxation, their introduction led to poorer results in our experiments. We also observe numerically that by correlating the profit of each item with its dimension, the resulting instances are more easily solved.

The rest of this paper is organized as follows. In Section 2, we formally define the SCPD problem, describe some of its properties, and discuss the related literature. In Section 3, we present a compact MILP formulation of the problem. In Section 4, we introduce an arc-flow formulation of the SCPD problem. In Section 5, we describe several improvements to the arc-flow formulation that help to keep its size tractable. In Section 6, we detail how we generated our instances and present the results of our computational experiments. We offer brief concluding remarks in Section 7.

2 Problem statement and related literature

In this section, we give a formal definition of the problem.

2.1 Problem definition

Let $(H_{\max}, W_{\max}, L_{\max})$ be a *storage cabinet* with *height* $H_{\max} \in \mathbb{N}$, *width* $W_{\max} \in \mathbb{N}$ and *length* $L_{\max} \in \mathbb{N}$. Let $\mathcal{I}$ be a set of *items*. Each item $i \in \mathcal{I}$ has a *height* $h_i \in \mathbb{N}$, a *width* $w_i \in \mathbb{N}$, a *length* $\ell_i \in \mathbb{N}$, a *profit* $p_i \in \mathbb{R}_+$ and a *time interval* $[s_i, s_i + d_i)$ with $s_i \in \mathbb{N}$ and $d_i \in \mathbb{N}_{>0}$, during which the item, if selected, is present in the cabinet. We denote by $\mathcal{T} = \cup_{i \in \mathcal{I}} \{s_i, s_i + d_i\}$ the set of *time steps* to be considered.

In the problem, we define a *shelf* ψ as a three-dimensional rectangular object of size $(h_\psi, W_{\max}, L_{\max})$, where $h_\psi \leq H_{\max}$, and we define a compartment ϕ as a three-dimensional rectangular object of size $(h_\phi, w_\phi, L_{\max})$, where $h_\phi \leq H_{\max}$, $w_\phi \leq W_{\max}$. For the sake of simplicity and clarity, note that we abuse the notation h and w for the heights and widths of shelves, compartments, and items, but afterward, the context will make it easy to understand what is being referred to.

A solution to the problem is characterized by a set of shelves, a set of compartments, an assignment of the compartments to the shelves, and an assignment of a subset of items to the compartments. An item can be assigned to a compartment only if they have the same width. This constraint stems from a technical feature of the storage device. To retrieve an item, it is pushed along its compartment. If its width is less than that of the compartment, the item would rotate, which could block the mechanism. We also forbid items to be stacked vertically. The position of the items in a compartment is not fixed, i.e., we allow items to be shifted towards the compartment entrance at every time step. This makes the length of a compartment a renewable resource.

For clarity and precision, we formalize the decision and optimization versions of the problem.

Problem 1 (Storage Cabinet Physical Design decision problem (**SCPD-Decision problem**))**.** *Given* $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}, B\big)$ *with* $B \in \mathbb{R}_+$*, the Storage Cabinet Physical Design decision (SCPD-Decision) problem is formulated as follows: is there* $(\Psi, \Phi, \mathcal{I}', \mu, \rho)$*, where* Ψ *is a set of shelves,* Φ *a set of compartments,* $\mathcal{I}' \subseteq \mathcal{I}$ *a subset of items,* $\mu : \Phi \to \Psi$ *a mapping of the compartments to the shelves, and* $\rho : \mathcal{I}' \to \Phi$ *a mapping of the selected items to the compartments, which satisfies the following six conditions?*

1. *The total profit of the items in* $\mathcal{I}'$ *is at least* B*:*

$$\sum_{i \in \mathcal{I}'} p_i \geq B.$$

2. *The sum of the heights of the shelves in* Ψ *is less than or equal to* $H_{\max}$*:*

$$\sum_{\psi \in \Psi} h_\psi \leq H_{\max}.$$

3. *A compartment* ϕ *is assigned to a shelf* $\mu(\phi)$ *whose height is greater than or equal to its height:*

$$\forall \phi \in \Phi, \quad h_\phi \leq h_{\mu(\phi)}.$$

4. *For each shelf* ψ*, the sum of the widths of the compartments assigned to* ψ *is less than or equal to* $W_{\max}$*:*

$$\forall \psi \in \Psi, \quad \sum_{\phi \in \Phi : \mu(\phi) = \psi} w_\phi \leq W_{\max}.$$

5. *For each compartment* ϕ*, the items assigned to* ϕ *have a height less than or equal to the height of* ϕ *and a width equal to the width of* ϕ*:*

$$\forall \phi \in \Phi, i \in \mathcal{I}', \quad h_i \leq h_{\rho(i)} \text{ and } w_i = w_{\rho(i)}.$$

6. *For each compartment ϕ and each time step t, the sum of the lengths of the items in ϕ at t is less than or equal to $L_{\max}$:*

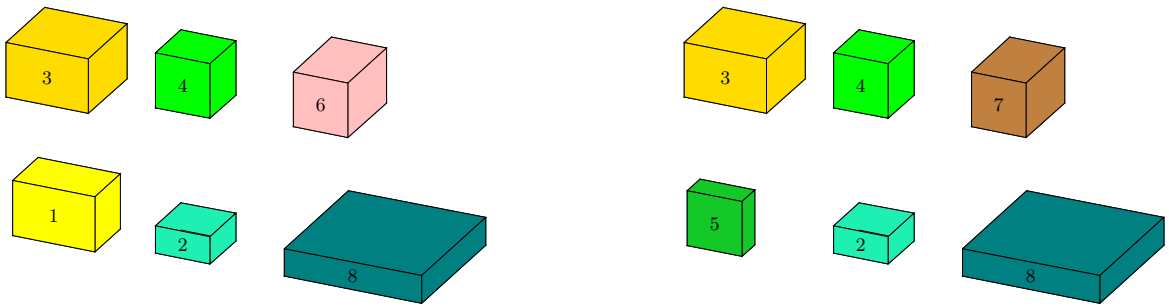
$$\forall t \in \mathcal{T}, \forall \phi \in \Phi, \quad \sum_{i \in \mathcal{I}' : \rho(i) = \phi, s_i \leq t < s_i + d_i} \ell_i \leq L_{\max}.$$

Table 1, Figure 1 and Figure 2 show an instance of the SCPD problem with eight items and two time steps (0 and 1), and a feasible solution. In this instance, if selected, items 1 and 6 are present only during time step 0, items 2, 3, 4 and 8 are present at time steps 0 and 1, and items 5 and 7 are present at time step 1 only. The solution we present in Figure 2 has two shelves. The first shelf has two compartments, and items 1, 2, 3, 4 and 5 are assigned to a compartment in this shelf. The second shelf has one compartment to which item 8 is assigned. Items 6 and 7 are not selected in the solution. This solution has cost 69, equal to the sum of the profits of the selected items. The gray parts in Figure 2 represent wasted space.

$H_{\max}$	$W_{\max}$	$L_{\max}$
3	6	5

i	h_i	w_i	ℓ_i	p_i	s_i	d_i
1	2	3	2	10	0	1
2	1	2	2	7	0	2
3	2	3	3	12	0	2
4	2	2	2	15	0	2
5	2	2	1	24	1	1
6	2	2	3	3	0	1
7	2	2	3	5	1	1
8	1	5	5	1	0	2

Table 1: An SCPD instance with eight items



(a) $t = 0$ (b) $t = 1$

Figure 1: The presence of items by time step for the instance described in Table 1

(a) $t = 0$ (b) $t = 1$

Figure 2: A feasible solution to the instance of Table 1

Problem 2 (Storage Cabinet Physical Design optimization problem **(SCPD problem)**). *Given* $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$*, the Storage Cabinet Physical Design optimization (SCPD) problem consists in finding the largest value*

of $B \in \mathbb{R}_+$ such that the answer to the SCPD-Decision problem (i.e., Problem 1) is yes for $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}, B\big)$.

The SCPD-Decision problem is NP-hard as a generalization of the Temporal Knapsack decision problem. See Section S.1 of the supplementary material for a formal proof.

Let us now discuss a restriction of the solution space. Since items do not stack vertically, the space above the items on a shelf is wasted in any solution. This is formalized in dominance rule 1, which states that it is never useful to build shelves that have a height strictly greater than all the items it can contain.

Dominance rule 1. *There exists an optimal solution to the SCPD problem such that the height of each shelf built in the cabinet is equal to the height of an item.*

2.2 Related literature

With the SCPD problem now precisely defined, we review the related problems more thoroughly than in the introduction, and we also review the literature concerning arc-flow formulations.

2.2.1 Related problems

The SCPD problem is related to several well-studied problems: two-dimensional cutting and packing problems, temporal packing problems, and shelf design problems.

Regarding the design aspect of the SCPD problem, the closest problem is the 2-staged Two-Dimensional Knapsack (2TDK, Lodi and Monaci (2003)). In this latter problem, a two-dimensional rectangular box (cabinet in our problem) has to be cut into strips (shelves in our problem), which are themselves cut into pieces (compartments in our problem) via a series of at most two guillotine (i.e., edge-to-edge) cuts. A trimming cut, i.e., a supplementary cut, is used to separate pieces from waste. The problem is to cut out the subset of rectangular pieces that generates the most profit. Lodi and Monaci (2003) and Lodi et al. (2004) introduced the concept of *levels* to model the first-stage guillotine cuts. Specifically, decisions include which items *initialize* the levels and which items are assigned to each level. Items can only be assigned to a level if their height is less than or equal to the height of the item initializing the level. It is worth noting that the term "level" can be translated as "shelf" in the SCPD problem. The main difference between the 2TDK problem and the SCPD problem is that the former neither considers time nor a third dimension. It is straightforward to see that the SCPD problem is a generalization of the 2TDK problem: for a two-dimensional bin of size $(H_{\max}, W_{\max})$, consider a storage cabinet of height $H_{\max}$, width $W_{\max}$ and length 1, and for each item i with height $\tilde{h}_i$, width $\tilde{w}_i$ and profit $\tilde{p}_i$ of the 2TDK problem, create a corresponding item in the SCPD problem with height $\tilde{h}_i$, width $\tilde{w}_i$, length 1, profit $\tilde{p}_i$, $s_i = 0$, and $d_i = 1$.

The structure of the length dimension, which models a capacity constraint, and the presence of time intervals also closely link the SCPD problem with the temporal knapsack problem (TKP). The capacity constraint related to each compartment has to be respected over time while items enter and leave the compartment. Trivially, the SCPD problem generalizes the TKP: for each item with "weight" (length in our problem) $\tilde{\ell}$, time interval $[\tilde{s}, \tilde{s} + \tilde{d})$ and profit $\tilde{p}$, we can create a corresponding item in the SCPD problem with height $H_{\max}$, width $W_{\max}$, length $\tilde{\ell}$, time interval $[\tilde{s}, \tilde{s} + \tilde{d})$ and profit $\tilde{p}$. See Caprara et al. (2013), Caprara et al. (2016), Gschwind and Irnich (2017), Clautiaux et al. (2021) and Clausen et al. (2022) for recent works on the TKP.

To some extent, the SCPD problem shares similarities with *joint shelf design and shelf space allocation* (JSD-SSA) problems in retail stores studied in Hübner et al. (2021) and Gecili and Parikh (2022). The JSD-SSA problems aim to find a shelf design (number of shelves, their length, and their position) along with an allocation of three-dimensional items to these shelves (number of facings and their location) while respecting a minimum and maximum number of facings per item. The objective is to maximize the retailer's profit. In both the SCPD and JSD-SSA problems, the question is to design a storage space with shelves and to allocate products

to the shelves to maximize profit. However, several notable differences exist between these two problems. Gecili and Parikh (2022) considered shelf height and width as decision variables, and Hübner et al. (2021) decided the vertical positions on racks of shelf segments of different heights and widths, whereas, in the SCPD problem, the width of a shelf is fixed equal to the width of the cabinet. Once the design is decided, the remaining problem in Hübner et al. (2021) and Gecili and Parikh (2022) is a *shelf space allocation* (SSA) problem (see Bianchi-Aguiar et al. (2021) for a survey). Decisions for each product include their space assignment and related quantity, as well as their vertical and horizontal positioning on the shelves. Only identical items can be lined up one behind the other on a shelf, and vertical stacking is allowed. In the SCPD problem, we cannot stack items vertically, but we can assign different items to the same compartment. In SSA problems, the demand for a product depends on its available quantity and its position on the shelves (this is known as *cross-space elasticity*). In contrast, in the SCPD problem, the demand for each item is at most one, and the profit of an item is independent of its location within the cabinet, as the latter is not visible to customers. The JSD-SSA problems (like most SSA problems) are given an assortment, i.e., a list of items that requires space allocation on the shelves. In contrast, the SCPD problem also involves deciding which products to place on the shelves. One of the main features of the SCPD problem is the presence of items in the cabinet only over a time interval. In the SSA problems, the date an item is placed on a shelf and the date it is sold are not explicitly accounted for but dealt with implicitly through replenishment processes. For instance, in Hübner et al. (2021), a cost is due when replenishment from the back room is necessary due to high demand for a product. Replenishment is not considered in Gecili and Parikh (2022).

There also exists some connection with the work of Esmaili et al. (2018), who optimized the design of an automated drug dispensing cabinet. They considered matrix drawers and shelves with lanes (that we refer to as compartments) on each shelf to store the items. Similarly to the SCPD problem, a compartment can only store items with the same width as the compartment, and the goal is to maximize the *value* of the subset of items to be stored in the cabinet. This value corresponds to the effort saved in retrieving it from a more distant storage location. Contrary to our study, each item has a demand that is not necessarily unitary, and a compartment can only store units of the same item. Esmaili et al. (2018) also assumed non-increasing marginal values for storing additional units of an item. The shelf height is restricted to be a multiple of a given value corresponding to the minimum allowed shelf height. Another different feature of their problem, which has no equivalent in the SCPD problem, is that certain drugs cannot be stored next to each other.

2.2.2 Arc-flow formulations

The related problems mentioned in the previous section have mainly been addressed by integer linear programming (ILP) models, either by applying a general purpose solver directly to the model (Hübner et al. (2021), Gecili and Parikh (2022), Esmaili et al. (2018)), or through reformulations and decomposition methods (Caprara et al. (2013), Gschwind and Irnich (2017), Caprara et al. (2016), Clausen et al. (2022)). Heuristic algorithms have also been leveraged to tackle larger instances (Cardona and Gue, 2020; Gecili and Parikh, 2022).

Most extended formulations proposed in the literature on similar problems are arc-flow formulations. These formulations have become increasingly popular in the field of cutting and packing since their first successful use (Valério de Carvalho, 1999). In Macedo et al. (2010), the authors extended to the two-dimensional bin packing problem an arc-flow formulation originally developed for its one-dimensional version (Valério de Carvalho, 1999) and introduce reduction criteria to reduce the size of the graph on which an arc-flow formulation is constructed. Recently, Martinovic et al. (2023) introduced arc-flow formulations for temporal bin packing problems and showed that their direct solution with a commercial solver outperforms existing approaches. Arc-flow formulations provide good linear relaxations for combinatorial problems and can be solved directly by a general-purpose MILP solver. We refer to de Lima et al. (2022) for a thorough survey about arc-flow formulations where the networks are derived from state transition graphs associated with dynamic programs.

Martin et al. (1990) studied arc-flow formulations in decision hypergraphs. These latter are derived for problems formulated as dynamic programs in which decisions combine partial solutions (i.e., multiple states) into a single solution (i.e., a single state). Arenales and Morabito (1995) implicitly used such formulations to solve two-dimensional non-guillotine cutting problems, but the name *and-or graph* replaces the term hypergraph. They detail an algorithm for constructing an and-or graph by studying specific structures of the solutions to the studied problem, developing reduction methods to restrict the solution space, and a branch-and-bound algorithm to compute an optimal solution. Clautiaux et al. (2018) explicitly used an arc-flow formulation in a decision hypergraph to solve a two-dimensional four-stage guillotine cut bounded knapsack problem. They show how dynamic programming-based techniques such as Lagrangian filtering techniques and state-space relaxation designed for graphs can be extended to hypergraphs.

The size of arc-flow formulations is generally the bottleneck when solving them directly with a MILP solver. Several techniques have been used in the literature to address this issue. Christofides and Whitlock (1977) and Valério de Carvalho (1999) propose reduction criteria for one-dimensional and two-dimensional cutting and packing problems based on the ordering of the items and on the maximum number of times an item can be cut from a bin. Brandão and Pedroso (2016) proposed so-called *graph compression* techniques, which exploit the structure of the problem to reduce the number of arcs and vertices in the network. Similar ideas have been used in the field of *decision diagrams* for years (see e.g. Castro et al. (2022)). Clautiaux et al. (2021) solved the TKP exactly using an iterative refinement method called *successive sublimation dynamic programming* to keep graphs of tractable sizes. The problem is formulated as a dynamic program of exponential size, and relaxations of the problem are obtained by projecting the formulation into lower-dimensional state spaces whose size is increased until the resulting solution satisfies the constraints of the original problem.

3 A compact formulation of the SCPD problem

In this section, we formulate the SCPD problem as a compact MILP model. Similarly to what is done in Lodi and Monaci (2003), the creation of shelves and compartments is modeled using representative items. Indeed, the Dominance rule 1 makes it possible to restrict to shelves whose height is equal to that of one of the items, and, by definition, the width of a compartment is always the width of an item. To reduce the size of the formulation and break symmetries, we consider that the items are sorted by non-increasing height, i.e., $i < j$ implies $h_i \geq h_j$ (ties are broken arbitrarily). We also say that a time step t dominates another time step t' if $\{i \in \mathcal{I} : s_i \leq t < s_i + d_i\} \subset \{i \in \mathcal{I} : s_i \leq t' < s_i + d_i\}$. We define $\mathcal{T}^{\text{nd}}$ as the set of time steps not dominated by any other time step. For $i \in \mathcal{I}$, the binary variable z_i is equal to 1 if a shelf is represented by item i, meaning it has height h_i, 0 otherwise. Each binary variable $y_{i,j}$ is equal to 1 if a compartment on the shelf represented by i is represented by item j, meaning it has width w_j, 0 otherwise. Finally, each binary variable $x_{j,k}$ is equal to 1 if the item $k \in \mathcal{I}$ is assigned to the compartment represented by j, 0 otherwise. For consistency, if an item represents a shelf, it must also represent a compartment on this shelf, and if an item represents a compartment, it must be assigned to the compartment.

The MILP model is as follows:

$$\text{maximize} \quad \sum_{j \in \mathcal{I}} \sum_{\substack{k \in \mathcal{I}: \\ k \geq j, \\ w_k = w_j}} p_k x_{j,k} \tag{1a}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}} h_i z_i \leq H_{\max} \tag{1b}$$

$$\sum_{\substack{j \in \mathcal{I}: \\ j \geq i}} w_j y_{i,j} \leq W_{\max} \qquad i \in \mathcal{I} \tag{1c}$$

$$\sum_{\substack{k\in\mathcal{I}:\\ k\geq j,\\ w_k=w_j,\\ s_k\leq t<s_k+d_k}} \ell_k x_{j,k} \leq L_{\max} \qquad j\in\mathcal{I}, t\in\mathcal{T}^{\mathrm{nd}} \tag{1d}$$

$$\sum_{\substack{j\in\mathcal{I}:\\ j\leq k,\\ w_j=w_k}} x_{j,k} \leq 1 \qquad k\in\mathcal{I} \tag{1e}$$

$$x_{j,k} \leq x_{j,j} \qquad j\in\mathcal{I}, k\in\mathcal{I}, k>j, w_k=w_j \tag{1f}$$

$$x_{j,j} = \sum_{\substack{i\in\mathcal{I}:\\ i\leq j}} y_{i,j} \qquad j\in\mathcal{I} \tag{1g}$$

$$y_{i,j} \leq z_i \qquad i\in\mathcal{I}, j\in\mathcal{I}, j>i \tag{1h}$$

$$y_{i,i} = z_i \qquad i\in\mathcal{I} \tag{1i}$$

$$z_i \in \{0,1\} \qquad i\in\mathcal{I} \tag{1j}$$

$$y_{i,j} \in \{0,1\} \qquad i\in\mathcal{I}, j\in\mathcal{I}, j\geq i \tag{1k}$$

$$x_{j,k} \in \{0,1\} \qquad j\in\mathcal{I}, k\in\mathcal{I}, k\geq j, w_k=w_j \tag{1l}$$

The objective function in (1a) is the total profit of the selected items. Constraint (1b) ensures that the sum of the heights of the shelves does not exceed the height of the cabinet. Constraints (1c) ensure that the sum of the widths of the compartments on a shelf does not exceed the width of the cabinet. Constraints (1d) state that at any time step, the sum of the lengths of the items in a compartment at this time does not exceed the length of the cabinet. Adding this constraint is sufficient only for the time steps when new items enter the compartment. Constraints (1e) ensure that each item is assigned to no more than one compartment. Constraints (1f) and (1g) ensure that an item is assigned to a compartment only if that compartment is created and that the item representing that compartment is assigned to it. Constraints (1h) and (1i) ensure that a compartment is created on a shelf only if that shelf is created and that the item representing the shelf represents one of the compartments built on the shelf. Constraints (1j)-(1l) require the variables to be binary.

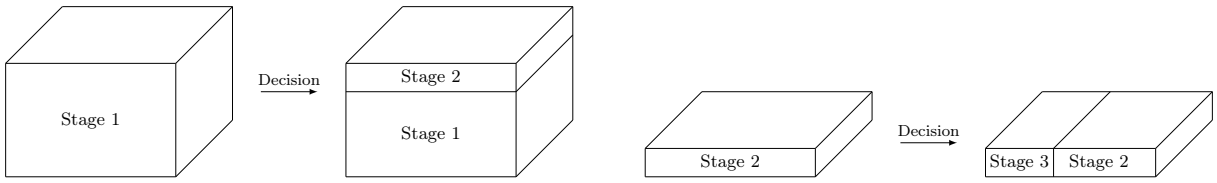
4 A hypergraph-based reformulation

This section proposes an arc-flow model for the SCPD problem based on its reformulation as a maximum cost flow problem with additional constraints. We use the following methodology. We formulate a relaxation of the SCPD problem as a dynamic program in Section 4.1. This dynamic program defines a decision hypergraph (Martin et al., 1990), which we use in Section 4.2 to derive an arc-flow formulation of the SCPD problem.

4.1 A dynamic program

We first propose a dynamic program to model the relaxation of the SCPD problem where each item can be assigned to more than one compartment. This corresponds to relaxing constraints (1e) in the MILP model (1).

We use a classical dynamic programming formalism in which states represent partial solutions, and decisions represent the possible extensions of a partial solution. Unlike in the classical case, where a decision from one state leads to another state, a decision here can lead to several states. In the problem we are modeling, there are several types of decisions: creating a shelf, creating a compartment, and selecting an item. We define three stages in our problem; each state belongs to one of them. A decision in a stage 1 state corresponds to building a shelf in the cabinet, leading to a stage 2 state and a residual stage 1 state. A decision in a stage 2 state corresponds to building a compartment on the shelf, leading to a stage 3 state and a residual stage 2 state. A decision in a stage 3 state corresponds to selecting an item to enter or leave the compartment, leading to another stage 3 state. Figure 3 shows decisions made from stage 1 and stage 2 states.



(a) Decision in a stage 1 state

(b) Decision in a stage 2 state

Figure 3: Illustration of the design decisions

We introduce our dynamic program bottom-up, from stage 3 to stage 1. We first describe the information needed to express a state for each stage. We then present a recursive formulation to compute the maximum profit that can be obtained from the current state by making decisions that lead to the dynamic program's initial states. We define $\mathcal{H} = \cup_{i\in\mathcal{I}}\{h_i\}$ as the set of possible shelf heights and, for each $h \in \mathcal{H}$, $\mathcal{W}_h = \{w : \exists i \in \mathcal{I}, h_i \leq h, w_i = w\}$ as the set of possible compartment widths. The union of each possible compartment width is noted $\mathcal{W} = \cup_{h\in\mathcal{H}}\mathcal{W}_h$.

In stage 3, each state is associated with a compartment of a given height $h \in \mathcal{H}$ and width $w \in \mathcal{W}_h$. Once the height and width of a compartment have been decided (we recall that all compartments have a length equal to $L_{\max}$), the problem restricted to this compartment is a TKP. Martinovic et al. (2023) and Clautiaux et al. (2021) presented two formulations of TKP as a dynamic program. The former leads to fewer states if no compression or reduction is applied. However, our recursive formulation for stage 3 is an adaptation of the latter, as it allows us to exploit more effectively the preprocessing and symmetry-breaking techniques described in Section 5.

The recursive formulation for stage 3 is an *event-based* formulation, where an event is defined by the beginning or the end of the time interval associated with an item. Specifically, we define two events for each item $i \in \mathcal{I}$, one when i can be selected at time step s_i to enter the compartment and one when i leaves the compartment at time step $s_i + d_i$ if present inside. Let $\mathcal{E} = (0, 1, \ldots, 2|\mathcal{I}|)$ be an ordered list of events where 0 represents a dummy event signaling the start of the time horizon. We define $\mathcal{E}^{\text{in}} \subset \mathcal{E}$ (resp. $\mathcal{E}^{\text{out}} \subset \mathcal{E}$) as the subset of events related to the start (resp. end) of a time interval associated with an item. For each event $e \in \mathcal{E}$, we denote by $i(e) \in \mathcal{I}$ the item to which the event relates and by $t(e)$ the time step at which it occurs, i.e., $t(e) = s_{i(e)}$ if $e \in \mathcal{E}^{\text{in}}$ and $t(e) = s_{i(e)} + d_{i(e)}$ if $e \in \mathcal{E}^{\text{out}}$. The events in $\mathcal{E}^{\text{in}} \cup \mathcal{E}^{\text{out}}$ are ordered from 1 to $2|\mathcal{I}|$ as follows: $e_1 < e_2$ if $t(e_1) < t(e_2)$ or $\big(t(e_1) = t(e_2) \wedge e_1 \in \mathcal{E}^{\text{out}} \wedge e_2 \in \mathcal{E}^{\text{in}}\big)$ (remaining ties are broken arbitrarily). For a compartment with height h and width w, we denote by $\mathcal{E}_{h,w} = \{e \in \mathcal{E} : h_{i(e)} \leq h, w_{i(e)} = w\}$ the events associated to items that can be assigned to it. We also note $E^*_{h,w}$ the last event of a compartment of height h and width w. For each event $e \in \mathcal{E}_{h,w}$, we denote by $E^-_{h,w}(e)$ the event preceding e in $\mathcal{E}_{h,w}$, i.e., $E^-_{h,w}(e) = \max\{\tilde{e} : \tilde{e} < e : w_{i(\tilde{e})} = w, h_{i(\tilde{e})} \leq h\}$.

The information needed to make decisions from a stage 3 state is the height $h \in \mathcal{H}$ of the compartment, its width $w \in \mathcal{W}_h$, its available length $L \in \{0, \ldots, L_{\max}\}$, the current event $e \in \mathcal{E}$, and the set of items currently in the compartment, represented as a binary vector $\mathbf{q}$ of size $|\mathcal{I}|$. A stage 3 state is thus denoted by $(h, w, L, e, \mathbf{q})$. In the remainder, $\alpha_3(h, w, L, e, \mathbf{q})$ is the maximum profit that can be obtained from a stage 3 state $(h, w, L, e, \mathbf{q})$ by making decisions that lead to the initial stage 3 state $(h, w, L_{\max}, 0, \mathbf{0})$. In other words, $\alpha_3(h, w, L, e, \mathbf{q})$ is the maximum profit that can be obtained by selecting items to be inserted in the compartment up to event e. Note that the profit associated with an item is only earned when it leaves the compartment. Denoting $\boldsymbol{\varepsilon_k}$ the characteristic vector of the subset $\{k\}$ of the underlying set $\mathcal{I}$, the forward recursion is as follows:

$$\alpha_3(h,w,L,e,\mathbf{q}) = \begin{cases} 0 & \text{if } e = 0 \\ \alpha_3(h,w,L,E^-_{h,w}(e),\mathbf{q}) & \text{if } e \in \mathcal{E}^{\text{in}} \wedge \mathbf{q}_{i(e)} = 0 \\ \alpha_3(h,w,L+\ell_{i(e)},E^-_{h,w}(e),\mathbf{q}-\boldsymbol{\varepsilon}_{\boldsymbol{i(e)}}) & \text{if } e \in \mathcal{E}^{\text{in}} \wedge \mathbf{q}_{i(e)} = 1 \wedge L+\ell_{i(e)} \leq L_{\max} \\ \max\left\{ \begin{array}{l} p_{i(e)} + \alpha_3(h,w,L-\ell_{i(e)},E^-_{h,w}(e),\mathbf{q}+\boldsymbol{\varepsilon}_{\boldsymbol{i(e)}}), \\ \alpha_3(h,w,L,E^-_{h,w}(e),\mathbf{q}) \end{array} \right\} & \text{if } e \in \mathcal{E}^{\text{out}} \wedge \mathbf{q}_{i(e)} = 0 \wedge L \geq \ell_{i(e)} \\ \alpha_3(h,w,L,E^-_{h,w}(e),\mathbf{q}) & \text{if } e \in \mathcal{E}^{\text{out}} \wedge \mathbf{q}_{i(e)} = 0 \wedge L < \ell_{i(e)} \end{cases}. \quad (2)$$

Five cases are considered in the above formula. The first case is the dummy event, and the profit is set at 0 because we are at the beginning of the time horizon. The second and third cases involve a state associated with an ingoing event e, and the decision to insert $i(e)$ or not insert this item into the compartment has already been made. The fourth and fifth cases involve a state associated with an outgoing event e, and the item $i(e)$ has already left the compartment if it was present.

In stage 2, only the height and width of the current shelf are needed to decide which compartments to build. A stage 2 state is thus denoted by (h, W), where $h \in \mathcal{H}$ and $W \in \{0, \ldots, W_{\max}\}$. We denote by $\alpha_2(h, W)$ the maximum profit that can be obtained from a stage 2 state (h, W) by recursively making decisions until all the remaining states are initial stage 3 states $(h, w, L_{\max}, 0, \mathbf{0})$ with $w \leq W$. In other words, $\alpha_2(h, W)$ is the maximum profit that can be obtained by building compartments on a shelf with height h, width W, and length $L_{\max}$. We set $\alpha_2(h, W) = 0$ if there is no item with height less than or equal to h or width less than or equal to W. The forward recursion is as follows:

$$\alpha_2(h,W) = \max_{w \in \mathcal{W}_h, w \leq W} \left\{ \alpha_3(h,w,L_{\max},E^*_{h,w},\mathbf{0}) + \alpha_2(h,W-w) \right\}. \quad (3)$$

In stage 1, each state only requires the height available in the cabinet for building shelves. A stage 1 state is denoted by (H), where $H \in \{0, \ldots, H_{\max}\}$. We denote by $\alpha_1(H)$ the maximum profit that can be obtained from a stage 1 state (H) by recursively making decisions until all the remaining states are initial stage 3 states $(h, w, L_{\max}, 0, \mathbf{0})$ with $h \leq H$ and $w \leq W_{\max}$. In other words, $\alpha_1(H)$ is the maximum profit that can be obtained by building shelves into a cabinet of height H, width $W_{\max}$, and length $L_{\max}$. We set $\alpha_1(H) = 0$ if there is no item of height less than or equal to H. The forward recursion is as follows:

$$\alpha_1(H) = \max_{h \in \mathcal{H}, h \leq H} \left\{ \alpha_2(h, W_{\max}) + \alpha_1(H-h) \right\}. \quad (4)$$

The optimal value of the relaxation of the SCPD problem defined at the beginning of this section is $\alpha_1(H_{\max})$. The time and space complexity of computing this value is exponential in the worst case because the maximum number of stage 3 states is exponential in the input size (the vector $\mathbf{q}$ can take at most $2^{|\mathcal{I}|}$ values). However, if the number of items of the same width that can be simultaneously present in a compartment is not large, the total number of states in the dynamic program remains tractable.

4.2 A hypergraph representation and an arc-flow formulation

We now introduce the so-called *decision hypergraph* associated with the dynamic program defined by the recursive formulae (2), (3), and (4). We use this hypergraph representation to build an arc-flow formulation of the SCPD problem. Using the vocabulary and notations of Martin et al. (1990), a decision hypergraph is the generalization of a directed acyclic graph. It consists of a set of vertices and a set of *decision hyperarcs* (hyperarcs in the remainder). In Martin et al. (1990) , a hyperarc is a triple $(\mathcal{F}, v, p)$, where $\mathcal{F}$ is a multiset of vertices called the *tail* of the hyperarc, v is a vertex called the *head* of the hyperarc, and p is a profit. To avoid unnecessarily heavy notations, we consider that $\mathcal{F}$ is a set of vertices instead of a multiset. In a hypergraph representation of a dynamic program, the vertices represent the states of the dynamic program, and the hyperarcs symbolize the decisions. Specifically, each hyperarc $(\mathcal{F}, v, p)$ represents the transition from the states at its tail to the state at its head.

For illustration purposes, the recursive formula (4) for a given $H \in \{0, \ldots, H_{\max}\}$ leads to a hyperarc with a zero profit for each $h \in \mathcal{H}$ such that $h \leq H$ where the two vertices at its tail represent the stage 2 state $(h, W_{\max})$ and the stage 1 state $(H - h)$, and the vertex at its head represents the stage 1 state (H). Similarly, the recursive formula (3) for given values of $h \in \mathcal{H}$ and $W \in \{0, \ldots, W_{\max}\}$ leads to a hyperarc with zero profit for each $w \in \mathcal{W}_h$ such that $w \leq W$ where the two vertices at its tail represent the stage 3 state $(h, w, L_{\max}, E^*_{h,w}, \mathbf{0})$ and the stage 2 state $(h, W - w)$, and the vertex at its head represents the stage 2 state (h, W). Finally, the recursive formula (2) gives rise to hyperarcs with a tail of cardinality one. Among these hyperarcs, the ones carrying the decision of an item leaving a compartment have a profit equal to the profit of the item. There is a unique vertex with no successor, called *sink*, which corresponds to the final state $(H_{\max})$ of the dynamic program. We assume that there also exists a vertex called *source* with no predecessor. For this purpose, each vertex associated with a stage 3 state $(h, w, L_{\max}, 0, \mathbf{0})$ is the head of a hyperarc with the source as the tail. The decision hypergraph is built recursively from the sink to the source. Algorithm 1 in Section S.2 of the supplementary material describes the creation procedure. It starts from the final state represented by the sink of the hypergraph and recursively adds vertices and hyperarcs using the recursive formulas (2)–(4). Note that we do not create vertices associated with stage 1 and stage 2 states from which no feasible decision can be taken (above, we have explicitly set the maximum profit associated with these states to 0).

Each feasible solution to the problem being modeled by the decision hypergraph corresponds to a flow (from the source to the sink) whose quantity arriving at the sink is one and whose value is the total cost of the flow. Such a flow in the decision hypergraph associated with the dynamic program defined by (2), (3), and (4) is not necessarily a feasible solution to the SCPD problem because it does not ensure that an item is assigned to no more than one compartment. We, therefore, formulate the SCPD problem as a maximum cost flow problem in the hypergraph but with additional linear constraints to ensure the feasibility of the solution represented by the flow.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be the decision hypergraph where $\mathcal{V}$ is the set of vertices, $\mathcal{A}$ the set of hyperarcs, v^+ its source and v^- its sink. We define $\mathcal{A}(i)$ as the set of hyperarcs associated with the exit of item i from a compartment (in which it was previously inserted). We denote by $\mathcal{A}^-(v)$ (resp. $\mathcal{A}^+(v)$) the set of hyperarcs of which $v \in \mathcal{V}$ is the head (resp. of which v belongs to the tail). With a slight abuse of notation, for each hyperarc $a \in \mathcal{A}$, we denote by p_a its profit. It is equal to p_i if $a \in \mathcal{A}(i)$ and to 0 otherwise. An arc-flow formulation of the SCPD problem is as follows:

$$\text{maximize} \quad \sum_{a \in \mathcal{A}} p_a x_a \tag{5a}$$

$$\text{subject to} \quad \sum_{a \in \mathcal{A}^-(v)} x_a - \sum_{a \in \mathcal{A}^+(v)} x_a = 0 \quad v \in \mathcal{V} \setminus \{v^+, v^-\} \tag{5b}$$

$$\sum_{a \in \mathcal{A}^-(v^-)} x_a = 1 \tag{5c}$$

$$\sum_{a \in \mathcal{A}(i)} x_a \leq 1 \qquad i \in \mathcal{I} \tag{5d}$$

$$x_a \in \mathbb{N} \qquad a \in \mathcal{A} \tag{5e}$$

The objective function in (5a) is the total profit obtained by inserting items into the cabinet. Constraints (5b) model flow conservation, while constraint (5c) imposes that the quantity of flow received at the sink is one (there is a single cabinet in the problem). Constraints (5d) ensure that each item is assigned at most once in the cabinet. Constraints (5e) require that the arc variables are integer. Note that the flow of a hyperarc can be greater than one, as shelves and compartments of the same size can be built into the cabinet (but different items must be selected in each).

Remark 1. *A solution to the arc-flow formulation* (5) *is converted into a solution to the SCPD problem by iterating over the hyperarcs with non-zero flow value starting from the sink and tracking the assignment*

of compartments to shelves and items to compartments (see Algorithm 5 in Section S.3 of the supplementary material). To handle the case where a flow greater than one exits a vertex, we store pairs consisting of its associated state and a distinct shelf or compartment. Without loss of generality, we assume that the shelves and compartments built with our procedure are placed in the cabinet from top to bottom and from left to right, respectively.

Table 2 describes an instance of the SCPD problem, and Figure 4 shows the decision hypergraph associated with this instance, and, in blue, a feasible solution to the problem, i.e., a flow from the source to the sink. We recall that the hypergraph is built from the sink v^-, and the decision to select an item to leave a compartment is taken before the decision to select the item to enter into it. To avoid overloading the figure, if the profit associated with a hyperarc is 0, it is not explicitly indicated. The same applies to a flow with a value of 0. Figure 5 illustrates the storage cabinet physical design system associated with the solution of Figure 4. From the initial state (10) corresponding to the whole cabinet, two decisions are possible: creating a compartment of height 5 (black hyperarc going to (10), or creating a compartment of height 9 (blue hyperarc). In the solution in blue, a shelf of height 9 is created, leading to state $(9, 5)$ and residual state (1), which is discarded since it is too small to pack any item. From state $(9, 5)$, a first compartment of width 2 is created, producing a residual space of size $(9, 3)$. State $(9, 3)$ also produces a compartment of size 2 and a remaining space that is too small to pack any item (state $(9, 1)$). State $(9, 2, 4, 4, (0, 0))$ is thus reached twice and leads to two decisions: selecting item 2 (upper blue hyperarc) and not selecting item 2 (lower blue hyperarc). The latter decision leads to state $(9, 2, 4, 3, (0, 0))$, from which item 1 is selected. In each compartment created, the remaining decision is to remove the item when the corresponding event is reached.

$H_{\max}$	$W_{\max}$	$L_{\max}$
10	5	4

i	h_i	w_i	ℓ_i	p_i	s_i	d_i
1	5	2	2	1	0	2
2	9	2	1	2	1	2

Table 2: An SCPD instance with two items

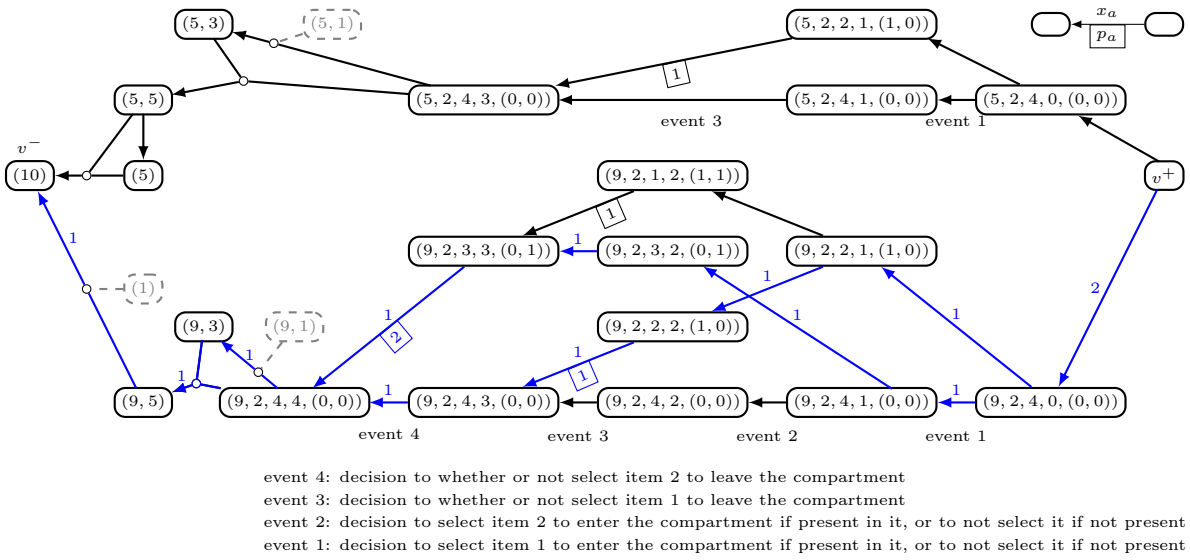
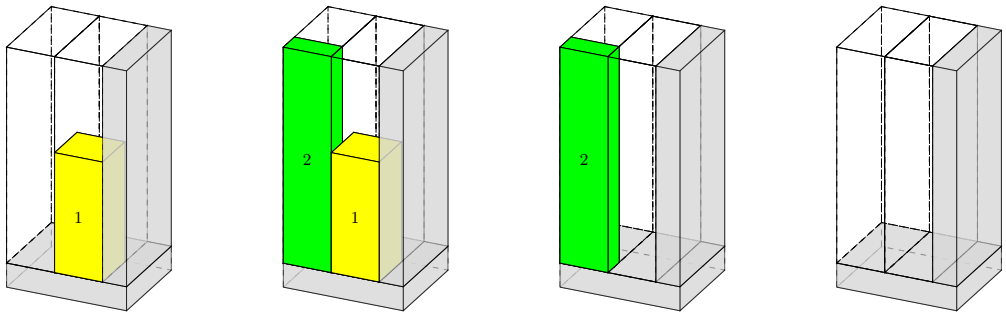


Figure 4: Decision hypergraph of the instance given in Table 2 and an optimal solution



(a) After event 1 (b) After event 2 (c) After event 3 (d) After event 4

Figure 5: Solution associated with the flow in blue in Figure 2

5 Improvements of the hypergraph-based formulation

This section describes improvements on the arc-flow formulation (5). To strengthen the quality of its linear relaxation, we introduce valid inequalities in Section 5.1. Since the formulation is non-polynomial in the size of the instance, we describe in Section 5.2 to Section 5.4 symmetry breaking and graph compression techniques to reduce the number of hyperarcs in $\mathcal{A}$. The modifications of Algorithm 1 to take into account all these latter improvements during the hypergraph generation procedure are given in Section S.4 of the supplementary material.

We first define two optimization problems that some of our techniques have to solve and describe the notation that will be used throughout this section.

Problem 3 (Bin Packing **(BP)**). *Suppose an infinite number of bins with capacity $C \in \mathbb{N}$ are given and let $\mathcal{J}$ be a set of* items. *Each item $j \in \mathcal{J}$ has a weight $\overline{w}_j \in \mathbb{N}$. The BP problem is to find an assignment of the items to the bins such that (i) the sum of the weights of the items assigned to each bin is less than or equal to C, (ii) the number of bins to which at least one item is assigned is minimized.*

Problem 4 (Temporal Bin Packing **(TBP)**). *Suppose an infinite number of bins with capacity $C \in \mathbb{N}$ and let $\mathcal{J}$ be a set of* items. *Each item $j \in \mathcal{J}$ has a weight $\overline{w}_j \in \mathbb{N}$ and a time interval $[\overline{s}_j, \overline{s}_j + \overline{d}_j)$ with $\overline{s}_j \in \mathbb{N}$ and $\overline{d}_j \in \mathbb{N}$, during which the item is present. The TBP problem is to find an assignment of the items to the bins such that (i) for each bin, the sum of the weights of the items simultaneously present in it is less than or equal to C, (ii) the number of bins to which at least one item is assigned is minimized.*

Let $z^*_{\mathrm{BP}(\mathcal{J},C)}$ (resp. $z^*_{\mathrm{TBP}(\mathcal{J},C)}$) denote the value of an optimal solution to the BP (resp. TBP) problem with capacity C and set of items $\mathcal{J}$. Let also $\mathcal{I}_{\leq h} = \{i \in \mathcal{I} : h_i \leq h\}$ and $\mathcal{I}_{\leq h,w} = \{i \in \mathcal{I} : h_i \leq h, w_i = w\}$ denote the set of items with a height less than or equal to h and the set of items with a height less than or equal to h and a width equal to w, respectively. Given a set of items $\mathcal{J} \subseteq \mathcal{I}$, we define $h_{\max}(\mathcal{J}) = \max_{j\in\mathcal{J}}\{h_j\}$. In the following, we abuse the notation $z^*_{\mathrm{TBP}(\mathcal{J},C)}$ by denoting as $z^*_{\mathrm{TBP}(\mathcal{I}_{\leq h,w},L_{\max})}$ the value of an optimal solution to the TBP problem with capacity $L_{\max}$ where the items are defined as follows: for each $i \in \mathcal{I}_{\leq h,w}$, create an item with weight ℓ_i and time interval $[s_i, s_i + d_i)$.

5.1 A family of valid inequalities

Successful applications of arc-flow models depend on the quality of the formulation obtained. In these cases, the value obtained by solving their linear relaxation is generally close to the optimum. For hypergraph-based arc-flow formulations, some structures may weaken the quality of the relaxation. We illustrate one such structure below and propose a set of valid inequalities to eliminate it.

Consider an instance with a cabinet of size $(2, 4, 1)$ and a single item of size $(2, 1, 1)$, and a solution for this instance expressed as a flow in a hypergraph, depicted in Figure 6. In this solution, 0.25 units of flow pass

through the leftmost hyperarc, corresponding to creating a shelf with height 2. Then, four compartments with identical widths equal to 1 are built on this shelf, and, due to flow conservation, 0.25 units of flow pass through the corresponding rightmost hyperarcs. This allows to obtain one unit of flow on the arc going out of the vertex associated with the stage 3 state. Although only "a quarter of a shelf" has been built, the item in our instance can be "entirely" selected in the compartment built on this shelf. In a solution of the linear relaxation to (5), fractional units of flow are successively combined to form a whole unit of flow, leaving vertices associated with stage 3 states, increasing the number of items selected and thus the total profit.

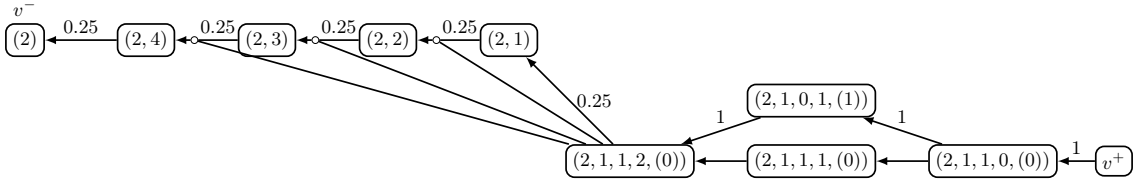


Figure 6: Example of a solution to the linear relaxation of model (5)

The following inequalities prevent an item from being selected if no shelf has been built to hold it. We note $\tilde{\mathcal{A}}(h)$ the set of hyperarcs associated with the design of a shelf of height $h \in \mathcal{H}$ in the cabinet. These hyperarcs are those whose head is a vertex associated with a stage 1 state and whose tail includes a vertex associated with a stage 2 state $(h, W_{\max})$.

Proposition 1. *For each item $i \in \mathcal{I}$, the following inequality is valid for model* (5)*:*

$$\sum_{a \in \mathcal{A}(i)} x_a \leq \sum_{h \in \mathcal{H}, h \geq h_i} \sum_{a \in \tilde{\mathcal{A}}(h)} x_a. \tag{6}$$

Proof. First, note that in the model (5), we have $\sum_{a \in \mathcal{A}(i)} x_a \leq 1$. In an integer solution, if $\sum_{a \in \mathcal{A}(i)} x_a = 0$, the inequality (6) holds due to the non-negativity of the variables. If $\sum_{a \in \mathcal{A}(i)} x_a = 1$ (i.e., item i is selected), then there must be a hyperarc $\bar{a} \in \tilde{\mathcal{A}}(h)$ with $h \geq h_i$ such that $x_{\bar{a}}$ is equal to 1 (i.e., a shelf with height greater than h_i should be built) . Therefore, the inequality (6) holds in this case too. □

The number of inequalities (6) is equal to the number of items in the instance. When we say later that we use these inequalities, this means that we add them all in the formulation (5). Note that the solution displayed in Figure 6 is eliminated because only 0.25 units of flow traverse the single hyperarc associated with the design of a shelf with a height greater than or equal to the height of the item.

5.2 Pruning hyperarcs using dominance rules

In this section, we introduce dominance rules (see, e.g., Jouglet and Carlier (2011)), which we use to remove hyperarcs associated with design decisions (i.e., building shelves and compartments in the cabinet). The dominance rules proposed in this section are mainly inspired by the literature on one-dimensional and two-dimensional cutting and packing problems (Christofides and Whitlock (1977), Valério de Carvalho (1999)). We use the following methodology. We first state a dominance rule, i.e., a property that is satisfied by at least one optimal solution to the SCPD problem. Then, we describe how we modify the hypergraph creation to accommodate it.

We first introduce Dominance rules 2 and 3 for the design of shelves within the cabinet.

Dominance rule 2. *There exists an optimal solution to the SCPD problem such that the shelves are ordered from top to down by non-decreasing height.*

Dominance rule 2 is straightforward to exploit in a constructive approach: only shelves whose height is less than that of the shelves present in the partial solution can be built subsequently. In the hypergraph, a vertex

associated with a given stage 1 state can be reached from the sink by different sequences of decisions (i.e., the vertex belongs to the tail of several hyperarcs), and thus, the sets of shelves built in each of the corresponding partial solutions are different. This means that we cannot force all solutions of the arc-flow formulation to satisfy Dominance rule 2. However, we partially enforce it by recording additional information at the vertices associated with stage 1 states and building their incoming hyperarcs, considering these vertices in non-increasing order of height. To each vertex $v \in \mathcal{V}$ associated with a stage 1 state, we attach an additional value $c_1(v,h) \in \mathbb{N}$ for each height $h \in \mathcal{H}$ which is the minimum number of times a hyperarc corresponding to the creation of a shelf of height h must be traversed to reach the sink v^-. Denoting by $h(v)$ the height of the stage 1 state represented by v, this value can be computed recursively as follows:

$$c_1(v^-,h) = 0 \qquad\qquad h \in \mathcal{H},$$
$$c_1(v,h) = \min_{u \in \mathcal{A}^+(v)} \{c_1(u,h) + [h(u) - h(v) = h]\} \qquad\qquad v \neq v^-, h \in \mathcal{H}.$$

If $c_1(v,h) \geq 1$, then at least one hyperarc associated with the creation of a shelf of height h belongs to any path from v to v^-, and therefore hyperarcs associated with the creation of shelves of height larger than h and incoming at v can be removed, or, better still, not built during the hypergraph creation. Figure 7 illustrates the dominance rule. Let us assume that state $v = (6)$ is such that $c_1(v,2) = 1$ and that the dashed hyperarc incoming at v is related to the creation of a shelf with height 4. If this hyperarc has a non-zero flow value in a solution of the formulation (5), then the shelves are not ordered from top to bottom by non-decreasing height in the solution to the SCPD problem obtained by the conversion procedure described in Remark 1.

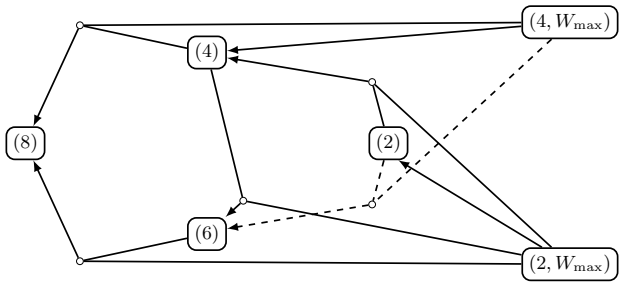


Figure 7: Illustration of how to exploit Dominance rule 2 in the hypergraph generation

The additional values $c_1(v,h)$ can also be used to enforce another dominance rule. Let $N_1(\mathcal{J})$ be an upper bound on the number of shelves (of any height) to build if one selects all items in $\mathcal{J} \subseteq \mathcal{I}$. For each height h, the smallest possible value for $N_1(\mathcal{I}_{\leq h})$ is $z^*_{\text{BP}(\mathcal{L},L_{\max})}$ where $\mathcal{L}$ contains $z^*_{\text{TBP}(\mathcal{I}_{\leq h,w},L_{\max})}$ items with weight w for each width $w \in \mathcal{W}_h$. The validity of this procedure is based on the fact that items assigned to the same compartment share the same width. Minimizing the number of compartments to be built can be done width by width. Once the minimum number of compartments for each width is known, minimizing the number of "bins" also minimizes the number of shelves to be built. Solving the BP problem and the TBP problem to optimality is not required to compute a valid upper bound using this procedure.

Dominance rule 3. *Let $\mathcal{I}_{h,w} = \{i \in \mathcal{I} : h_i = h, w_i = w\}$ be the set of items with height equal to h and width equal to w. Let $h \in \mathcal{H}$ and $U_1(h) = \min\left\{\left\lfloor \frac{H_{\max}}{h} \right\rfloor, N_1(\mathcal{I}_{\leq h}), \sum_{w \in \mathcal{W}_h} \min\left\{z^*_{TBP(\mathcal{I}_{\leq h,w},L_{\max})}, |\mathcal{I}_{h,w}|\right\}\right\}$. There exists an optimal solution to the SCPD problem where the number of shelves with height h is not greater than $U_1(h)$.*

Proof. The first term of the expression comes from the fact that a cabinet of height $H_{\max}$ cannot contain more than $\lfloor H_{\max}/h \rfloor$ shelves of height h.

Let S^* be an optimal solution to the SCPD problem such that there are $N_1(\mathcal{I}_{\leq h}) + k$ shelves of height h with $k \in \mathbb{N}_{>0}$. Since the items in $\mathcal{I}_{\leq h}$ can all fit into $N_1(\mathcal{I}_{\leq h})$ shelves, we can obtain a solution with the same

value by removing the $N_1(\mathcal{I}_{\leq h}) + k$ shelves, and reassigning their contents to $N_1(\mathcal{I}_{\leq h})$ shelves. This is always possible by the definition of $N_1(\mathcal{I}_{\leq h})$.

The validity of the third term follows from a similar argument. Consider a feasible solution S to the SCPD problem with more than $\sum_{w \in \mathcal{W}_h} \min\{z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}, |\mathcal{I}_{h,w}|\}$ shelves of height h. Now consider a solution $\tilde{S}$ obtained from S by removing all shelves of height h, and creating new shelves as follows. For each width $w \in \mathcal{W}_h$, group all items of the set $\mathcal{I}_{\leq h,w}$ selected in S into $z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}$, compartments and create one shelf per compartment. The height of each shelf in $\tilde{S}$ is the maximum height of an item in its compartment. Therefore, there cannot be more than $|\mathcal{I}_{h,w}|$ shelves containing an item of height h. The new solution $\tilde{S}$ contains for each w at most $z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}$ shelves among which at most $|\mathcal{I}_{h,w}|$ have height h. The solution $\tilde{S}$ has at most $\sum_{w \in \mathcal{W}_h} \min\{z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}, |\mathcal{I}_{h,w}|\}$ shelves of height h. □

We use Dominance rule 3 to remove hyperarcs that would create shelves unused in at least one optimal solution. When extending a stage 1 state $v = (H)$ during the creation of the hypergraph, the decision to design a shelf with height $h \in \mathcal{H}$ is skipped if $c_1(v, h) = U_1(h)$.

We also introduce Dominance rules 4 and 5 for the design of compartments on a shelf.

Dominance rule 4. *There exists an optimal solution to the SCPD problem such that the compartments in each shelf are ordered from left to right by non-decreasing height of their tallest item (i.e., the item with the greatest height).*

During the creation of the hypergraph, the assignment of the items to the compartments is unknown. Therefore, we cannot strictly enforce Dominance rule 4. Instead, we order the compartments by non-decreasing height of the tallest item they can contain. To enforce this, we modify the height in the two states generated when making a compartment design decision at a stage 2 state. Specifically, creating a compartment of width $w \in \mathcal{W}_h$ from a stage 2 state (h, W) leads to the creation of a stage 3 state $(h_{\max}(\mathcal{I}_{\leq h,w}), w, L_{\max}, E^*_{h,w}, \mathbf{0})$ and a stage 2 state $(h_{\max}(\mathcal{I}_{\leq h,w}), W - w)$.

Dominance rule 5 extends Dominance rule 3 to compartments built on a shelf. Let $N_2(\mathcal{J})$ be an upper bound on the minimum number of compartments to build if one selects all items in $\mathcal{J} \subseteq \mathcal{I}$. For each height $h \in \mathcal{H}$ and width $w \in \mathcal{W}_h$, the smallest possible value for $N_2(\mathcal{I}_{\leq h,w})$ is equal to $z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}$.

Dominance rule 5. *Let $h \in \mathcal{H}$, $w \in \mathcal{W}_h$ and $U_2(h, w) = \min\left\{\left\lfloor \frac{W_{\max}}{w} \right\rfloor, N_2(\mathcal{I}_{\leq h,w})\right\}$. There exists an optimal solution to the SCPD problem where on a shelf of height h, the number of compartments of width w is less than or equal to $U_2(h, w)$.*

Proof. Each shelf in a feasible solution of the SCPD problem, and hence in any optimal solution, cannot have more compartments of width w than the number of compartments that can fit on the shelf, hence $U_2(h, w) \leq \left\lfloor \frac{W_{\max}}{w} \right\rfloor$. If, in a solution, a shelf of height h contains more than $N_2(\mathcal{I}_{\leq h,w})$ compartments of width w, then it is possible to remove these compartments and replace them with at most $N_2(\mathcal{I}_{\leq h,w})$ compartments of width w with the same set of items. □

To enforce Dominance rule 5, we attach an additional value $c_2(v, w) \in \mathbb{N}$ to each vertex v for each width w, which is the minimum number of times a hyperarc corresponding to the creation of a compartment of width w must be traversed to reach the sink v^-. Denoting $w(v)$ the width of a stage 2 state represented by v, this value can be computed recursively as follows:

$$
\begin{aligned}
&c_2(v^-, w) = 0 && w \in \mathcal{W}, \\
&c_2(v, w) = \min_{u \in A^+(v)} \{c_2(u, w) + [w(u) - w(v) = w]\} && v \neq v^-, w \in \mathcal{W}.
\end{aligned}
$$

When extending a stage 2 state $v = (h, W)$ during the creation of the hypergraph, the decision to design a compartment of width $w \in \mathcal{W}_h$ is skipped if $c_2(v, w) = U_2(h, w)$.

In Dominance rule 6, we exploit Dominance rule 2 to formalize the fact that it is never useful to create a shelf whose size is greater than the tallest object it can hold.

Dominance rule 6. *There exists an optimal solution to the SCPD problem such that, for each shelf of height $h \in \mathcal{H}$, the leftmost compartment has a width such that there is an item of this width with a height equal to h.*

Proof. Let S^* be an optimal solution to the SCPD problem and let ψ be a shelf in S^*. The shelf ψ can be transformed in two steps into a shelf where the leftmost compartment contains an item with a height equal to h_ψ. Since the compartments in ψ can be permuted without changing the feasibility or the optimality of S^*, a compartment that can contain an item with the largest height $\tilde{h} \leq h$ can be permuted with the leftmost compartment in ψ. If $\tilde{h} < h$, then the height of shelf ψ can be reduced to $\tilde{h}$. □

Considering Dominance rule 6 in the hypergraph generation is straightforward since it means building only compartments of width in the set $\{w \in \mathcal{W}_h : \exists i \in \mathcal{I}_{\leq h,w}, h_i = h\}$ instead of in the set $\{w \in \mathcal{W}_h : \exists i \in \mathcal{I}_{\leq h,w}, h_i \leq h\}$ when a stage 2 state $v = (h, W_{\max})$ is considered.

5.3 Merging equivalent states

To further reduce the size of the hypergraph, we now aim at merging states that represent equivalent subproblems, i.e., the possible item selection decisions from each of these states to the source v^+ of the hypergraph are identical. For a given state v, let $\eta(v) \subseteq \mathbb{P}(\mathcal{I})$ be the set of partial solutions, in terms of items, that can be derived from v, i.e., the set of possible subsets of items selected in a solution to the subproblem represented by v. Note that this set is independent of the structure of the solution and the order in which the different items are selected. Definition 1 specifies what we call equivalent states.

Definition 1. *(Equivalent states) We say that two states (vertices) v_1 and v_2 of the decision hypergraph $(\mathcal{V}, \mathcal{A})$ are equivalent if $\eta(v_1) = \eta(v_2)$.*

The procedures we implement to reduce the hypergraph are inspired by those of Brandão and Pedroso (2016) and Bergman et al. (2016), initially introduced for simple graphs. Merging two states corresponds to removing the two corresponding vertices and creating a new vertex whose set of ingoing (resp. outgoing) hyperarcs is the union of the hyperarcs ingoing to (resp. outgoing from) the two original vertices. Proposition 2 indicates that merging vertices associated with equivalent states does not change the optimal value of the arc-flow formulation (5).

Proposition 2. *Let v_1 and v_2 be two vertices of the decision hypergraph $(\mathcal{V}, \mathcal{A})$. If $\eta(v_1) = \eta(v_2)$, then merging the states associated with v_1 and v_2 in $(\mathcal{V}, \mathcal{A})$ does not modify the optimal value of the arc-flow formulation* (5).

Proof. Any feasible flow obtained by solving formulation (5) considering the original decision hypergraph can be converted into a feasible flow in the modified hypergraph by simply replacing the two vertices v_1 and v_2 by the new vertex corresponding to the state resulting from the merge of the two states. Since $\eta(v_1) = \eta(v_2)$, no additional solutions in terms of item selection are added by this procedure. □

The following three propositions introduce sufficient conditions for states to be equivalent. We now introduce additional notations. We call a height $H \in \{0, \ldots, H_{\max}\}$ (resp. width $W \in \{0, \ldots, W_{\max}\}$) *reachable* if there exists a subset of items such that the sum of their heights (resp. widths) equals H (resp. W). We define the sets $\mathcal{RH}_H = \left\{\tilde{H} : \exists \mathcal{J} \subseteq \mathcal{I}_{\leq H}, \sum_{j \in \mathcal{J}} h_j = \tilde{H}\right\}$ and $\mathcal{RW}_H = \left\{\tilde{W} : \exists \mathcal{J} \subseteq \mathcal{I}_{\leq H}, \sum_{j \in \mathcal{J}} w_j = \tilde{W}\right\}$ as the sets of reachable heights and reachable widths considering only items with height less than or equal to $H \in \{0, \ldots, H_{\max}\}$.

Proposition 3 is inspired by a classical technique used in packing problems and is related to the notion of *raster points* (Scheithauer, 2018).

Proposition 3. *Let $H_1, H_2 \in \{0, \ldots, H_{\max}\}$. The stage 1 states $v_1 = (H_1)$ and $v_2 = (H_2)$ of the decision hypergraph are equivalent if*

$$\max_{\tilde{H} \in \mathcal{RH}_{H_1}:\ \tilde{H} \leq H_1} \{\tilde{H}\} = \max_{\tilde{H} \in \mathcal{RH}_{H_2}:\ \tilde{H} \leq H_2} \{\tilde{H}\}.$$

Proof. Let $H = \max_{\tilde{H} \in \mathcal{RH}_{H_1}:\ \tilde{H} \leq H_1} \{\tilde{H}\} = \max_{\tilde{H} \in \mathcal{RH}_{H_2}:\ \tilde{H} \leq H_2} \{\tilde{H}\}$. Since the largest possible combination of item heights is equal to $H \leq \min\{H_1, H_2\}$, the same subset of shelves can be built inside a cabinet of height H_1 or H_2. This translates into $\eta(v_1) = \eta(v_2)$. □

During the creation of the hypergraph, the two equivalent states $v_1 = (H_1)$ and $v_2 = (H_2)$ are merged by changing their height to the same value $\hat{H} = \max\{\tilde{H} \in \mathcal{RH}_{H_1} :\ \tilde{H} \leq H_1\}$. Note that computing $\hat{H}$ involves solving a subset sum problem, known to be weakly NP-hard. However, considering that the target instances of the SCPD problem have hundreds of items, computing this value is never a bottleneck. Furthermore, in the specific case of stage 1 states, Proposition 3 can be improved by combining it with Dominance rule 2. When creating a shelf of height h in a cabinet of height H, instead of modifying the height of the remaining stage 1 state to $\max\{\tilde{H} \in \mathcal{RH}_H :\ \tilde{H} \leq H\}$, we change it to $\max\{\tilde{H} \in \mathcal{RH}_h :\ \tilde{H} \leq H\}$.

Similar ideas for widths rather than heights lead to Proposition 4.

Proposition 4. *Let $h \in \mathcal{H}$ and $W_1, W_2 \in \{0, \ldots, W_{\max}\}$. The stage 2 states $v_1 = (h, W_1)$ and $v_2 = (h, W_2)$ are equivalent if*

$$\max_{\tilde{W} \in \mathcal{RW}_h:\ \tilde{W} \leq W_1} \{\tilde{W}\} = \max_{\tilde{W} \in \mathcal{RW}_h:\ \tilde{W} \leq W_2} \{\tilde{W}\}.$$

Proof. Let $W = \max_{\tilde{W} \in \mathcal{RW}_h:\ \tilde{W} \leq W_1} \{\tilde{W}\} = \max_{\tilde{W} \in \mathcal{RW}_h:\ \tilde{W} \leq W_2} \{\tilde{W}\}$. Since the largest possible combination of item widths is equal to $W \leq \min\{W_1, W_2\}$, the same subset of compartments can be built inside a cabinet of height h and width W_1 or W_2. This translates into $\eta(v_1) = \eta(v_2)$. □

During the creation of the hypergraph, the two equivalent states $v_1 = (h, W_1)$ and $v_2 = (h, W_2)$ are merged by changing their width to the same value $\hat{W} = \max\{\tilde{W} \in \mathcal{RW}_h :\ \tilde{W} \leq W_1\}$. Computing this value is as difficult as computing $\hat{H}$, but the same observation holds.

Figure 8 illustrates how we merge vertices of the hypergraph when they represent equivalent stage 2 states. In this example, the items have heights in the set $\{3, 6, 7\}$, and the cabinet has a height of 10. Building a shelf of height 6 leads to a remaining stage 1 state with height 4, and building a shelf with height 7 leads to a remaining stage 1 state with height 3. The two stage 1 states (4) and (3) are equivalent because only items of height less than or equal to 3 can fit into the remaining cabinet.

(a) Without merging

(b) With merging

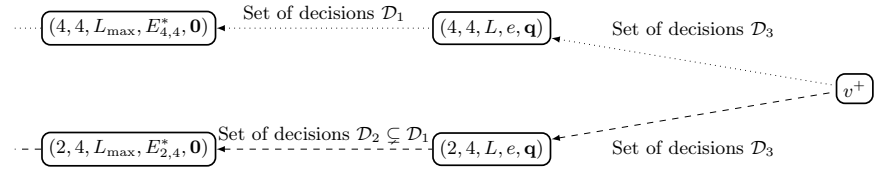
Figure 8: Example of equivalent stage 2 states

Proposition 5 gives a sufficient condition for the equivalence of stage 3 states.

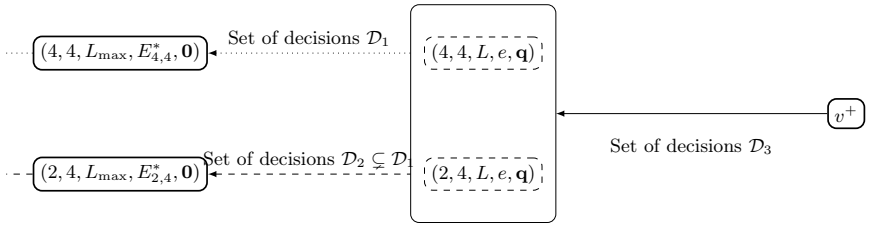
Proposition 5. *Let $h_1, h_2 \in \mathcal{H}$, $w \in \mathcal{W}$, $L \in \{0, \ldots, L_{\max}\}$, $e_1, e_2 \in \mathcal{E}$, and $\mathbf{q} \in \{0,1\}^{|\mathcal{I}|}$. The two stage 3 states $v_1 = (h_1, w, L, e_1, \mathbf{q})$ and $v_2 = (h_2, w, L, e_2, \mathbf{q})$ are equivalent if $\{\tilde{e} \in \mathcal{E}_{h_1,w} : \tilde{e} \leq e_1\} = \{\tilde{e} \in \mathcal{E}_{h_2,w} : \tilde{e} \leq e_2\}$.*

Proof. Note that v_1 and v_2 are both associated with a compartment of width w, which currently contains the subset of selected items $\mathbf{q}$. The condition ensures that, when considering the events in descending order, the sets of items that can enter or leave the compartment are the same from e_1 as from e_2. Therefore, $\eta(v_1) = \eta(v_2)$. □

We use Proposition 5 to merge vertices associated with equivalent stage 3 states. For each event $e \in \mathcal{E}$, we compute $H^-(e,h) = \max\left\{h_{i(\tilde{e})} : \tilde{e} \in \mathcal{E}, \tilde{e} \leq e, h_{i(\tilde{e})} \leq h, w_{i(\tilde{e})} = w_{i(e)}\right\}$ as the maximum height of items in the events occurring before e that share their width with $i(e)$, considering only items with a height less than or equal to h. After making the item selection decision associated with event e, we change the height of the resulting stage 3 states to $H^-(E^-_{h,w}(e),h)$, which in a sense corresponds to reducing the height of the compartment. Figure 9 illustrates how we merge vertices of the hypergraph when they represent equivalent stage 3 states. Note that this procedure reduces the number of vertices and hyperarcs in the decision hypergraph.



(a) Without merging vertices associated with equivalent stage 3 states, the dashed and dotted paths are distinct



(b) After merging vertices associated with equivalent stage 3 states, the two paths merge as soon as the remaining decisions are equal

Figure 9: Example of equivalent stage 3 states

5.4 Exploiting trivial subproblems

In the SCPD problem, some states represent *trivial subproblems*, i.e., all the admissible items can be selected by making decisions from these states to the source of the hypergraph. For a stage 1 state (H), an admissible item is an item that fits into a cabinet with height H, width $W_{\max}$, and length $L_{\max}$. For a stage 2 state (h,W), an admissible item is an item that fits on a shelf with height h, width W, and length $L_{\max}$. For a stage 3 state $(h,w,L,e,\mathbf{q})$, an admissible item is an item that leaves a compartment with height h, width w, and length equal to L before or at event e. We refer to a state associated with a trivial subproblem as a *trivial state*. We first clarify this notion in Definition 2, then we show how it can be leveraged to simplify our dynamic program and thus reduce the size of the decision hypergraph.

Definition 2. *A state v of the dynamic program (2)–(4) is said to be trivial if*

- *it is a stage 1 state (H) and $\eta(v) = \mathbb{P}(\mathcal{I}_{\leq H})$.*
- *it is a stage 2 state (h,W) and $\eta(v) = \mathbb{P}\left(\bigcup_{w \leq W} \mathcal{I}_{\leq h,w}\right)$.*
- *it is a stage 3 state $(h,w,L,e,\mathbf{q})$ and $\eta(v) = \mathbb{P}\left(\{i \in \mathcal{I}_{\leq h,w} : \exists \tilde{e} \in \mathcal{E}^{out}, \tilde{e} \leq e, i(\tilde{e}) = i\}\right)$.*

Nothing prevents an item from being selected more than once in the dynamic program (2)–(4). Selecting an item at most once is only enforced by constraints (5d) in the arc-flow formulation. This prevents us from creating directly an arc from the source to the vertex corresponding to a trivial state, with a profit equal to the sum of the profits of the admissible items. Since each item in an instance of the SCPD problem cannot be selected twice, even if taking an item is locally obviously profitable, we have to consider the possibility of not

selecting this item. When a trivial state is detected, we use a simplified recursion: we only need to decide which items to select and sum their respective profits. We introduce two recursions: one used when we detect a trivial stage 1 or stage 2 state, and another used when we detect a trivial stage 3 state.

Before defining our simplified states, let us define additional notation to simplify the formulas. We denote by I_w the list of items of $\mathcal{I}$ with width w ordered by non-decreasing value of their height. We also denote by $\mathrm{I}_{\leq h,w}$ the list of items of $\mathcal{I}_{\leq h,w}$, ordered by non-decreasing value of their exiting time step (the exiting time step of an item $i \in \mathcal{I}$ is $s_i + d_i$). Let $\mathrm{I}_w[k]$ (resp. $\mathrm{I}_{\leq h,w}[k]$) be the item at index k in I_w (resp. $\mathrm{I}_{\leq h,w}$).

Let $\beta^{\ddagger}(h, w, k)$ be the maximum profit that can be obtained considering the selection of items in $\mathrm{I}_{\leq h,w}$ at an index lower than or equal to $k \in \{0, \ldots, |\mathrm{I}_{\leq h,w}|\}$. The forward recursion is as follows:

$$\beta^{\ddagger}(h, w, k) = \begin{cases} 0 & \text{if } k = 0 \\ \max\left\{p_{\mathrm{I}_{\leq h,w}[k]} + \beta^{\ddagger}(h_{\mathrm{I}_{\leq h,w}[k]}, w, k-1), \beta^{\ddagger}(h_{\mathrm{I}_{\leq h,w}[k]}, w, k-1)\right\} & \text{otherwise} \end{cases}. \tag{7}$$

Note that if an item could be selected any number of times, only the first term within the maximum operator in (7) would be necessary. However, both terms are necessary to allow the non-selection of an item in a trivial suproblem, and to keep the arc-flow formulation valid in the hypergraph associated with the dynamic program.

If a stage 3 state $v = (h, w, L, e, \mathbf{q})$ is detected trivial, we compute the index k_e of an item in $\mathrm{I}_{\leq h,w}$ related to the first outgoing event occurring before or exactly at e, i.e., $k_e = \max\{k \in \{1, \ldots, |\mathrm{I}_{\leq h,w}|\} : e^{\mathrm{out}}(\mathrm{I}_{\leq h,w}[k]) \leq e\}$ where $e^{\mathrm{out}}(i)$ is the outgoing event of an item $i \in \mathcal{I}$. Then we do the following substitution:

$$\alpha_3(h, w, L, e, \mathbf{q}) = \beta^{\ddagger}(h, w, k_e). \tag{8}$$

We refer to the states created by the combined use of equations (7) and (8) as trivial stage 3 states, but we denote them as $(h, w, k)^{\ddagger}$, where $h \in \mathcal{H}, w \in \mathcal{W}$ and $k \in \{1, \ldots, |\mathrm{I}_{\leq h,w}|\}$.

Let $\beta^{\dagger}(w, k)$ be the maximum profit that can be obtained given a selection of items in I_w with an index less than or equal to $k \in \{0, \ldots, |\mathrm{I}_w|\}$. The forward recursion is as follows:

$$\beta^{\dagger}(w, k) = \begin{cases} 0 & \text{if } k = 0 \\ \max\left\{p_{\mathrm{I}_w[k]} + \beta^{\dagger}(w, k-1), \beta^{\dagger}(w, k-1)\right\} & \text{otherwise} \end{cases}. \tag{9}$$

For each height $H \in \{0, \ldots, H_{\max}\}$ and width $w \in \mathcal{W}_H$, let $k_w(H)$ be the index in I_w of the first item of width w whose height is less than or equal to H, i.e., $k_w(H) = \max\{k \in \{0, \ldots, |\mathrm{I}_w|\} : h_{\mathrm{I}_w[k]} \leq H\}$. If a stage 2 state (h, W) or a stage 1 state (H) is found to be trivial, we make the following substitutions:

$$\alpha_2(h, W) = \sum_{w \in \mathcal{W}_h : w \leq W} \beta^{\dagger}(w, k_w(h)), \tag{10}$$

$$\alpha_1(H) = \sum_{w \in \mathcal{W}_H} \beta^{\dagger}(w, k_w(H)). \tag{11}$$

As before, we refer to the states created by the combined use of the equations (9), (10) and (11) as trivial states, but we denote them as $(w, k)^{\dagger}$, where $w \in \mathcal{W}$ and $k \in \{1, \ldots, |\mathrm{I}_w|\}$.

In the specific case where a stage 3 state $(h, w, L_{\max}, E^*_{h,w}, \mathbf{0})$ is detected trivial, we use the following non-recursive formula to indicate whether all or none of the items are selected:

$$\alpha_3(h, w, L_{\max}, E^*_{h,w}, \mathbf{0}) = \max\left\{\sum_{i \in \mathcal{I}_{\leq h,w}} p_i,\ 0\right\}. \tag{12}$$

Again, if an item could be selected any number of times, only the first term within the maximum operator in (12) would be needed. We show below that Equation (12) is valid.

Dominance rule 7. *If a stage 3 state $(h, w, L_{\max}, E^*_{h,w}, \mathbf{0})$ is trivial, then there exists at least one optimal solution to the SCPD problem where exactly one of the following assertions is true:*

- *there is no compartment of size* (h, w).
- *all compartments of size* (h, w) *are empty.*
- *exactly one compartment of size* (h, w) *is not empty and contains all items of* $\mathcal{I}_{\leq h,w}$.

Proof. Let S^* be an optimal solution to the SCPD problem and assume that none of the three assertions is true. This means that there exist multiple non-empty compartments of size (h, w), or a single non-empty compartment of size (h, w) containing only a subset of $\mathcal{I}_{\leq h,w}$. In both cases, a solution that satisfies the above condition can be obtained by removing all items from all compartments of size (h, w), and assigning all items of $\mathcal{I}_{\leq h,w}$ to a single compartment. This is always feasible since $(h, w, L_{\max}, E^*_{h,w}, \mathbf{0})$ is trivial. □

Figure 10 illustrates how the recursive formula (9) and Equation (10) leads to the creation of hyperarcs when we encounter a trivial stage 2 state (h, W). In this example, the items have widths in the set $\{1, 2, 3\}$. Note that this modification to the dynamic program makes the design of shelves and compartments implicit in the decision hypergraph. Once the arc-flow formulation has been solved, if the flow entering trivial states is non-zero, we consider the same design decisions as in the solution where each item is assigned, but we remove the items that are not selected.

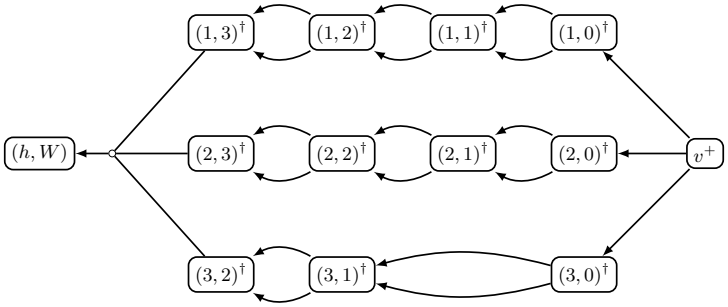


Figure 10: Illustration of the hyperarcs created from a trivial stage 2 state (h, W)

We now extend the notion of trivial subproblems. If a stage 1 state (H) is not trivial, it may happen that if one considers the possibility of building only shelves of height $h < H$ or less in a cabinet of height H, all admissible items can be selected. In this case, we say that v is $h-$*trivial*. We specify this notion in Definition 3.

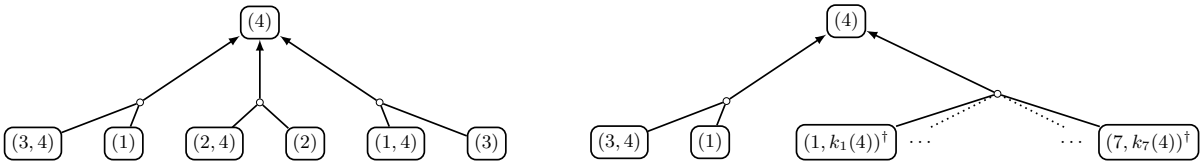
Definition 3. *Let* $H \in \{0, \ldots, H_{\max}\}$ *and* $h \in \mathcal{H}$ *such that* $h < H$. *A stage 1 state* (H) *is said to be* $h-$*trivial if* $\eta(v) \supseteq \mathbb{P}(\mathcal{I}_{\leq h})$.

If a stage 1 state $v = (H)$ is detected $\overline{h}-$trivial for $\overline{h} \in \mathcal{H}$, then Equation (4) can be replaced by

$$\alpha_1(H) = \max \left\{ \max_{h \in \mathcal{H}, \overline{h} < h \leq H} \{\alpha_2(h, W_{\max}) + \alpha_1(H - h)\}, \sum_{w \in \mathcal{W}_{\overline{h}}} \beta^\dagger(\overline{h}, w) \right\}. \tag{13}$$

This equation is derived by separating a stage 1 state into two cases. Either a shelf with a height greater than $\overline{h}$ is built, in which case the recursion does not change from the initial one, or a shelf of height less than or equal to $\overline{h}$ is built, in which case, by Dominance rule 2 only shelves of height $\overline{h}$ or less will be subsequently used. Since, by assumption, the problem is trivial by using only shelves of height $\overline{h}$ of less, the right-hand part of the equation, which allows the selection of any subset of items of height less than $\overline{h}$, is valid. Figure 11 illustrates how the recursive formula (9) and equation (13) leads to the creation of hyperarcs when we encounter a stage 1 state (H) that is $\overline{h}-$trivial with $H = 4$ and $\overline{h} = 2$. In this example, the items have a height in the set $\{1, 2, 3\}$, a width in the set $\{1, \ldots, 7\}$, and there is enough space to place each item with height less than 2 in shelves of height 2. The creation of shelves of height 1 is ignored, and the creation of a shelf of height 2 is replaced by a transition to the trivial states.

The following observation highlights the fact that all of the proposed reductions can be applied together while maintaining the validity of the arc-flow model.



(a) Without considering that (4) is 2−trivial

(b) Considering that (4) is 2−trivial

Figure 11: Example of h-trivial stage 1 problems reduction

Observation 1. *Because* (12) *is the only simplification of the dynamic program that reduces the set of partial solutions in terms of item selection, simultaneously using* (8)*,* (10)*,* (11)*,* (12)*, and* (13) *does not cut off all optimal solutions to the SCPD problem.*

To conclude this section, we now describe our procedures for detecting whether a state v is trivial or $h-$trivial if v is a stage 1 state. We detect whether a stage 3 state $v = (h, w, L, e, \mathbf{q})$ is trivial by trying to select each item $i \in \mathcal{I}_{\leq h,w}$ such that $s_i + d_i \leq t(e)$. We formalize this procedure in Proposition 6. For each $e \in \mathcal{E}$ and $\mathbf{q} \in \{0,1\}^{|\mathcal{I}|}$, let $\sigma(e)$ be the event $\tilde{e} \neq e$ such that $i(\tilde{e}) = i(e)$ and $\tilde{\ell}(e, \mathbf{q})$ defined as follows:

$$\tilde{\ell}(e, \mathbf{q}) = \begin{cases} -\ell_{i(e)} & \text{if } e \in \mathcal{E}^{\text{out}} \\ \ell_{i(e)} & \text{if } e \in \mathcal{E}^{\text{in}} \wedge \sigma(e) > e \wedge \mathbf{q}_{i(e)} = 1 \\ 0 & \text{if } e \in \mathcal{E}^{\text{in}} \wedge \sigma(e) > e \wedge \mathbf{q}_{i(e)} = 0 \\ \ell_{i(e)} & \text{if } e \in \mathcal{E}^{\text{in}} \wedge \sigma(e) \leq e \end{cases}.$$

Proposition 6. *A stage 3 state $v = (h, w, L, e, \mathbf{q})$ is trivial if and only if the following condition holds for each $\tilde{e} \in \mathcal{E}_{h,w}$ with $\tilde{e} < e$:*

$$L + \sum_{\hat{e}=\tilde{e}}^{e} \tilde{\ell}(\hat{e}, \mathbf{q}) \geq 0. \tag{14}$$

Proof. Let $v = (h, w, L, e, \mathbf{q})$ be a stage 3 state. The state v is trivial if, at each time step before $t(e)$, each item is selected in the solution, and the remaining available length in the compartment does not fall below 0. This condition is equivalent to the statement that, for each event $\tilde{e} \leq e$, the sum of the contributions to the length constraint of events between $\tilde{e}$ and e is less than or equal to $L_{\max}$. Therefore, v is trivial if Condition (14) holds. □

We determine whether a stage 2 problem is trivial by computing an upper bound on the minimum width required to select each admissible item on the shelf. If this value is less than or equal to the width $W_{\max}$ of the cabinet, then the stage 2 problem is trivial.

Proposition 7. *A stage 2 state $v = (h, W)$ is trivial if the following condition holds:*

$$\sum_{w \in \mathcal{W}_h} w \cdot z^*_{TBP(\mathcal{I}_{\leq h,w}, L_{\max})} \leq W. \tag{15}$$

Proof. Let $v = (h, W)$ be a stage 2 state. For each width $w \in \mathcal{W}_h$, $z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}$ is the minimum number of compartments of width w required to select each item $i \in \mathcal{I}_{\leq h,w}$. Hence, $\sum_{w \in \mathcal{W}_h} w \cdot z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}$ denotes an upper bound on the minimum width required to place each item on the shelf. □

Finally, let $\underline{H}(\mathcal{J})$ be the minimum height of a cabinet of width $W_{\max}$ and length $L_{\max}$ required to select all items in $\mathcal{J} \subseteq \mathcal{I}$. For a given $h \in \mathcal{H}$, we compute $\underline{H}(\mathcal{I}_{\leq h})$ by solving a BP problem considering $W_{\max}$ as the capacity of the bins and defining $\mathcal{J}$ with $z^*_{\text{TBP}(\mathcal{I}_{\leq h,w}, L_{\max})}$ items with weight w for each $w \in \mathcal{W}_h$. Each item of weight w in the BP instance corresponds to a bin in a solution to the TBP problem defined with the items from $\mathcal{I}_{\leq h,w}$ (the weight of these latter items in the TBP problem is equal to their length in the SCPD problem).

We associate to each item of the BP instance a height computed as the maximum height of an item of $\mathcal{I}_{\leq h,w}$ assigned to the corresponding bin in the solution to the TBP problem. For each bin in the optimal solution to the BP problem, we compute the maximum height of an item assigned to it. The value $\underline{H}(\mathcal{I}_{\leq h})$ is then the sum of the maximum height of each bin. We determine whether a stage 1 state $v = (H)$ is $h-$trivial by comparing $\underline{H}(\mathcal{I}_{\leq h})$ and h. This procedure does not require optimally solving the BP and the TBP problems to compute a valid upper bound.

Proposition 8. *A stage 1 state $v = (H)$ is $h-$trivial with $h \in \mathcal{H}$ such that $h \leq H$ if $\underline{H}(\mathcal{I}_{\leq h}) \leq H$.*

Note that if $v = (H)$ is $h-$trivial and there is no $\tilde{h} \in \mathcal{H}$ such that $h < \tilde{h} \leq H$, then v is trivial.

6 Computational experiments

In this section, we first explain in Section 6.1 how we generated instances to the SCPD problem and state in Section 6.2 some improvements we implemented for the compact formulation. We then describe in Sections 6.3 and 6.4 the outcomes of the computational experiments we performed.

The aim of our experiments is twofold. First, we assess the added value of the improvements described in Sections 5.1, 5.2, 5.3 and 5.4 on the results of the arc-flow formulation. Second, we compare using several metrics (size of the formulation, integrality gap, number of instances solved, solution time, and optimality gap) the results obtained by the solver with the compact MILP formulation (1) and the arc-flow formulation (5) with and without all the improvements introduced in Section 5.

6.1 Data sets

Our experiments are conducted using randomly generated instances[1], using a methodology similar to Caprara et al. (2013) for the temporal knapsack aspect of the problem. In each instance, the box has dimensions $(H_{\max}, W_{\max}, L_{\max}) = (100, 100, 100)$. To ensure that the compartments represent relevant problems, we randomly draw $\mathsf{m} \in \mathbb{N}_{>0}$ values from the discrete uniform distribution $\mathcal{U}\{r_{\min}, r_{\max}\}$. Let M denote the set of drawn values. The widths of the items can only be elements of M. The heights of the items are also randomly drawn from $\mathcal{U}\{r_{\min}, r_{\max}\}$, with no limitation on the number of different heights present in the instance. The lengths and profits of the items are randomly drawn from $\mathcal{U}\{1, 100\}$. For the time intervals, we separate the instances into two classes.

For the first class of instances, denoted I, we generate the time intervals using a *clique-based* methodology. The notion of clique comes from the interval graph representation of the time intervals. A clique in an interval graph corresponds to a set of pairwise overlapping intervals. In the clique-based framework, we consider each width independently of the others. For each width of M, we generate T cliques, where each time step corresponds to a different clique of items. The generation algorithm uses two integer parameters a and b. It starts by creating a set of items at the first time step, i.e., items i such that $s_i = 0$. Iteratively, at each time step t, the algorithm creates a new maximal clique from the previous one by removing $(100 - \mathsf{b})\%$ items, i.e., items such that $d_i = t - s_i$, and adding a new items, i.e., items such that $s_i = t$. The algorithm stops when the last clique has been generated. Trivially, each instance has $\mathsf{m} \cdot \mathsf{a} \cdot \mathsf{T}$ items. Algorithm 12 in Section S.5 of the supplementary material summarizes the generation algorithm for the instances of class I. For our experiments, we set $\mathsf{a} = 5$ and $\mathsf{b} = 90$.

For the second class of instances, denoted U, and for each width of M, we generate k items where the starts of the time intervals are randomly drawn from $\mathcal{U}\{0, 1000\}$ and their durations are drawn from $\mathcal{U}\{100, 500\}$. Trivially, each instance has $\mathsf{m} \cdot \mathsf{k}$ items. Algorithm 13 in Section S.5 of the supplementary material summarizes the generation algorithm for the instances of class U.

[1] available online from `https://gitlab.inria.fr/edge/scpd/scpd-artifacts`

We now define the different instance groups we created and the numerical values of the parameters used to generate the instances. Table 3 summarizes the generation parameters for the instances of classes I and U, respectively. We generate 10 instances per group for a total of 200 instances.

Group	T	$r_{\min}$	$r_{\max}$	m
I	5	20	40	5
II	10	20	40	5
III	15	20	40	5
IV	10	20	40	10
V	5	10	20	5
VI	10	10	20	5
VII	15	10	20	5
VIII	10	10	20	10
IX	10	5	10	5
X	15	5	10	5

(a) Class I

Group	k	$r_{\min}$	$r_{\max}$	m
XI	20	20	40	5
XII	40	20	40	5
XIII	60	20	40	5
XIV	40	20	40	10
XV	20	10	20	5
XVI	40	10	20	5
XVII	60	10	20	5
XVIII	40	10	20	10
XIX	40	5	10	5
XX	60	5	10	5

(b) Class U

Table 3: Generation parameters for the instances

6.2 Implementation details and experimental setting

To allow a fair comparison between the compact and arc-flow formulations, we implemented some of the improvements of Section 5 in the compact formulation, namely Dominance rules 3 and 5, as well as the exploitation of trivial subproblems. Because the compact formulation does not order the shelves and the compartments, Dominance rules 2 and 4 are not applicable. Dominance rule 6 is implicitly considered in constraints (1i). To improve the linear relaxation of the compact formulation, in a pre-processing step, we also lift the dimensions of the items, taking inspiration from an algorithm introduced by Boschetti et al. (2002). The goal is to increase the length, width, and height of the items without modifying the set of feasible solutions. The improved compact formulation is detailed in Section S.6 of the supplementary material. In the remaining, let **C** denote the improved compact formulation. All of our experiments were run on a Haswell Intel Xeon E5-2680 v3 CPU running at 2.5 GHz with 128 Go RAM. For each instance, we solved each MILP formulation with CPLEX 20.1 using 8 threads with a CPU time limit of 60 minutes. Regarding the improvements described in Sections 5.2 and 5.4 that require the solution of TBP and BP problems, we solved the compact MILP formulation described in Section 3.1 of Dell'Amico et al. (2020) with CPLEX 20.1 with default parameters and a time limit of 10 seconds. The individual results for every tested instance are available online[2].

The linear relaxation of the arc-flow model (5a)-(5e) was solved using a column generation algorithm. The pricing subproblem is a search of a flow with value 1 in the decision hypergraph, i.e., it contains constraints (5b), (5c) and (5e). The master problem contains constraints (5d) and the convexity constraint. Because the system of constraints of the subproblem is totally dual integral, the optimum value of the master problem and the value of the LP relaxation of our arc-flow formulation are the same. Using column generation is motivated by the fact that, in our preliminary experiments, the linear relaxation of the arc-flow model could take more than 60 hours to solve for some instances.

In the remainder, we note **AF** the arc-flow formulation (5) and $\mathbf{AF}^{+\mathbf{ALL}}$ the arc-flow formulation (5) with

[2] `https://gitlab.inria.fr/edge/scpd/scpd-artifacts`

the improvements described in Section 5. We consider an instance to be solved if the solver returns a proven optimal solution.

6.3 Evaluating the improvements made to the arc-flow formulation

In this section, we examine the individual added value of the improvements to the formulation **AF** contained in the formulation $\mathbf{AF}^{+\mathbf{ALL}}$. For this purpose, we classify the improvements into four families, each family corresponding to ones of the Sections 5.1 (valid inequalities, **VI**), 5.2 (pruning hyperarcs, **PH**), 5.3 (merging equivalent states, **MES**) and 5.4 (trivial subproblems, **TS**). For each family of improvements, we consider the addition of this family to formulation **AF**, obtaining the formulations $\mathbf{AF}^{+\mathbf{VI}}$, $\mathbf{AF}^{+\mathbf{PH}}$, $\mathbf{AF}^{+\mathbf{MES}}$ and $\mathbf{AF}^{+\mathbf{TS}}$, respectively. Second, considering the formulation $\mathbf{AF}^{+\mathbf{ALL}}$, for each family of improvements, we remove the improvements listed in the corresponding subsection, creating the configurations $\mathbf{AF}^{+\mathbf{ALL}-\mathbf{VI}}$, $\mathbf{AF}^{+\mathbf{ALL}-\mathbf{PH}}$, $\mathbf{AF}^{+\mathbf{ALL}-\mathbf{MES}}$ and $\mathbf{AF}^{+\mathbf{ALL}-\mathbf{TS}}$, respectively. For each configuration, we report in Table 4 the average ratio between the number of variables and constraints of the model obtained and those of **AF** and the total number of instances solved. The largest formulation created contains around 17 million variables and 16 million constraints.

Formulation	Variables	Constraints	Solved	Difference
AF	1	1	95	—
$\mathbf{AF}^{+\mathbf{VI}}$	1	1	89	-6
$\mathbf{AF}^{+\mathbf{PH}}$	1.0	1.0	99	+4
$\mathbf{AF}^{+\mathbf{MES}}$	0.9	0.9	102	+7
$\mathbf{AF}^{+\mathbf{TS}}$	0.8	0.8	100	+5

Variables/Constraints: The average ratio between the number of variables and the number of constraints of the formulation and the formulation **AF**.
Solved: The number of instances solved (to optimality).
Difference: The difference in the number of instances solved between the formulation and the formulation **AF**.

(a) Arc-flow formulation with one family of improvements added

Formulation	Variables	Constraints	Solved	Difference
$\mathbf{AF}^{+\mathbf{ALL}}$	1	1	108	—
$\mathbf{AF}^{+\mathbf{ALL}-\mathbf{VI}}$	1	1	110	+2
$\mathbf{AF}^{+\mathbf{ALL}-\mathbf{PH}}$	1.0	1.0	103	-5
$\mathbf{AF}^{+\mathbf{ALL}-\mathbf{MES}}$	1.1	1.1	102	-6
$\mathbf{AF}^{+\mathbf{ALL}-\mathbf{TS}}$	1.2	1.2	98	-10

Variables/Constraints: The average ratio between the number of variables and the number of constraints of the formulation and the formulation $\mathbf{AF}^{+\mathbf{ALL}}$.
Solved: The number of instances solved (to optimality).
Difference: The difference in the number of instances solved between the formulation and the formulation $\mathbf{AF}^{+\mathbf{ALL}}$.

(b) Arc-flow formulation with each family of improvements added except one

Table 4: Impact of the improvements on the size and the number of the instances solved using the arc-flow formulation

Exploiting trivial subproblems (**TS**) is the most effective technique to reduce the size of the model. It removes 20% variables and constraints from the original formulation, and applying the other techniques does not compensate for its absence. Merging equivalent states (**MES**) has the largest impact in terms of instances solved, with 102 instances solved. The number of variables and constraints in the formulation $\mathbf{AF}^{+\mathbf{ALL}}$ is reduced by about 22% on average compared to the formulation **AF**. Valid inequalities (**VI**) do not improve the performance of **AF**, with six fewer instances solved, and they are detrimental to the quality of the results when all other techniques are used, with two instances less solved. From the results, we observe that the best configuration of the arc-flow formulation is $\mathbf{AF}^{+\mathbf{ALL}-\mathbf{VI}}$, i.e., to use all the improvements described in this paper with the exception of valid inequalities. With this configuration, hereafter renamed as $\mathbf{AF}^{*}$, 110 solutions are proven optimal, compared to 95 with **AF**.

6.4 Comparing the results obtained with the compact and the arc-flow formulations

We now compare different versions of the arc-flow formulations and the compact formulation in terms of integrality gap (Table 5), number of instances solved, and number of best primal and dual bounds found (Table

6). We also report in Figure 12 a performance profile with the number of solved instances over time and the number of instances for which the optimality gap is less than a given value.

We first study the linear relaxations of the different formulations. We consider only the 110 instances for which the optimal value is known (i.e., the instance has been solved during at least one of our tests). Table 5 reports the average integrality gap over each instance group for each formulation. For each formulation and each of these instances, we compute the *integrality gap* with the formula $100 \cdot \frac{|z^* - d|}{|d|}$ where z^* is the optimal value of the instance, and d is the value obtained solving by the linear relaxation of the formulation.

Group	$\|\mathcal{I}\|$	Integrality gap				
		C	**AF**	**AF$^{+\mathbf{VI}}$**	**AF$^{+\mathbf{ALL-VI}}$**	**AF$^{+\mathbf{ALL}}$**
I	125	43.46	4.68	4.49	4.66	4.46
II	250	55.74	3.83	3.66	3.83	3.66
III	375	58.79	1.69	1.69	1.69	1.69
IV	500	68.27	3.02	2.91	3.02	2.91
V	125	0.49	0.38	0.38	0.38	0.38
VI	250	—	—	—	—	—
VII	375	—	—	—	—	—
VIII	500	—	—	—	—	—
IX	250	0	0	0	0	0
X	375	0	0	0	0	0
XI	100	35.01	3.96	3.78	3.92	3.73
XII	200	45.98	4.65	4.42	4.62	4.37
XIII	300	58.09	4.40	4.38	4.40	4.38
XIV	400	66.58	4.21	4.02	4.20	4.00
XV	100	0	0	0	0	0
XVI	200	—	—	—	—	—
XVII	300	—	—	—	—	—
XVIII	400	—	—	—	—	—
XIX	200	0	0	0	0	0
XX	300	0	0	0	0	0
Mean	—	27.02	2.15	2.06	2.14	2.05

Table 5: Integrality gap of the formulations

We observe that the arc-flow formulations have a significantly smaller integrality gap than the compact formulation. For the compact formulation, the gap between the dual bound obtained at the root node of the branch-and-bound tree (when solving the MILP) and the optimal value is on average larger than 8% and therefore still significantly higher than the average integrality gap obtained with **AF** which is around 2%. The valid inequalities we add help to improve the linear relaxation of the formulation **AF**. Specifically, the formulation **AF$^{+\mathbf{VI}}$** has a better linear relaxation than the formulation **AF** for 45 out of the 200 instances. The difference between the integrality gap obtained with and without the valid inequalities is at most 0.61%. We observe that the other improvements of Section 5 further improve the linear relaxation of the formulation **AF**. For 50 instances, the integrality gap is equal to 0. In our experiments, this corresponds to instances where all the items are selected in the solution in an optimal solution, such as all the instances of groups IX, X, XIX, and XX.

We now examine for which formulation the solver returns the largest number of proven optimal solutions or gives the best solutions among the formulations **C** and **AF***. For each instance, we calculate the best primal solution and the best dual solution obtained by the solver over the two formulations. For each group of ten instances and each formulation, Table 6 reports in columns "Primal best", "Dual best" and "Solved", the number of times the best primal solution is obtained with the formulation, the number of times the best dual solution is obtained with the formulation and the number of instances solved, respectively. The gray values in the column "Primal best" (resp. "Dual best") are the number of times the solver found a feasible primal (resp. dual) solution with this formulation.

Group	$\|\mathcal{I}\|$	**C**			**AF**			**AF***		
		Primal best	Dual best	Solved	Primal best	Dual best	Solved	Primal best	Dual best	Solved
I	125	10 /10	9 /10	9	10 /10	10 /10	10	10 /10	10 /10	10
II	250	6 /10	2 /10	1	8 /10	6 /10	6	10 /10	9 /10	7
III	375	1 /10	0 /10	0	2 /10	4 /10	1	9 /9	7 /9	1
IV	500	0 /10	0 /10	0	10 /10	5 /10	5	10 /10	10 /10	9
V	125	9 /10	0 /10	0	7 /10	0 /10	0	10 /10	10 /10	1
VI	250	1 /10	4 /10	0	3 /10	0 /10	0	6 /10	6 /10	0
VII	375	9 /10	0 /10	0	0 /5	1 /5	0	1 /10	9 /10	0
VIII	500	0 /10	0 /10	0	2 /8	1 /8	0	8 /10	9 /10	0
IX	250	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
X	375	10 /10	10 /10	10	5 /8	8 /8	5	10 /10	10 /10	10
XI	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XII	200	6 /10	0 /10	0	10 /10	6 /10	6	10 /10	10 /10	10
XIII	300	1 /10	0 /10	0	9 /10	6 /10	3	8 /10	7 /10	3
XIV	400	0 /10	0 /10	0	10 /10	9 /10	9	10 /10	10 /10	9
XV	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XVI	200	5 /10	1 /10	0	1 /10	1 /10	0	8 /10	10 /10	0
XVII	300	0 /10	0 /10	0	2 /9	1 /9	0	8 /10	9 /10	0
XVIII	400	0 /10	0 /10	0	3 /10	1 /10	0	7 /10	9 /10	0
XIX	200	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XX	300	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
Total	—	108 /200	76 /200	70	132 /190	109 /190	95	175 /199	185 /199	110

Primal best: The number of times the formulation found the best primal solution and, in gray, the number of times the formulation found a feasible primal solution.

Dual best: The number of times the formulation found the best dual solution and, in gray, the number of times the formulation found a feasible dual solution.

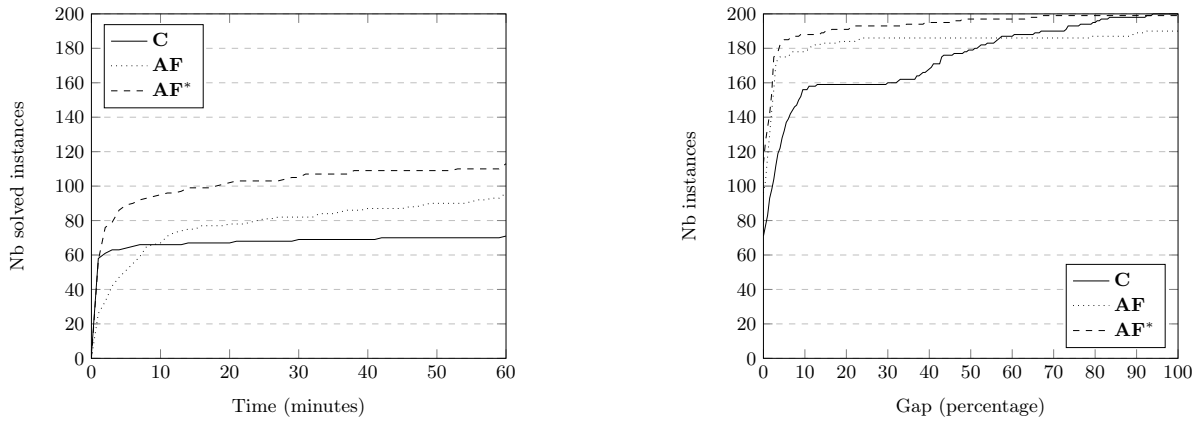
Solved: The number of instances solved (to optimality) by the formulation.

Table 6: Results obtained by the solver within 3600 seconds for the formulations **C**, **AF** and **AF***

We observe that 40 more instances are solved with the formulation **AF*** than with the formulation **C**. Every instance solved using **C** is also solved using **AF***. Of the 110 solved instances, the 50 instances where all the items are selected are all solved in less than 21 seconds with the formulation **AF*** and in less than 28 seconds with the formulation **C**. Formulation **AF*** finds the best primal solution known in 175 cases out of 200, against 108 cases for **C**. The difference is significantly larger concerning the dual bounds: **AF*** leads to the best dual bound for 185 cases, against 76 for **C**. For the 13 instances (resp. 92) instances where the solver returns a better primal solution with **C** (resp. **AF***) than with **AF*** (resp. **C**), the gap between the value of

the primal solution obtained by the two formulations is on average equal to 25.87% (resp. 8.43%). This gap is calculated using the formula $100 \cdot \frac{|z^* - z|}{|z^*|}$ where z^* is the primal bound for the formulation **C** (resp. **AF***) and z is the primal bound for the formulation **AF*** (resp. **C**). None of the formulations can solve the instances of groups VI, VII, VIII, XVI, XVII, and XVIII. In these instances, the items have small heights and widths in relation to the height and width of the box, compared with the same values for instances of groups II, III, IV, XII, XIII, and XIV. This suggests that the instances where the design aspects are more prominent are the most challenging unless all items fit in the box (instances of groups IX, X, XIX, and XX).

We finally examine the solution times and the gaps obtained for the formulations **C**, **AF** and **AF***. Figure 12a shows the number of instances solved for each formulation as a function of time. We observe that formulations **C** and **AF*** allow to solve the 50 easiest instances in less than two minutes. In contrast, **AF** takes 10 minutes to solve those instances. However, the number of solved instances increases regularly, while **C** struggles to solve instances when it has not converged in a few minutes. Figure 12b plots for each formulation the number of instances for which the optimality gap is smaller than the value given on the x-axis. For each formulation and each instance, the optimality gap is calculated using the formula $100 \cdot \frac{|z-d|}{|d|}$ where z and d are the primal and dual bounds returned by the solver. Overall, better optimality gaps are obtained with the formulation **AF***. The average optimality gap is equal to 12.94%, 3.32% and 2.68% for formulations **C**, **AF** and **AF***, respectively. For the 40 instances solved with the formulation **AF*** but not with the formulation **C**, the average optimality gap returned by the solver for the latter is 29.32%. All the results in this section indicate that for the instances we have generated, the results obtained with the formulation **AF*** are significantly better than those obtained with the formulation **C**.



(a) Number of solved instances over time

(b) Number of instances over gap

Figure 12: Performance profiles for the three MILP formulations

6.5 Studying the case of correlated instances

We now compare our formulations on instances where the profits of the items are correlated with some metric. We created correlated instances from those of groups I to XX by modifying their profits according to four types of correlation: length (**L**), i.e., $p_i = \ell_i$, temporal length (**TL**), i.e., $p_i = \ell_i d_i$, volume (**V**), i.e., $p_i = h_i w_i \ell_i$ and temporal volume (**TV**), i.e., $p_i = h_i w_i \ell_i d_i$ for each item $i \in \mathcal{I}$.

Table 7 shows for each type of correlation and for each method the absolute difference between the number of instances solved with this correlation and without correlation (baseline). Tables S.8, S.9, S.10 and S.11 of the supplementary material show a more detailed view of the results on correlated instances. The correlations **L**, **V** and **TV** make the instances easier to solve on average for all formulations. More specifically, the number of instances solved with each formulation increases. The correlation **TL** makes the instances easier to solve for **AF** and **AF***, but not for **C**, where one fewer instance is solved. Introducing the volume in the correlation

makes the instances significantly easier (correlations **V**, and **TV**). The average gaps for correlations **L**, **TL**, **V**, and **TV** are respectively 15.94%, 16.37%, 15.04%, and 15.6% for **C**, 1.9%, 1.25%, 2.05%, and 0.86% for **AF**, and 2.1%, 1.46%, 1.81%, and 0.72% for **AF***. However, hard classes where no instances are solved to optimality by any method remain hard. This is the opposite of what is observed for one-dimensional problems, but this is consistent with the results from the literature on problems that the SCPD generalizes. For the 2TDK (see Hifi and Roucairol (2001) and Lodi and Monaci (2003)) and the TKP (see Caprara et al. (2013)), the correlated instances are easier to solve than the uncorrelated ones. To better understand this difference in difficulty, we analyzed the results obtained over the 55 instances where **C**, **AF**, and **AF*** have optimally solved the uncorrelated and the correlated versions of the instances. The difference does not come from a better linear relaxation: we have not observed a significantly better integrality gap for the correlated instances. In contrast, the average number of shelves, compartments, and items in the solutions is lower for the correlated instances by 6.37%, 11.81%, and 16.70%, respectively. This is not surprising: optimal solutions tend to contain many large items whose profits are significantly higher than the profits of the small items.

Correlation	Solved		
	C	**AF**	**AF***
None	70	95	110
L	+3	+14	+14
TL	-1	+16	+18
V	+14	+28	+20
TV	+13	+31	+21

Solved: The number of instances solved (to optimality). The values are given as the difference between the number of instances solved for the correlation and the number of instances solved without any correlation.

Table 7: Number of instances solved for each type of correlation

7 Conclusion

In this paper, we considered the design of shelves and compartments in a storage cabinet. We defined the problem as a combination of a guillotine-cutting problem and a set of temporal knapsack problems. We presented a compact mixed-integer linear program for it and an arc-flow formulation based on an exponentially large hypergraph corresponding to a decision hypergraph of a dynamic program for a relaxation of the SCPD problem. We showed that using an unimproved arc-flow formulation already leads to more solved instances than a compact formulation. We also presented several improvements to reduce the size of the hypergraph while maintaining the validity of the formulation. With the introduced refinements, we reduced the hypergraph to approximately 78% of its original size. The arc-flow formulation based on this reduced hypergraph produces better results than those obtained with the original hypergraph, solving 15 more instances and providing better primal and dual bounds. Compared to the compact formulation, 40 more instances are solved using the arc-flow formulation on the reduced hypergraph, and the average gap is reduced from 12.94% to 2.68%.

This paper is one of the few contributions leveraging arc-flow models in hypergraphs for solving a combinatorial optimization problem. It shows that they are efficient tools for modeling and solving problems with a recursive structure, even when they combine two hard combinatorial optimization problems. Our experiments confirm that the quality of their LP relaxation is affected by additional constraints (in our case, the bounds on the number of times an item can be selected). To address this issue, we introduced a set of valid inequalities.

They slightly improved the linear relaxation of the model, but their inclusion is detrimental to the solver's performance. Investigating new classes of valid inequalities for hypergraph-based arc-flow models is a promising research direction. From a methodological point of view, several filtering and reduction techniques we developed specifically for our formulation can be generalized to more general decision hypergraphs. Being able to detect and process them automatically would be useful for a broader class of applications.

Our approach was unable to find any optimal solution for several classes of instances (groups VI, VII, VIII, XVI, XVII, and XVIII). Several improvements can be studied. Computing an initial primal bound using ad-hoc heuristics could help the solver eliminate additional variables at the root node. The size of the formulation is also an issue for large instances. Several techniques, using column-and-row generation or dynamic state-space relaxation techniques, could be investigated to avoid generating the full model.

Considering the SCPD problem, several generalizations can be studied. Modeling the extension to multiple cabinets would be straightforward with the arc-flow formulations. In contrast, stochastic versions of the problem would lead to significantly harder problems. In practice, the problem is multi-stage, and at each time period, the time intervals are discovered, and the decision to accept or not each new item is made. A first useful approximation can be obtained by considering a two-stage stochastic model, where the set of possible items is known in the design phase, with an estimation of the time intervals, and their precise values are revealed in the second stage.

8 Acknowledgments

This research was funded by the French Agence Nationale de la Recherche through project AD-LIB (ANR-22-CE23-0014-01). Experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB, and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux and CNRS (and ANR in accordance to the programme d'investissements d'Avenir (see `http://www.plafrim.fr`))

References

Arenales, M. and Morabito, R. (1995). An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. *European Journal of Operational Research*, 84(3):599–617.

Bartlett, M., Frisch, A. M., Hamadi, Y., Miguel, I., Tarim, S. A., and Unsworth, C. (2005). The Temporal Knapsack Problem and Its Solution. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3524, pages 34–48. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bergman, D., Cire, A. A., van Hoeve, W.-J., and Hooker, J. (2016). *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, Cham.

Bianchi-Aguiar, T., Hübner, A., Carravilla, M. A., and Oliveira, J. F. (2021). Retail shelf space planning problems: A comprehensive review and classification framework. *European Journal of Operational Research*, 289(1):1–16.

Bonsma, P., Schulz, J., and Wiese, A. (2014). A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM journal on computing*, 43(2):767–799.

Boschetti, M. A., Mingozzi, A., and Hadjiconstantinou, E. (2002). New upper bounds for the two-dimensional orthogonal non-guillotine cutting stock problem. *Ima Journal of Management Mathematics*, 13:95–119.

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

Caprara, A., Furini, F., and Malaguti, E. (2013). Uncommon dantzig-wolfe reformulation for the temporal knapsack problem. *INFORMS Journal on Computing*, 25(3):560–571.

Caprara, A., Furini, F., Malaguti, E., and Traversi, E. (2016). Solving the temporal knapsack problem via recursive dantzig-wolfe reformulation. *Information Processing Letters*, 116(5):379 – 386.

Cardona, L. F. and Gue, K. R. (2020). Layouts of unit-load warehouses with multiple slot heights. *Transportation Science*, 54(5):1332–1350.

Castro, M. P., Cire, A. A., and Beck, J. C. (2022). Decision Diagrams for Discrete Optimization: A Survey of Recent Advances. *INFORMS Journal on Computing*, 34(4):2271–2295.

Christofides, N. and Whitlock, C. (1977). An algorithm for two-dimensional cutting problems. *Oper. Res.*, 25:30–44.

Clausen, J. V., Lusby, R., and Ropke, S. (2022). Consistency cuts for dantzig-wolfe reformulations. *Operations Research*, 70(5):2883–2905.

Clautiaux, F., Detienne, B., and Guillot, G. (2021). An iterative dynamic programming approach for the temporal knapsack problem. *European Journal of Operational Research*, 293(2):442–456.

Clautiaux, F., Sadykov, R., Vanderbeck, F., and Viaud, Q. (2018). Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization*, 29:18–44.

de Lima, V. L., Alves, C., Clautiaux, F., Iori, M., and de Carvalho, J. M. V. (2022). Arc Flow Formulations Based on Dynamic Programming: Theoretical Foundations and Applications. *European Journal of Operational Research*, 296(1):3–21.

Dell'Amico, M., Furini, F., and Iori, M. (2020). A branch-and-price algorithm for the temporal bin packing problem. *Computers & Operations Research*, 114:104825.

Esmaili, N., Norman, B. A., and Rajgopal, J. (2018). Shelf-space optimization models in decentralized automated dispensing cabinets. *Operations Research for Health Care*, 19:92–106.

Gecili, H. and Parikh, P. J. (2022). Joint shelf design and shelf space allocation problem for retailers. *Omega*, 111:102634.

Gschwind, T. and Irnich, S. (2017). Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities. *OR Spectrum*, 39(2):541–556.

Hifi, M. and Roucairol, C. (2001). Approximate and exact algorithms for constrained (un) weighted two-dimensional two-staged cutting stock problems. *Journal of Combinatorial Optimization*, 5:465–494.

Hübner, A., Düsterhöft, T., and Ostermeier, M. (2021). Shelf space dimensioning and product allocation in retail stores. *European Journal of Operational Research*, 292(1):155–171.

Jouglet, A. and Carlier, J. (2011). Dominance rules in combinatorial optimization problems. *European Journal of Operational Research*, 212(3):433–444.

Lodi, A., Martello, S., and Vigo, D. (2004). Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8:363–379.

Lodi, A. and Monaci, M. (2003). Integer linear programming models for 2-staged two-dimensional knapsack problems. *Mathematical Programming*, 94(2):257–278.

Macedo, R., Alves, C., and De Carvalho, J. V. (2010). Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research*, 37(6):991–1001.

Martin, R. K., Rardin, R. L., and Campbell, B. A. (1990). Polyhedral Characterization of Discrete Dynamic Programming. *Operations Research*, 38(1):127–138.

Martinovic, J., Strasdat, N., de Carvalho, J. V., and Furini, F. (2023). A combinatorial flow-based formulation for temporal bin packing problems. *European Journal of Operational Research*, 307(2):554–574.

Scheithauer, G. (2018). Introduction to cutting and packing optimization. *Operations Research and Management Science*.

Valério de Carvalho, J. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629–659.

Supplementary material

S.1. Complexity of the SCPD-Decision problem

We prove that the SCPD-Decision problem is NP-hard by polynomially reducing the Temporal Knapsack decision problem (Problem 5), known to be NP-hard in the strong sense (Bonsma et al., 2014), to it.

Problem 5 (Temporal Knapsack decision problem **(TK-Decision problem)**). *An instance of the Temporal Knapsack decision (TK-Decision) problem is a tuple* $(C, \mathcal{J}, \overline{B})$ *where* $C \in \mathbb{N}$ *is the capacity of the knapsack,* $\mathcal{J}$ *is a set of* items*, and* $\overline{B} \in \mathbb{R}_+$*. Each item* $j \in \mathcal{J}$ *has a weight* $\overline{w}_j \in \mathbb{N}$*, a profit* $\overline{p}_j \in \mathbb{R}_+$ *and a time interval* $[\overline{s}_j, \overline{s}_j + \overline{d}_j)$ *with* $\overline{s}_j \in \mathbb{N}$ *and* $\overline{d}_j \in \mathbb{N}$*, during which, when selected, it is present in the knapsack. Given* $(C, \mathcal{J}, \overline{B})$*, the TK-Decision problem is formulated as follows: is there* $\mathcal{J}' \subseteq \mathcal{J}$ *such that the following two conditions are satisfied?*

(i) At each time step, the sum of the weights of the items assigned to the knapsack in that time step is less than or equal to C*:*

$$\forall t \in \bigcup_{j \in \mathcal{J}'} \{\overline{s}_j, \overline{s}_j + \overline{d}_j\}, \qquad \sum_{j \in \mathcal{J}' : \overline{s}_j \leq t < \overline{s}_j + \overline{d}_j} \overline{w}_j \leq C.$$

(ii) The sum of the profits of the items assigned to the knapsack is at least $\overline{B}$*:*

$$\sum_{j \in \mathcal{J}'} \overline{p}_j \geq \overline{B}.$$

Proposition 9 (NP-hardness of the SCPD-Decision problem). *The SCPD-Decision problem is NP-hard in the strong sense.*

Proof. The TK-Decision problem is polynomially reducible to the SCPD-Decision problem. To reduce an instance $(C, \mathcal{J}, \overline{B})$ of the TK-Decision problem to an instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}, B\big)$ of the SCPD-Decision problem, take $(H_{\max}, W_{\max}, L_{\max}) = (1, 1, C)$, $B = \overline{B}$ and construct $\mathcal{I}$ as follows: for each item $j \in \mathcal{J}$, create an item $i \in \mathcal{I}$ with attributes $h_i = 1$, $w_i = 1$, $l_i = \overline{w}_j$, $p_i = \overline{p}_j$, $s_i = \overline{s}_j$, and $d_i = \overline{d}_j$. □

S.2. Hypergraph generation algorithms

Algorithm 1: Generating the decision hypergraph associated with the dynamic program defined by the recursive formulas (2)–(4)

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$

Output: The decision hypergraph associated with the dynamic program applied to the given instance

$\mathcal{V} \leftarrow \{v^+, (H_{\max})\}$; // set of vertices of the hypergraph

$\mathcal{A} \leftarrow \emptyset$; // set of hyperarcs

$\mathcal{S}^1 \leftarrow \{(H_{\max})\}$; // stage 1 states to process

$\mathcal{S}^2 \leftarrow \emptyset$; // stage 2 states to process

$\mathcal{S}^3 \leftarrow \emptyset$; // stage 3 states to process

while $\mathcal{S}^1 \neq \emptyset$ **do**

  $v = (H) \leftarrow$ element of $\mathcal{S}^1$ that maximizes H ;

  $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \setminus \{v\}$;

  Apply Algorithm 2 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^1$, $\mathcal{S}^2$, and $(\mathcal{V}, \mathcal{A})$ as inputs ;

while $\mathcal{S}^2 \neq \emptyset$ **do**

  $v = (h, W) \leftarrow$ element of $\mathcal{S}^2$ that maximizes lexicographically h and W ;

  $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \setminus \{v\}$;

  Apply Algorithm 3 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^2$, $\mathcal{S}^3$, and $(\mathcal{V}, \mathcal{A})$ as inputs ;

while $\mathcal{S}^3 \neq \emptyset$ **do**

  $v = (h, w, L, e, \mathbf{q}) \leftarrow$ element of $\mathcal{S}^3$ that maximizes lexicographically h, w, e ;

  $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \setminus \{v\}$;

  Apply Algorithm 4 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^3$ and $(\mathcal{V}, \mathcal{A})$ as inputs ;

Algorithm 2: Create the decision hyperarcs incoming to a vertex associated with a stage 1 state

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a stage 1 state $v = (H)$, two sets $\mathcal{S}^1$ and $\mathcal{S}^2$, a hypergraph $(\mathcal{V}, \mathcal{A})$

Result: Creates the hyperarcs incoming to v, updates $\mathcal{V}, \mathcal{A}, \mathcal{S}^1$, $\mathcal{S}^2$

for $h \in \mathcal{H}, h \leq H$ **do**

  $v_1 \leftarrow (h, W_{\max})$; // generate a stage 2 state

  $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$, $\mathcal{F} \leftarrow \{v_1\}$;

  if $\exists i \in \mathcal{I}, h_i \leq H - h$ **then**

    $v_2 \leftarrow (H - h)$; // generate a stage 1 state

    $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \cup \{v_2\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$;

  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$; // create the hyperarc

Algorithm 3: Create the decision hyperarcs incoming to a vertex associated with a stage 2 state

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a stage 2 state $v = (h, W)$, two sets $\mathcal{S}^2$ and $\mathcal{S}^3$, a hypergraph $(\mathcal{V}, \mathcal{A})$

Result: Creates the hyperarcs incoming to v, updates $\mathcal{V}, \mathcal{A}, \mathcal{S}^2$, $\mathcal{S}^3$

for $w \in \mathcal{W}_h, w \leq W$ **do**
- $v_1 \leftarrow (h, w, L_{\max}, E^*_{h,w}, \mathbf{0})$; // generate a stage 3 state
- $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$,$\mathcal{F} \leftarrow \{v_1\}$;
- **if** $\exists i \in \mathcal{I} : h_i \leq h, w_i \leq W - w$ **then**
  - $v_2 \leftarrow (h, W - w)$; // generate a stage 2 state
  - $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_2\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$;
- $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$; // create the hyperarc

Algorithm 4: Create the decision hyperarcs incoming to a vertex associated with a stage 3 state

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a stage 3 state $v = (h, w, L, e, \mathbf{q})$, a set $\mathcal{S}^3$, a hypergraph $(\mathcal{V}, \mathcal{A})$

Result: Creates the hyperarcs incoming to v, updates $\mathcal{V}, \mathcal{A}, \mathcal{S}^3$

if $e = 0$ **then**
- $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v^+\}, 0)\}$; // connect the source with this state

else if $e \in \mathcal{E}^{out}$ **then**
- $v_1 \leftarrow (h, w, L, E^-_{h,w}(e), \mathbf{q})$; // skipping the item
- $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
- $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; // create the hyperarc
- **if** $L \geq \ell_{i(e)}$ **then**
  - $v_2 \leftarrow (h, w, L - \ell_{i(e)}, E^-_{h,w}(e), \mathbf{q} + \boldsymbol{\varepsilon_{i(e)}})$; // selecting the item
  - $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_2\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$;
  - $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_2\}, p_{i(e)})\}$; // create the hyperarc

else if $e \in \mathcal{E}^{in}$ **then**
- **if** $\mathbf{q}_{i(e)} = 0$ **then**
  - $v_1 \leftarrow (h, w, L, E^-_{h,w}(e), \mathbf{q})$; // skipping the item
- **else**
  - $v_1 \leftarrow (h, w, L + \ell_{i(e)}, E^-_{h,w}(e), \mathbf{q} - \boldsymbol{\varepsilon_{i(e)}})$; // selecting the item
- $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
- $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; // create the hyperarc

S.3. Conversion from a flow to a solution to the SCPD problem algorithm

Algorithm 5: Convert a solution to the arc-flow model to a solution to the SCPD problem

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ the decision hypergraph, $x \in \mathbb{N}^{|\mathcal{A}|}$ the solution to the arc-flow model

Output: A solution to the SCPD problem

$\mathcal{I}' \leftarrow \emptyset$; // Set of items in the solution

$\mathcal{L}^1 \leftarrow \{(H_{\max})\}$; // stage 1 states to process

$\mathcal{L}^2 \leftarrow \emptyset$; // pairs (stage 2 state, shelf) to process

$\mathcal{L}^3 \leftarrow \emptyset$; // pairs (stage 3 state, compartment) to process

while $\mathcal{L}^1 \neq \emptyset$ **do**

  $v = (H) \leftarrow$ arbitrary element of $\mathcal{L}^1$;

  $\mathcal{L}^1 \leftarrow \mathcal{L}^1 \setminus \{v\}$;

  Choose arbitrarily $\mathcal{F}$ such that $x_{(\mathcal{F},v)} \geq 1$;

  $x_{(\mathcal{F},v)} \leftarrow x_{(\mathcal{F},v)} - 1$;

  for $u \in \mathcal{F}$ **do**

    if *u is a stage 2 state* $(h_u, W_{\max})$ **then**

      $\psi \leftarrow (h_u, W_{\max}, L_{\max})$; // creation of a shelf with height h_u

      $\Psi \leftarrow \Psi \cup \{\psi\}$;

      $\mathcal{L}^2 \leftarrow \mathcal{L}^2 \cup \{(u, \psi)\}$; // add the shelf information to u

    else

      $\mathcal{L}^1 \leftarrow \mathcal{L}^1 \cup \{u\}$;

while $\mathcal{L}^2 \neq \emptyset$ **do**

  $(v, \psi) = ((h_\psi, W), \psi) \leftarrow$ arbitrary element of $\mathcal{L}^2$;

  $\mathcal{L}^2 \leftarrow \mathcal{L}^2 \setminus \{(v, \psi)\}$;

  Choose arbitrarily $\mathcal{F}$ such that $x_{(\mathcal{F},v)} \geq 1$;

  $x_{(\mathcal{F},v)} \leftarrow x_{(\mathcal{F},v)} - 1$;

  for $u \in \mathcal{F}$ **do**

    if *u is a stage 3 state* $(h_\psi, w_u, L_{\max}, E^*_{h_\psi, w_u}, \mathbf{0})$ **then**

      $\phi \leftarrow (h_\psi, w_u, L_{\max})$; // creation of a compartment with height h_ψ, width w_u

      $\Phi \leftarrow \Phi \cup \{\phi\}$, $\mu(\phi) \leftarrow \psi$;

      $\mathcal{L}^3 \leftarrow \mathcal{L}^3 \cup \{(u, \phi)\}$; // add the compartment information to u

    else

      $\mathcal{L}^2 \leftarrow \mathcal{L}^2 \cup \{(u, \psi)\}$;

while $\mathcal{L}^3 \neq \emptyset$ **do**

  $(v, \phi) = ((h_\phi, w_\phi, L, e, \mathbf{q}), \phi) \leftarrow$ arbitrary element of $\mathcal{L}^3$;

  $\mathcal{L}^3 \leftarrow \mathcal{L}^3 \setminus \{(v, \phi)\}$;

  Choose arbitrarily $\mathcal{F}$ such that $x_{(\mathcal{F},v)} \geq 1$;

  $x_{(\mathcal{F},v)} \leftarrow x_{(\mathcal{F},v)} - 1$;

  $u = (h_\phi, w_\phi, \tilde{L}, \tilde{e}, \tilde{\mathbf{q}}) \leftarrow$ the single element of $\mathcal{F}$;

  if $e \in \mathcal{E}^{out}$ *and* $\tilde{\mathbf{q}}_{i(e)} = 1$ **then**

    $\rho(i(e)) \leftarrow \phi$;

    $\mathcal{I}' \leftarrow \mathcal{I}' \cup \{i(e)\}$;

  if $\tilde{e} \neq 0$ **then**

    $\mathcal{L}^3 \leftarrow \mathcal{L}^3 \cup \{(u, \phi)\}$

return $(\Psi, \Phi, \mathcal{I}', \mu, \rho)$

S.4. Hypergraph generation algorithms with all improvements of Section 5

Algorithm 6: Generating the decision hypergraph associated with the dynamic programs (2)–(4) and the improvements of Section 5

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$

Output: A decision hypergraph associated with the given instance

$\mathcal{V} \leftarrow \{s, (H_{\max})\}$; // set of vertices of the hypergraph

$\mathcal{A} \leftarrow \emptyset$; // set of hyperarcs

$\mathcal{S}^1 \leftarrow \{(H_{\max})\}$; // stage 1 states to process

$\mathcal{S}^2 \leftarrow \emptyset$; // stage 2 states to process

$\mathcal{S}^3 \leftarrow \emptyset$; // stage 3 states to process

$\mathcal{S}^\dagger \leftarrow \emptyset$, $\mathcal{S}^\ddagger \leftarrow \emptyset$; // trivial states to process

while $\mathcal{S}^1 \neq \emptyset$ **do**

  $v = (H) \leftarrow$ element of $\mathcal{S}^1$ that maximizes h ;

  $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \setminus \{v\}$;

  Apply Algorithm 7 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^1$, $\mathcal{S}^2$, $\mathcal{S}^\dagger$ and $(\mathcal{V}, \mathcal{A})$ as inputs ;

while $\mathcal{S}^2 \neq \emptyset$ **do**

  $v = (h, W) \leftarrow$ element of $\mathcal{S}^2$ that maximizes lexicographically h and w ;

  $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \setminus \{v\}$;

  Apply Algorithm 8 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^2$, $\mathcal{S}^3$, $\mathcal{S}^\dagger$, and $(\mathcal{V}, \mathcal{A})$ as inputs ;

while $\mathcal{S}^3 \neq \emptyset$ **do**

  $v = (h, w, L, e, \mathbf{q}) \leftarrow$ element of $\mathcal{S}^3$ that maximizes lexicographically h, w, e ;

  $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \setminus \{v\}$;

  Apply Algorithm 9 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^3$, $\mathcal{S}^\ddagger$ and $(\mathcal{V}, \mathcal{A})$ as inputs ;

while $\mathcal{S}^\dagger \neq \emptyset$ **do**

  $v = (w, k)^\dagger \leftarrow$ element of $\mathcal{S}^\dagger$ that maximizes lexicographically w, k ;

  $\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \setminus \{v\}$;

  Apply Algorithm 10 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^\dagger$ and $(\mathcal{V}, \mathcal{A})$ as inputs ;

while $\mathcal{S}^\ddagger \neq \emptyset$ **do**

  $v = (h, w, k)^\ddagger \leftarrow$ element of $\mathcal{S}^\ddagger$ that maximizes lexicographically h, w, k ;

  $\mathcal{S}^\ddagger \leftarrow \mathcal{S}^\ddagger \setminus \{v\}$;

  Apply Algorithm 11 with $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, v, $\mathcal{S}^\ddagger$ and $(\mathcal{V}, \mathcal{A})$ as inputs ;

Algorithm 7: Create the decision hyperarcs incoming to a vertex associated with a stage 1 state (improved)

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a stage 1 state $v = (H)$, three sets $\mathcal{S}^1$, $\mathcal{S}^2$ and $\mathcal{S}^\dagger$, a hypergraph $(\mathcal{V}, \mathcal{A})$

Result: Creates the hyperarcs incoming to v, updates $\mathcal{V}, \mathcal{A}, \mathcal{S}^1$, $\mathcal{S}^2$, and $\mathcal{S}^\dagger$

$\overline{h} \leftarrow$ the maximum height such that v is $\overline{h}-$trivial, 0 if it does not exist ; // $h-$triviality of v

$\tilde{h} \leftarrow +\infty$; // Dominance rule 2

if $\{\pi : c_1(v, \pi) = 1\} \neq \emptyset$ **then**

  $\tilde{h} \leftarrow \min\{\pi : c_1(v, \pi) = 1\}$;

$\tilde{h} \leftarrow \min\{\tilde{h}, h\}$;

$C_2(h) \leftarrow$ True if $h \leq \tilde{h}$, False otherwise ; // Dominance rule 2

$C_3(h) \leftarrow$ True if $c_1(v, h) < U_1(h)$, False otherwise ; // Dominance rule 3

$C_7(h) \leftarrow$ False if $h \geq \overline{h}$, True otherwise ; // $h-$triviality of v

for $h \in \mathcal{H}, h \leq H$ *and* $C_2(h)$ *and* $C_3(h)$ *and not* $C_7(h)$ **do**

  $\hat{H} \leftarrow \max\{\pi \in \mathcal{RH}_h : \pi \leq H - h\}$; // Proposition 3

  $\hat{W} \leftarrow \max\{\pi \in \mathcal{RW}_h : \pi \leq W_{\max}\}$; // Proposition 4

  $v_1 \leftarrow (h, \hat{W})$; // generate a stage 2 state

  $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$, $\mathcal{F} \leftarrow \{v_1\}$;

  if $\exists i \in \mathcal{I}, h_i \leq \hat{H}$ **then**

    $v_2 \leftarrow (\hat{H})$; // generate a stage 1 state

    $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \cup \{v_2\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$;

  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$; // generate a hyperarc

if $\overline{h} \neq 0$ **then** // $h-$triviality of v

  for $w \in \mathcal{W}_{\overline{h}}$ **do**

    $k_w(\overline{h}) \leftarrow \max\left\{k \in \{0, \ldots, |\mathrm{I}_w|\} : h_{\mathrm{I}_w[k]} \leq \overline{h}\right\}$;

    $v_1 \leftarrow (w, k_w(\overline{h}))^\dagger$;

    $\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;

    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; // generate a hyperarc

Algorithm 8: Create the decision hyperarcs incoming to a vertex associated with a stage 2 state (improved)

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a stage 2 state $v = (h, W)$, three sets $\mathcal{S}^2$, $\mathcal{S}^3$, and $\mathcal{S}^\dagger$ a hypergraph $(\mathcal{V}, \mathcal{A})$

Result: Creates the hyperarcs incoming to v, updates $\mathcal{V}, \mathcal{A}, \mathcal{S}^2$, $\mathcal{S}^3$ and $\mathcal{S}^\dagger$

if *v is trivial* **then**
  for $w \in \mathcal{W}_h$ **do**
    $k_w(h) \leftarrow \max\big\{k \in \{0, \ldots, |\mathrm{I}_w|\} : h_{\mathrm{I}_w[k]} \leq h\big\}$;
    $v_1 \leftarrow (h, w, k_w(h))^\ddagger$;
    $\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \cup \{v_1\}$,
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; `// generate a hyperarc`
else
  $C_5(w) \leftarrow$ True if $c_2(v, w) < U_2(h, w)$, False otherwise ; `// Dominance rule 5`
  $C_6(w) \leftarrow$ True if $(((W = W_{\max}) \wedge (\exists i \in \mathcal{I}_{\leq h} : h_i = h, w_i = w)) \vee W \neq W_{\max})$, False otherwise ; `// Dominance rule 6`
  for $w \in \mathcal{W}_h, w \leq W$ *and* $C_5(w)$ *and* $C_6(w)$ **do**
    $\hat{h} \leftarrow h_{\max}(\mathcal{I}_{\leq h,w})$; `// Dominance rule 4`
    $\hat{W} \leftarrow \max\{\pi \in \mathcal{RW}_{h_{\max}(\mathcal{I}_{\leq h,w})} : \pi \leq W - w\}$; `// Proposition 4`
    $v_1 \leftarrow (\hat{h}, w, L_{\max}, E^*_{\hat{h},w}, \mathbf{0})$;
    $v_2 \leftarrow (\hat{h}, \hat{W})$;
    $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$, $tail \leftarrow \mathcal{F} \cup \{v_1\}$;
    if $\exists i \in \mathcal{I} : h_i \leq h, w_i \leq w - \tilde{w}$ **then**
      $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_2\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$;
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$; `// generate a hyperarc`

Algorithm 9: Create the decision hyperarcs incoming to a vertex associated with a stage 3 state (improved)

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a stage 3 state $v = (h, w, L, e, \mathbf{q})$, a set $\mathcal{S}^3$ and $\mathcal{S}^{\ddagger}$ a hypergraph $(\mathcal{V}, \mathcal{A})$

Result: Adds predecessors to v, updates $\mathcal{V}, \mathcal{A}, \mathcal{S}^3$, and $\mathcal{S}^{\ddagger}$

if *v is trivial* **then**
  if $e = E^*_{h,w}$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v^+\}, \sum_{i \in \mathcal{I}_{\leq h,w}} p_i), (v, \{v^+\}, 0)\}$;
  else
    $k_e \leftarrow \max\{k \in \{1, \ldots, |\mathrm{I}_{\leq h,w}|\} : e^{\mathrm{out}}(\mathrm{I}_{\leq h,w}[k]) \leq e\}$;
    $v_1 \leftarrow (h, w, k_e)^{\ddagger}$;
    $\mathcal{S}^{\ddagger} \leftarrow \mathcal{S}^{\ddagger} \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; `// generate a hyperarc`
else
  if $e = 0$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{s\}, 0)\}$; `// connect the source with this state`
  else if $e \in \mathcal{E}^{out}$ **then**
    $\hat{h} \leftarrow H^-(E^-_{h,w}(e), h)$; `// Proposition 5`
    $v_1 \leftarrow (\hat{h}, w, L, E^-_{h,w}(e), \mathbf{q})$; `// skipping the item`
    $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; `// generate a hyperarc`
    if $L \geq \ell_{i(e)}$ **then**
      $v_2 \leftarrow (\hat{h}, w, L - \ell_{i(e)}, E^-_{h,w}(e), \mathbf{q} + \boldsymbol{\varepsilon_{i(e)}})$; `// selecting the item`
      $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_2\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$;
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_2\}, p_{i(e)})\}$; `// generate a hyperarc`
  else if $e \in \mathcal{E}^{in}$ **then**
    $\hat{h} \leftarrow H^-(E^-_{h,w}(e), h)$; `// Proposition 5`
    if $\mathbf{q}_{i(e)} = 0$ **then**
      $v_1 \leftarrow (\hat{h}, w, L, E^-_{h,w}(e), \mathbf{q})$; `// skipping the item`
    else
      $v_1 \leftarrow (\hat{h}, w, L + \ell_{i(e)}, E^-_{h,w}(e), \mathbf{q} - \boldsymbol{\varepsilon_{i(e)}})$; `// selecting the item`
    $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$; `// generate a hyperarc`

Algorithm 10: Create trivial states

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a trivial state $v = (w, k)^{\dagger}$, a set $\mathcal{S}^{\dagger}$, a hypergraph $(\mathcal{V}, \mathcal{A})$

if $k = 0$ **then**
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{s\}, 0)\}$; `// connect the source with this state`
else
  $v_1 \leftarrow (w, k-1)^{\dagger}$;
  $\mathcal{S}^{\dagger} \leftarrow \mathcal{S}^{\dagger} \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0), (v, \{v_1\}, p_{\mathrm{I}_w[k]})\}$; `// generate two hyperarcs`

Algorithm 11: Create trivial states

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, a trivial state $v = (h, w, k)^{\ddagger}$, a set $\mathcal{S}^{\ddagger}$, a hypergraph $(\mathcal{V}, \mathcal{A})$

if $k = 0$ **then**
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{s\}, 0)\}$; `// connect the source with this state`
else
  $v_1 \leftarrow (H^{-}(k-1, h), w, k-1)^{\ddagger}$;
  $\mathcal{S}^{\ddagger} \leftarrow \mathcal{S}^{\ddagger} \cup \{v_1\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$;
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0), (v, \{v_1\}, p_{\mathrm{I}_{\leq h, w}[k]})\}$; `// generate two hyperarcs`

S.5. Instance generation

Algorithm 12: Generation algorithm for the instances of class I

Input: m the number of different widths, T the number of cliques per width

Output: An instance of the SCPD problem

$\mathcal{I} \leftarrow \emptyset$ denotes the set of items ;
$\mathsf{M} \leftarrow \emptyset$ denotes the set of different widths ;
for $k \in \{0, \ldots, \mathsf{m}\}$ **do**
    $w \leftarrow$ value from $\mathcal{U}\{r_{\min}, r_{\max}\}$;
    $\mathsf{M} \leftarrow \mathsf{M} \cup \{w\}$;
for $w \in \mathsf{M}$ **do**
    $c \leftarrow \emptyset$ denotes a clique of items ;
    for $k \in \{1, \ldots, \mathsf{a}\}$ **do**
        i denotes an item ;
        $h_i \leftarrow$ value from $\mathcal{U}\{r_{\min}, r_{\max}\}$;
        $w_i \leftarrow w$;
        $\ell_i \leftarrow$ value from $\mathcal{U}\{1, 100\}$;
        $p_i \leftarrow$ value from $\mathcal{U}\{1, 100\}$;
        $s_i \leftarrow 0$;
        $d_i \leftarrow -1$;
        $c \leftarrow c \cup \{i\}$;
    for $t \in \{1, \ldots, C\}$ **do**
        for $i \in c$ **do**
            $p \leftarrow$ value from $\mathcal{U}(0, 100)$;
            if $p > \mathsf{b}$ **then**
                $d_i \leftarrow t - s_i$;
                $I \leftarrow I \cup \{i\}$;
                $c \leftarrow c \setminus \{i\}$;
        for $k \in \{1, \ldots, \mathsf{a}\}$ **do**
            i denotes an item ;
            $h_i \leftarrow$ value from $\mathcal{U}\{r_{\min}, r_{\max}\}$;
            $w_i \leftarrow w$;
            $\ell_i \leftarrow$ value from $\mathcal{U}\{1, 100\}$;
            $p_i \leftarrow$ value from $\mathcal{U}\{1, 100\}$;
            $s_i \leftarrow t$;
            $c \leftarrow c \cup \{i\}$;
return $((100, 100, 100), \mathcal{I})$

Algorithm 13: Generation algorithm for the instances of class U

Input: m the number of different widths, k the number of items per width

Output: An instance of the SCPD problem

$\mathcal{I} \leftarrow \emptyset$ denotes the set of items ;
$\mathsf{M} \leftarrow \emptyset$ denotes the set of different widths ;
for $k \in \{0, \ldots, \mathsf{m}\}$ **do**
    $w \leftarrow$ value from $\mathcal{U}\{r_{\min}, r_{\max}\}$;
    $\mathsf{M} \leftarrow \mathsf{M} \cup \{w\}$;
for $w \in \mathcal{W}$ **do**
    for $k \in \{0, \ldots, \mathsf{k}\}$ **do**
        i denotes an item ;
        $h_i \leftarrow$ value from $\mathcal{U}\{r_{\min}, r_{\max}\}$;
        $w_i \leftarrow w$;
        $\ell_i \leftarrow$ value from $\mathcal{U}\{1, 100\}$;
        $p_i \leftarrow$ value from $\mathcal{U}\{1, 100\}$;
        $s_i \leftarrow$ value from $\mathcal{U}\{0, 1000\}$;
        $d_i \leftarrow$ value from $\mathcal{U}\{100, 500\}$;
return $((100, 100, 100), \mathcal{I})$

S.6. Improved compact formulation

We describe how we improve the compact formulation introduced in Section 3.

We start by lifting the dimensions of the items with Algorithm 14. The algorithm runs in three independent phases, one for the length, one for the width, and one for the height of the items, and requires an order of the items for each phase. For each dimension, we consider the items iteratively and calculate whether we can increase their dimension, without changing the set of feasible solutions to the problem, by solving subset-sum problems with a dynamic programming recursion.

For the length dimension, we compute (lines 1-6) for each time step at which it is present the smallest difference ℓ^* between the length of the cabinet $L_{\max}$ and the sum of the lengths of items that could be in the same compartment as a given item j. We increase the length of j by $L_{\max} - \ell^*$. By construction, the items that are used to obtain the value ℓ^* (identified by ξ^* in the algorithm) cannot have their length increased.

The width of an item cannot be increased independently of the other items of the same width. Let $\mathcal{J}$ a set of items sharing the same width $\tilde{w}$, a shelf can contain at most $\min\{\lfloor \frac{W_{\max}}{\tilde{w}} \rfloor, |\mathcal{J}|\}$ compartments with width $\tilde{w}$. Therefore, we compute (lines 7-20) the smallest difference $w^{(k)}$ between the width of the cabinet $W_{\max}$ and the sum of the widths of compartments that could be on the same shelf as k compartments of width $\tilde{w}$. If any solution can only contain k compartments of width $\tilde{w}$ or none, we can increase $\tilde{w}$ by $W_{\max} - w^{(k)}$. To consider all possible cases, we increase the width $\tilde{w}$ considering all the possible values for k.

For the height dimension, we compute (lines 21-26) the smallest difference h^* between the height of the cabinet $H_{\max}$ and the sum of the heights of shelves that would fit in the cabinet, one of which would have the height of a given item j. We then increase the height of j by $H_{\max} - h^*$, and again we do not consider increasing the height of items used to obtain the value h^* (identified by ξ^* in the algorithm).

Algorithm 14: Lifting algorithm

Input: An SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$, $\mathcal{O}^H$ a sorted set of items for the height dimension, $\mathcal{O}^W$ a sorted set of widths of items, $\mathcal{O}^L$ a sorted set of items for the length dimension

Output: A lifted equivalent SCPD instance $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$

$\mathcal{J} \leftarrow \mathcal{O}^L$;
while $\mathcal{J} \neq \emptyset$ **do**
  $j \leftarrow$ first element of $\mathcal{J}$;
  $\xi^*, \ell^* \leftarrow$ optimal solution and objective of
    $\max_{t \in \{s_j, \ldots, s_j + d_j - 1\}} \{\ell = \ell_j + \sum_{i \in \mathcal{I}_t \setminus \{j\} : w_i = w_j} \ell_i \xi_i : \ell \leq L_{\max}, \xi_i \in \{0,1\}, i \in \mathcal{I} \setminus \{j\}\}$;
  $\ell_j \leftarrow \ell_j + (L_{\max} - \ell^*)$;
  $\mathcal{J} \leftarrow \mathcal{J} \setminus \big\{\{j\} \cup \{k \in \mathcal{J} \setminus \{j\} : \xi^*_k = 1\}\big\}$;
$\mathcal{O} \leftarrow \mathcal{O}^W$;
while $\mathcal{O} \neq \emptyset$ **do**
  $\tilde{w} \leftarrow$ first element of $\mathcal{O}$;
  $\mathcal{J} \leftarrow \{i : i \in \mathcal{I}, w_i = \tilde{w}\}$;
  $\xi^* \leftarrow \mathbf{0}$;
  $\delta \leftarrow +\infty$;
  for $k \in \{1, \ldots, \min\{\lfloor \frac{W_{\max}}{\tilde{w}} \rfloor, |\mathcal{J}|\}\}$ **do**
    $\xi^{(k)}, w^{(k)} \leftarrow$ optimal solution and objective of
      $\max\{w = k \cdot \tilde{w} + \sum_{i \in \mathcal{I} \setminus \mathcal{J}} w_i \xi_i : w \leq W_{\max}, \xi_i \in \{0,1\}, i \in \mathcal{I} \setminus \mathcal{J}\}$;
    if $\lfloor \frac{W_{\max} - w^{(k)}}{k} \rfloor < \delta$ **then**
      $\xi^* \leftarrow \xi^{(k)}$;
      $\delta \leftarrow \lfloor \frac{W_{\max} - w^{(k)}}{k} \rfloor$;
  for $i \in \mathcal{J}$ **do**
    $w_i \leftarrow w_i + \delta$;
  $\mathcal{O} \leftarrow \mathcal{O} \setminus \big\{\{\tilde{w}\} \cup \{w \in \mathcal{W} : \exists i \in \mathcal{I}, w_i = w, \xi^*_i = 1\}\big\}$;
$\mathcal{J} \leftarrow \mathcal{O}^H$;
while $\mathcal{J} \neq \emptyset$ **do**
  $j \leftarrow$ first element of $\mathcal{J}$;
  $\xi^*, h^* \leftarrow$ optimal solution and objective of $\max\{h = h_j + \sum_{i \in \mathcal{I} \setminus \{j\}} h_i \xi_i : h \leq H_{\max}, \xi_i \in \{0,1\}, i \in \mathcal{I} \setminus \{j\}\}$;
  $h_j \leftarrow h_j + (H_{\max} - h^*)$;
  $\mathcal{J} \leftarrow \mathcal{J} \setminus \big\{\{j\} \cup \{k \in \mathcal{J} \setminus \{j\} : \xi^*_k = 1\}\big\}$;
Sort $\mathcal{I}$ by non-increasing height;
return $\big((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}\big)$

We now detail the modifications we apply to the compact formulation. First, note that, after applying the lifting algorithm, some items may have the same width when this was not originally the case, which would allow them to be assigned to the same compartment. However, this is not permitted by the definition of the SCPD problem. To prevent this, we introduce new notation. Let $\mathcal{K}(i)$ denote the set of items with the same width as i before applying the lifting algorithm. For each $\mathcal{J} \in \{\mathcal{K}(i) : i \in \mathcal{I}\}$, let $w(\mathcal{J})$ denote the width of the items in $\mathcal{J}$ prior to the application of the lifting algorithm. Let $\mathcal{Q}^c$ denote the set of items j such that a stage 3 state $\alpha_3(h_j, w_j, L_{\max}, E^*_{h_j,w_j}, \mathbf{0})$ would be trivial (Definition 2), and let $\mathcal{Q}^s$ denote the set of items i such that a stage 2 state $\alpha_2(h_i, W_{\max})$ would be trivial.

For $i \in \mathcal{I}$, the binary variable z_i is equal to 1 if a shelf is represented by item i, meaning it has height h_i, 0 otherwise. Each binary variable $y_{i,j}$ is equal to 1 if a compartment on the shelf represented by i is represented by item j, meaning it has width w_j, 0 otherwise. Each binary variable $x_{j,k}$ is equal to 1 if the item $k \in \mathcal{I}$ is assigned to the compartment represented by j, 0 otherwise. Finally, each binary variable $\tilde{x}_k$ is equal to 1 if k is assigned to a compartment represented by an item $j \in \mathcal{Q}^c$ or to a shelf represented by an item $i \in \mathcal{Q}^s$, 0 otherwise. In this case, we say that k is *trivially assigned*. For each $j \in \mathcal{Q}^c$, we do not create the variables $x_{j,k}$, and for each $i \in \mathcal{Q}^s$, we do not create the variables $y_{i,j}$. We recall that, for each height $h \in \mathcal{H}$, $U_1(h)$ denotes an upper bound on the number of shelves with height h designed into the cabinet and that, for each height $h \in \mathcal{H}$ and each width $w \in \mathcal{W}_h$, $U_2(h, w)$ denotes an upper bound on the number of compartments with width w designed into a shelf with height h. We also recall that the items are ordered by non-increasing height, i.e., $i < j$ implies $h_i \geq h_j$ (ties are broken arbitrarily).

An improved compact formulation of the SCPD problem is as follows:

$$\text{maximize} \quad \sum_{k \in \mathcal{I}} p_k \tilde{x}_k + \sum_{j \in \mathcal{I} \setminus \mathcal{Q}^c} \sum_{\substack{k \in \mathcal{K}(j): \\ k \geq j}} p_k x_{j,k} \tag{16a}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}} h_i z_i \leq H_{\max} \tag{16b}$$

$$\sum_{\substack{j \in \mathcal{I}: \\ j \geq i}} w_j y_{i,j} \leq W_{\max} \qquad i \in \mathcal{I} \setminus \mathcal{Q}^s \tag{16c}$$

$$\sum_{\substack{k \in \mathcal{K}(j): \\ k \geq j, \\ s_k \leq t < s_k + d_k}} \ell_k x_{j,k} \leq L_{\max} \qquad j \in \mathcal{I} \setminus \mathcal{Q}^c, t \in \mathcal{T}^{\text{nd}} \tag{16d}$$

$$\tilde{x}_k + \sum_{\substack{j \in \mathcal{K}(k) \setminus \mathcal{Q}^c: \\ j \leq k}} x_{j,k} \leq 1 \qquad k \in \mathcal{I} \tag{16e}$$

$$x_{j,k} \leq x_{j,j} \qquad j \in \mathcal{I} \setminus \mathcal{Q}^c, k \in \mathcal{K}(j), k > j \tag{16f}$$

$$x_{j,j} = \sum_{\substack{i \in \mathcal{I} \setminus \mathcal{Q}^s: \\ i \leq j}} y_{i,j} \qquad j \in \mathcal{I} \setminus \mathcal{Q}^c \tag{16g}$$

$$y_{i,j} \leq z_i \qquad i \in \mathcal{I} \setminus \mathcal{Q}^s, j \in \mathcal{I}, j > i \tag{16h}$$

$$y_{i,i} = z_i \qquad i \in \mathcal{I} \setminus \mathcal{Q}^s \tag{16i}$$

$$\sum_{i \in \mathcal{I}: h_i = h} z_i \leq U_1(h) \qquad h \in \mathcal{H} \tag{16j}$$

$$\sum_{\substack{j \in \mathcal{J}: \\ j \geq i}} y_{i,j} \leq U_2(h_i, w(\mathcal{J})) \qquad i \in \mathcal{I} \setminus \mathcal{Q}^s, \mathcal{J} \in \{\mathcal{K}(j) : j \in \mathcal{I}\} \tag{16k}$$

$$\tilde{x}_k \leq \sum_{\substack{i \in \mathcal{Q}^s: \\ i \leq k}} z_i + \sum_{\substack{i \in \mathcal{I} \setminus \mathcal{Q}^s: \\ i \leq k}} \sum_{\substack{j \in \mathcal{K}(k) \cap \mathcal{Q}^c: \\ i \leq j \leq k}} y_{i,j} \qquad k \in \mathcal{I} \tag{16l}$$

$$z_i \in \{0, 1\} \qquad i \in \mathcal{I} \tag{16m}$$

$$y_{i,j} \in \{0, 1\} \qquad i \in \mathcal{I} \setminus \mathcal{Q}^s, j \in \mathcal{I}, j \geq i \tag{16n}$$

$$x_{j,k} \in \{0,1\} \qquad j \in \mathcal{I} \setminus \mathcal{Q}^c, k \in \mathcal{K}(j), k \geq j \tag{16o}$$

$$\tilde{x}_k \in \{0,1\} \qquad k \in \mathcal{I} \tag{16p}$$

The objective function in (16a) is equal to the sum of profits of the assigned items. The constraints (16b), (16c) and (16d) are the same as in the compact formulation and model the dimensions constraints. The constraints (16e) ensure that each item is assigned at most once in the cabinet, where an item is either trivially assigned or assigned to a real compartment. Constraints (16f) and (16g) ensure that an item is assigned to a compartment only if that compartment is created and that the item representing that compartment is assigned to it. Constraints (16h) and (16i) ensure that a compartment is created on a shelf only if that shelf is created and that the item representing the shelf represents one of the compartments built on the shelf. Constraints (16j) and (16k) implement Dominance rules 3 and 5. Constraints (16l) ensure that an item is trivially assigned if a shelf represented by an item $i \in \mathcal{Q}^s$ or a compartment represented by an item $j \in \mathcal{Q}^c$ is created.

S.7. Results on the correlated instances

In this section, we provide a detailed view on the results obtained with formulations **C**, **AF**, and **AF*** on the correlated instances. The columns 'Primal best", "Dual best" and "Solved" correspond to the number of times the best primal solution is obtained with the formulation, the number of times the best dual solution is obtained with the formulation, and the number of instances that have been solved, respectively.

Group	$\|\mathcal{I}\|$	**C**			**AF**			**AF***		
		Primal best	Dual best	Solved	Primal best	Dual best	Solved	Primal best	Dual best	Solved
I	125	10 /10	9 /10	9	10 /10	10 /10	10	10 /10	10 /10	10
II	250	7 /10	2 /10	2	10 /10	10 /10	10	10 /10	10 /10	10
III	375	1 /10	0 /10	0	7 /10	5 /10	3	6 /10	8 /10	4
IV	500	0 /10	0 /10	0	10 /10	6 /10	6	10 /10	10 /10	9
V	125	4 /10	0 /10	0	5 /10	0 /10	0	10 /10	10 /10	1
VI	250	3 /10	5 /10	0	1 /10	2 /10	0	6 /10	3 /10	0
VII	375	8 /10	0 /10	0	1 /3	1 /3	0	1 /10	9 /10	0
VIII	500	0 /10	0 /10	0	6 /10	2 /10	0	5 /10	8 /10	0
IX	250	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
X	375	10 /10	10 /10	10	5 /7	7 /7	5	10 /10	10 /10	10
XI	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XII	200	5 /10	2 /10	2	10 /10	10 /10	10	10 /10	10 /10	10
XIII	300	0 /10	0 /10	0	10 /10	7 /10	7	10 /10	10 /10	10
XIV	400	0 /10	0 /10	0	10 /10	8 /10	8	10 /10	10 /10	10
XV	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XVI	200	3 /10	1 /10	0	4 /10	1 /10	0	3 /10	10 /10	0
XVII	300	0 /10	0 /10	0	4 /10	2 /10	0	6 /10	8 /10	0
XVIII	400	0 /10	0 /10	0	2 /10	3 /10	0	8 /10	7 /10	0
XIX	200	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XX	300	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
Total	—	101 /200	79 /200	73	145 /190	124 /190	109	165 /200	183 /200	124

Table 8: Results obtained by the solver on instances with length (**L**) correlation

Group	$\lvert\mathcal{I}\rvert$	**C**			**AF**			**AF***		
		Primal best	Dual best	Solved	Primal best	Dual best	Solved	Primal best	Dual best	Solved
I	125	9 /10	6 /10	6	10 /10	10 /10	10	10 /10	10 /10	10
II	250	3 /10	1 /10	1	10 /10	9 /10	9	10 /10	10 /10	9
III	375	0 /10	0 /10	0	9 /10	6 /10	5	8 /10	9 /10	7
IV	500	0 /10	0 /10	0	10 /10	6 /10	6	10 /10	10 /10	10
V	125	7 /10	0 /10	0	7 /10	1 /10	1	10 /10	10 /10	2
VI	250	5 /10	3 /10	0	2 /10	1 /10	0	4 /10	6 /10	0
VII	375	4 /10	0 /10	0	2 /6	1 /6	0	4 /10	9 /10	0
VIII	500	0 /10	0 /10	0	5 /10	1 /10	0	6 /10	9 /10	0
IX	250	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
X	375	10 /10	10 /10	10	5 /7	7 /7	5	10 /10	10 /10	10
XI	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XII	200	5 /10	2 /10	2	10 /10	10 /10	10	10 /10	10 /10	10
XIII	300	0 /10	0 /10	0	10 /10	6 /10	6	10 /10	10 /10	10
XIV	400	1 /10	0 /10	0	10 /10	9 /10	9	10 /10	10 /10	10
XV	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XVI	200	2 /10	1 /10	0	3 /10	2 /10	0	5 /10	9 /10	0
XVII	300	0 /10	0 /10	0	3 /10	2 /10	0	7 /10	8 /10	0
XVIII	400	0 /10	0 /10	0	5 /10	2 /10	0	5 /10	8 /10	0
XIX	200	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XX	300	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
Total	—	96 /200	73 /200	69	151 /193	123 /193	111	169 /200	188 /200	128

Table 9: Results obtained by the solver on instances with temporal length (**TL**) correlation

Group	$\|\mathcal{I}\|$	**C**			**AF**			**AF***		
		Primal best	Dual best	Solved	Primal best	Dual best	Solved	Primal best	Dual best	Solved
I	125	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
II	250	9 /10	6 /10	6	10 /10	10 /10	10	10 /10	10 /10	10
III	375	2 /10	0 /10	0	8 /10	8 /10	7	8 /9	9 /9	7
IV	500	0 /10	0 /10	0	10 /10	9 /10	9	10 /10	10 /10	10
V	125	6 /10	0 /10	0	4 /10	0 /10	0	10 /10	10 /10	3
VI	250	2 /10	2 /10	0	4 /10	1 /10	0	7 /10	7 /10	0
VII	375	6 /10	0 /10	0	0 /5	3 /5	0	4 /10	7 /10	0
VIII	500	0 /10	0 /10	0	3 /10	4 /10	0	7 /10	6 /10	0
IX	250	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
X	375	10 /10	10 /10	10	7 /7	7 /7	7	10 /10	10 /10	10
XI	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XII	200	9 /10	8 /10	8	10 /10	10 /10	10	10 /10	10 /10	10
XIII	300	5 /10	0 /10	0	10 /10	10 /10	10	10 /10	10 /10	10
XIV	400	0 /10	0 /10	0	10 /10	10 /10	10	10 /10	10 /10	10
XV	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XVI	200	3 /10	1 /10	0	1 /10	2 /10	0	8 /10	9 /10	0
XVII	300	0 /10	0 /10	0	4 /10	5 /10	0	6 /10	5 /10	0
XVIII	400	0 /10	0 /10	0	3 /10	3 /10	0	7 /10	7 /10	0
XIX	200	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XX	300	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
Total	—	112 /200	87 /200	84	144 /192	142 /192	123	177 /199	180 /199	130

Table 10: Results obtained by the solver on instances with volume (**V**) correlation

Group	$\lvert\mathcal{I}\rvert$	**C**			**AF**			**AF***		
		Primal best	Dual best	Solved	Primal best	Dual best	Solved	Primal best	Dual best	Solved
I	125	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
II	250	10 /10	7 /10	7	10 /10	10 /10	10	10 /10	10 /10	10
III	375	0 /10	0 /10	0	9 /10	8 /10	8	10 /10	9 /10	8
IV	500	0 /10	0 /10	0	10 /10	10 /10	10	10 /10	10 /10	10
V	125	7 /10	0 /10	0	7 /10	1 /10	1	10 /10	10 /10	3
VI	250	1 /10	1 /10	0	2 /10	0 /10	0	9 /10	9 /10	0
VII	375	4 /10	0 /10	0	1 /7	4 /7	0	5 /10	6 /10	0
VIII	500	0 /10	0 /10	0	2 /10	3 /10	0	8 /10	7 /10	0
IX	250	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
X	375	10 /10	10 /10	10	7 /8	8 /8	7	10 /10	10 /10	10
XI	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XII	200	10 /10	6 /10	6	10 /10	10 /10	10	10 /10	10 /10	10
XIII	300	2 /10	0 /10	0	10 /10	10 /10	10	10 /10	10 /10	10
XIV	400	0 /10	0 /10	0	10 /10	10 /10	10	10 /10	10 /10	10
XV	100	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XVI	200	3 /10	1 /10	0	2 /10	2 /10	0	7 /10	9 /10	0
XVII	300	0 /10	0 /10	0	4 /10	2 /10	0	6 /10	8 /10	0
XVIII	400	0 /10	1 /10	0	3 /10	1 /10	0	7 /10	8 /10	0
XIX	200	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
XX	300	10 /10	10 /10	10	10 /10	10 /10	10	10 /10	10 /10	10
Total	—	107 /200	86 /200	83	147 /195	139 /195	126	182 /200	186 /200	131

Table 11: Results obtained by the solver on instances with temporal volume (**TV**) correlation