



**HAL**  
open science

# Mathematical models based on decision hypergraphs for designing a storage cabinet

Luis Marques, François Clautiaux, Aurélien Froger

## ► To cite this version:

Luis Marques, François Clautiaux, Aurélien Froger. Mathematical models based on decision hypergraphs for designing a storage cabinet. 2023. hal-04303041v1

**HAL Id: hal-04303041**

**<https://inria.hal.science/hal-04303041v1>**

Preprint submitted on 23 Nov 2023 (v1), last revised 10 Sep 2024 (v3)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Mathematical models based on decision hypergraphs for designing a storage cabinet

Luis Marques<sup>1</sup>

François Clautiaux<sup>1</sup>

Aurélien Froger<sup>1</sup>

<sup>1</sup>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, a set of compartments in each shelf and to select the items to be inserted 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-dimensional guillotine cutting problem for the design of the shelves and compartments with a set of temporal knapsack problems for the selection and assignment of 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 all 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 article, 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) items, which may come from historical or forecast data. We assume that the cabinet may be undersized for the items that are candidates for storage over time. We must therefore decide the selection of the items to be stored so as to maximize their profit contribution (a measure of their total utility) and, where appropriate, their allocation to a compartment. As the contents of the compartments are supposed to slide (or be pushed by a mechanism) forward when opened, the items must be the same width as the compartment in which they are stored. We aim to mathematically model and solve this problem.

Our problem is inspired by a real-life application involving inner-city pharmacies that use an automatic under-the-counter storage and retrieval cabinet for the most common drugs. A related application is in small retail stores, where a storage cabinet can be installed close to the counter and store the most popular products, while the others are stored in the backroom. Here we tackle a simplified version of such problems as a first step towards optimizing such systems.

Our problem shares some similarities with the *shelf dimensioning and product allocation problem* in retail stores studied in (Hübner et al., 2021). However, there are several notable differences. Our design decisions belong to a larger search space as they include shelves but also compartments on those shelves, whereas these decisions are related to the vertical levels of shelf segments in (Hübner et al., 2021). Once the design is decided, the remaining problem in (Hübner et al., 2021) is a *retail shelf space planning problem* (Bianchi-Aguiar et al., 2021). 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, which is not the case in our problem. The demand of a product depends on its available quantity and its position on the shelves, which is a significant difference compared to our work where the profit of an item is independent from its location within the cabinet. However, the placement of a product on the shelf and its sale over the time horizon is not explicitly accounted for in their model. A cost is due when a replenishment from the backroom is necessary due to high demand for a product. As in our problem, the objective is to maximize the total profit. Hübner et al. (2021) formulated their problem as an integer linear program and solved it with a commercial solver.

Mathematical methods for the design of (mainly automatic) *storage and retrieval systems* have been the subject of many research works over the years, mostly for warehouses. In Roodbergen and Vis (2009), the authors give an overview of some key design decisions for automatic storage and retrieval systems such as physical design, storage assignment, batching, dwell-point location, sequencing orders. In the warehouse context, physical design usually consists in deciding the location and layout of the storage racks, and storage assignment consists in deciding the racks to associate to each storage and retrieval request. These decisions are primarily determined in order to maximize the number of retrieval operations per minute because response time is generally more important than the space occupied by the system. Thus, the problem we study differs from this literature from an objective function perspective. Maximizing space utilization, although important in warehouse optimization (Gu et al., 2010), has been less studied. Two recent works (Cardona and Gue, 2019, 2020) optimize the layout of a unit-load warehouse considering a service level constraint and show space and cost savings when allowing slots with different heights in rack-bays.

From a combinatorial optimization point of view, we study a two-phase and three-dimensional variant of the temporal knapsack problem that we refer to as the *Storage Cabinet Physical Design* (SCPD) problem. In the first phase, called the *design phase*, a three-dimensional *storage cabinet* or *cabinet* for the rest of this paper, is horizontally divided into *shelves*, which in turn are vertically divided into *compartments*. The second phase, called the *assignment phase*, corresponds to the selection of requests to be satisfied, hereinafter referred to as *items*, and in this case, their allocation to compartments.

This problem generalizes several well-known hard combinatorial optimization problems. The design part of the problem is closely related to the *two-stage two-dimensional guillotine cutting problem* (2D-GCP) (Gilmore and Gomory, 1965). In this 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). A third dimension, representing the capacity of the compartments, is added to define the SCPD problem. In order for an item to be assigned to a compartment, it must satisfy the dimensional constraints and the temporal constraints, i.e. the item is present during a given time interval if it is assigned to a compartment. This is a generalization of the so-called *temporal knapsack problem* (TKP) (Caprara et al., 2013).

Two-dimensional packing problems have been considered in many papers (see Iori et al. (2021) for a survey). In Lodi and Monaci (2003) and Lodi et al. (2004), the authors studied a guillotine version of the two-dimensional knapsack and the two-dimensional bin packing problems, where they introduce the concept of *levels* to model the first-stage guillotine cut. The model studied decides which items *initialize* the levels and which items are assigned to the levels. 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. 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.

The TKP is more recent, but there exist several exact methods, based on integer programming and dynamic programming (Caprara et al., 2013; Bartlett et al., 2005; Gschwind and Irnich, 2017; Caprara et al., 2016; Clautiaux et al., 2021). Clautiaux et al. (2021) solved the TKP exactly using an iterative refinement method called *successive sublimation dynamic programming*. 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.

In this paper, we focus on mixed integer linear programming (MILP) formulations, and in particular on arc flow reformulations in *decision hypergraphs*. Arc flow formulations are increasingly popular in the field of cutting and packing problems since their first successful use (Valério de Carvalho, 1999). Such formulations are known to 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, develop reduction methods to restrict the solution space and develop 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 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. 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)).

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 the introduction of 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, 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 on randomly generated instances the advantages of solving the arc flow formulation over a compact MILP formulation. We first observe that the results obtained by solving the arc flow formulation without any graph reduction or valid inequalities are not competitive with those obtained by solving the compact formulation. By strengthening the linear relaxation of the arc flow formulation, reducing the size of the decision hypergraph and breaking certain symmetries, 25 more instances (out of 180) are optimally solved compared to the compact MILP formulation and the average gap is 4.38% versus 6.32%.

The rest of this paper is organized as follows. In section 2 we formally define the SCPD problem, describe

some of its properties, and present a compact MILP formulation. In section 3 we introduce an arc flow formulation of the SCPD problem. In section 4 we describe several improvements to the arc flow formulation that help to keep its size tractable. In section 5 we present how we generated our instances and the results of our computational experiments. In section 6, we offer a few brief concluding remarks.

## 2 Problem definition

In this section, we give a formal definition of the problem, state some of its properties, and formulate it as a compact MILP model.

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  $d_i \in \mathbb{N}^*$ , 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*  $\psi$  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*  $\phi$  as a three-dimensional rectangular object of size  $(h_\phi, w_\phi, L_{\max})$ , where  $h_\phi \leq H_{\max}$ ,  $w_\phi \leq W_{\max}$ .

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. In the problem, we restrict items to be assigned only to compartments with exactly 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.

For the sake of 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  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}, B)$  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  $\Psi$  is a set of shelves,  $\Phi$  a set of compartments,  $\mathcal{I}' \subseteq \mathcal{I}$  a subset of items,  $\mu : \Phi \rightarrow \Psi$  a mapping of the compartments to the shelves, and  $\rho : \mathcal{I}' \rightarrow \Phi$  a mapping of the selected items to the compartments, which satisfy 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  $\Psi$  is less than or equal to  $H_{\max}$ .

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

3. A compartment  $\phi$  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  $\psi$ , the sum of the widths of the compartments assigned to  $\psi$  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  $\phi$ , the items assigned to  $\phi$  have a height less than or equal to the height of  $\phi$  and a width equal to the width of  $\phi$ .

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

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

$$\forall t \in \mathcal{T}, \forall \phi \in \Phi, \sum_{i \in \mathcal{I}: \rho(i) = \phi, s_i \leq t < s_i + d_i} l_i \leq L_{\max}$$

Figure 1 and Figure 2 show an instance of the SCPD problem with four items and two time steps and a feasible solution. In this instance, items 1 and 3 are present only during time step 1, 2 and 4 are present at time steps 1 and 2, and items 5 and 6 are present at time step 2. The solution we present has one shelf with two compartments and each item of the instance has been assigned to a compartment. The gray parts represent wasted space.

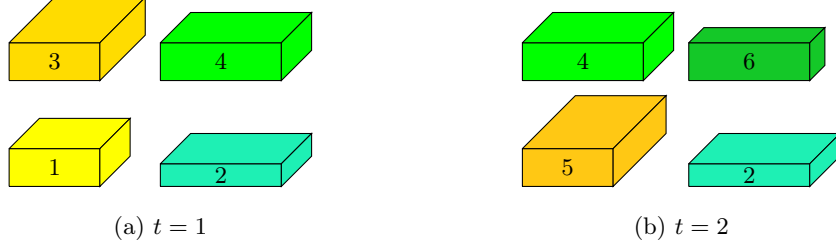


Figure 1: An instance with four items and two time steps



Figure 2: A feasible solution to the instance displayed in Figure 1

**Problem 2** (Storage Cabinet Physical Design optimization problem (**SCPD problem**)). Given  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ , 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  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}, B)$ .

We prove that the SCPD-Decision problem is NP-hard by polynomially reducing the Temporal Knapsack decision problem (Problem 3), known to be NP-hard in the strong sense (Bonsma et al., 2014), to it.

**Problem 3** (Temporal Knapsack decision problem (**TK-Decision problem**)). An instance of the Temporal Knapsack decision (TK-Decision) problem is a tuple  $(C, \mathcal{J}, \bar{B})$  where  $C \in \mathbb{N}$  is the capacity of the knapsack,  $\mathcal{J}$  is a set of items, and  $\bar{B} \in \mathbb{R}_+$ . Each item  $j \in \mathcal{J}$  has a weight  $\bar{w}_j \in \mathbb{N}$ , a profit  $\bar{p}_j \in \mathbb{R}_+$  and a time interval  $[\bar{s}_j, \bar{s}_j + \bar{d}_j)$  with  $\bar{s}_j \in \mathbb{N}$  and  $\bar{d}_j \in \mathbb{N}$ , during which, when selected, it is present in the knapsack. Given  $(C, \mathcal{J}, \bar{B})$ , 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  $C$ .

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

- (ii) The sum of the profits of the items assigned to the knapsack is at least  $\bar{B}$ .

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

**Proposition 1** (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  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}, B)$  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  $d_i = \overline{d}_j$ .  $\square$

Let us now discuss a restriction of the solution space. Dominance rule 1 states that it is never useful to build shelves that have a height greater than all the items it can contain. It is easy to see that, since items do not stack vertically, the space between the highest item a shelf can contain and the top of the shelf is wasted space in any solution.

**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.*

We now introduce a compact MILP model for the SCPD problem. Thanks to Dominance rule 1, we do not need to consider all possible shelf heights, and thus restrict ourselves to heights equal to those of the items. By definition, the width of a compartment is always the width of an item. In our formulation, the creation of shelves and compartments is modelled using representative items. 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 in 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. We define the sets  $\mathcal{I}_{\leq h} = \{i \in \mathcal{I} : h_i \leq h\}$ ,  $\mathcal{I}_{\leq h, w} = \{i \in \mathcal{I} : h_i \leq h, w_i = w\}$  and  $\mathcal{I}_{h, w} = \{i \in \mathcal{I} : h_i = h, w_i = w\}$  as the set of items with a height less than or equal to  $h$ , the set of items with a height less than or equal to  $h$  and a width equal to  $w$ , and the set of items with a height equal to  $h$  and a width equal to  $w$ , respectively. The MILP model is as follows:

$$\text{maximize} \quad \sum_{j \in \mathcal{I}} \sum_{k \in \mathcal{I}_{\leq h_j, w_j}} p_k x_{j,k} \quad (1a)$$

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

$$\sum_{j \in \mathcal{I}_{\leq h_i}} w_j y_{i,j} \leq W_{\max} \quad i \in \mathcal{I} \quad (1c)$$

$$\sum_{\substack{k \in \mathcal{I}_{\leq h_j, w_j} \\ s_k \leq t < s_k + d_k}} \ell_k x_{j,k} \leq L_{\max} \quad j \in \mathcal{I}, t \in \mathcal{T} \quad (1d)$$

$$\sum_{\substack{j \in \mathcal{I} \\ k \in \mathcal{I}_{\leq h_j, w_j}}} x_{j,k} \leq 1 \quad k \in \mathcal{I} \quad (1e)$$

$$x_{j,k} \leq x_{j,j} \quad j \in \mathcal{I}, k \in \mathcal{I}_{\leq h_j, w_j} \quad (1f)$$

$$x_{j,j} = \sum_{\substack{i \in \mathcal{I} \\ j \in \mathcal{I}_{\leq h_i}}} y_{i,j} \quad j \in \mathcal{I} \quad (1g)$$

$$y_{i,j} \leq z_i \quad i \in \mathcal{I}, j \in \mathcal{I}_{\leq h_i} \quad (1h)$$

$$y_{i,i} = z_i \quad i \in \mathcal{I} \quad (1i)$$

$$z_i \in \{0, 1\} \quad i \in \mathcal{I} \quad (1j)$$

$$y_{i,j} \in \{0, 1\} \quad i \in \mathcal{I}, j \in \mathcal{I}_{\leq h_i} \quad (1k)$$

$$x_{j,k} \in \{0, 1\} \quad j \in \mathcal{I}, k \in \mathcal{I}_{\leq h_j, w_j} \quad (1l)$$

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 in a shelf does not exceed the width of the cabinet. Constraints (1d) state that the sum of the lengths of the items in a compartment does not exceed the length of the cabinet with respect to the time steps at which the items are present. Note that it is sufficient to add this constraint 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 in a shelf only if that shelf is created and that the item representing the shelf represents one of the compartments built into the shelf. Constraints (1j)-(1l) require the variables to be binary.

### 3 An hypergraph-based reformulation

In this section, we propose 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 §3.1. This dynamic program defines a decision hypergraph (Martin et al., 1990) which we use in §3.2 as the network on which we derive an arc flow formulation of the SCPD problem.

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

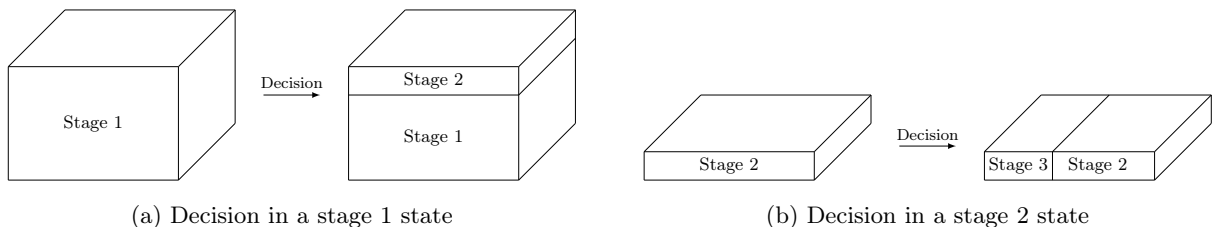


Figure 3: Illustration of the design decisions

We introduce our dynamic program in a bottom-up fashion, from stage 3 to stage 1. For each stage, we first describe the information needed to express a state. We then present a recursive formulation to compute the maximum profit that can be obtained from the current state by making decisions leading to the initial states of the dynamic program. We define  $\mathcal{H} = \{h : \exists i \in \mathcal{I}, h_i = h\}$  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. If no compression or reduction is applied, the former leads to fewer states. However, our recursive formulation for stage 3 is an adaptation of the latter, as it allows us to make better use of the preprocessing and symmetry breaking techniques described in Section 4.

It is an event-based formulation. 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  $\mathcal{E} = (0, 1, \dots, 2|\mathcal{I}|)$  be an ordered list of events where 0 represents a dummy event signalling 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  $(t(e_1) = t(e_2) \wedge e_1 \in \mathcal{E}^{\text{out}} \wedge e_2 \in \mathcal{E}^{\text{in}})$ . 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  $L \in \{0, \dots, L_{\max}\}$ , 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:

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

Five cases are considered in the above formula. The first case is the dummy event and the profit is set to 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 whether or not to insert  $i(e)$  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 of the current shelf and its width are needed to decide which compartments to build. A stage 2 state is thus denoted by  $(h, W)$ , where  $h \in \mathcal{H}$  and  $W \in \{0, \dots, W_{\max}\}$ . We denote by  $\alpha_2(h, W)$  the maximum profit that can be obtained from a stage 2 state  $(h, W)$  making decisions that lead to 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 into 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:

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

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  $H \in \{0, \dots, H_{\max}\}$ . We denote by  $\alpha_1(H)$  the maximum profit that can be obtained from a stage 1 state  $(H)$  making decisions that lead to 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:

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

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.

### 3.2 An 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 and consists of a set of vertices and a set of *decision hyperarcs* (hyperarcs in the remainder). In Martin et al. (1990), an 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 an hypergraph representation of a dynamic program, the vertices are 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  $H \in \{0, \dots, H_{\max}\}$  leads to an 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  $W \in \{0, \dots, W_{\max}\}$  leads to an 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 one carrying to 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 an hyperarc with the source as the tail. The decision hypergraph is built recursively from the sink to the source. Algorithm 1 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 modelled 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

---

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

---

**Input:** An SCPD instance  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$   
**Output:** The decision hypergraph associated with the dynamic program applied to the given instance

```

1  $\mathcal{V} \leftarrow \{v^+, (H_{\max})\}$ ; // set of vertices of the hypergraph
2  $\mathcal{A} \leftarrow \emptyset$ ; // set of hyperarcs
3  $\mathcal{S}^1 \leftarrow \{(H_{\max})\}$ ; // stage 1 states to process
4  $\mathcal{S}^2 \leftarrow \emptyset$ ; // stage 2 states to process
5  $\mathcal{S}^3 \leftarrow \emptyset$ ; // stage 3 states to process
6 while  $\mathcal{S}^1 \neq \emptyset$  do
7    $v = (H) \leftarrow$  element of  $\mathcal{S}^1$  that maximizes  $H$ ;
8    $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \setminus \{v\}$ ;
9   Apply Algorithm 2 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ ,  $v$ ,  $\mathcal{S}^1$ ,  $\mathcal{S}^2$ , and  $(\mathcal{V}, \mathcal{A})$  as inputs;
10 while  $\mathcal{S}^2 \neq \emptyset$  do
11    $v = (h, W) \leftarrow$  element of  $\mathcal{S}^2$  that maximizes lexicographically  $h$  and  $W$ ;
12    $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \setminus \{v\}$ ;
13   Apply Algorithm 3 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ ,  $v$ ,  $\mathcal{S}^2$ ,  $\mathcal{S}^3$ , and  $(\mathcal{V}, \mathcal{A})$  as inputs;
14 while  $\mathcal{S}^3 \neq \emptyset$  do
15    $v = (h, w, L, e, \mathbf{q}) \leftarrow$  element of  $\mathcal{S}^3$  that maximizes lexicographically  $h, w, e$ ;
16    $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \setminus \{v\}$ ;
17   Apply Algorithm 4 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ ,  $v$ ,  $\mathcal{S}^3$  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  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ , a stage 1 state  $v = (H)$ , two sets  $\mathcal{S}^1$  and  $\mathcal{S}^2$ , an hypergraph  $(\mathcal{V}, \mathcal{A})$   
**Result:** Creates the hyperarcs incoming to  $v$ , updates  $\mathcal{V}, \mathcal{A}, \mathcal{S}^1, \mathcal{S}^2$

```

1 for  $h \in \mathcal{H}, h \leq H$  do
2    $v_1 \leftarrow (h, W_{\max})$ ; // generate a stage 2 state
3    $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_1\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}, \mathcal{F} \leftarrow \{v_1\}$ ;
4   if  $\exists i \in \mathcal{I}, h_i \leq H - h$  then
5      $v_2 \leftarrow (H - h)$ ; // generate a stage 1 state
6      $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \cup \{v_2\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}, \mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$ ;
7    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$ ; // create the hyperarc

```

---

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

---

**Input:** An SCPD instance  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ , a stage 2 state  $v = (h, W)$ , two sets  $\mathcal{S}^2$  and  $\mathcal{S}^3$ , an hypergraph  $(\mathcal{V}, \mathcal{A})$   
**Result:** Creates the hyperarcs incoming to  $v$ , updates  $\mathcal{V}, \mathcal{A}, \mathcal{S}^2, \mathcal{S}^3$

```

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

```

---

---

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

---

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

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

```

1 if  $e = 0$  then
2    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v^+\}, 0)\};$  // connect the source with this state
3 else if  $e \in \mathcal{E}^{out}$  then
4    $v_1 \leftarrow (h, w, L, E_{h,w}^-(e), \mathbf{q});$  // skipping the item
5    $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\};$ 
6    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\};$  // create the hyperarc
7   if  $L \geq \ell_{i(e)}$  then
8      $v_2 \leftarrow (h, w, L - \ell_{i(e)}, E_{h,w}^-(e), \mathbf{q} + \varepsilon_{i(e)});$  // selecting the item
9      $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_2\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\};$ 
10     $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_2\}, p_{i(e)})\};$  // create the hyperarc
11 else if  $e \in \mathcal{E}^{in}$  then
12   if  $\mathbf{q}_{i(e)} = 0$  then
13      $v_1 \leftarrow (h, w, L, E_{h,w}^-(e), \mathbf{q});$  // skipping the item
14   else
15      $v_1 \leftarrow (h, w, L + \ell_{i(e)}, E_{h,w}^-(e), \mathbf{q} - \varepsilon_{i(e)});$  // selecting the item
16      $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\};$ 
17      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\};$  // create the hyperarc

```

---

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:

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

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

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

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

$$x_a \in \mathbb{N} \quad a \in \mathcal{A} \quad (5e)$$

The objective function in (5a) is the total profit obtained 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 an 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 §S.1 of the supplementary material). To handle the case where a flow greater than one exits a vertex/state, we store pairs consisting of a stage 2 state and a shelf in  $\mathcal{L}^2$ , and pairs consisting of a stage 3 state and a compartment in  $\mathcal{L}^3$ . 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 1 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 decisions to select an item to leave a compartment must be taken before the decisions to select the item to enter into it. To avoid overloading the figure, if the profit associated with an hyperarc is 0, then 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.

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

$i$	$h_i$	$w_i$	$\ell_i$	$p_i$	$s_i$	$d_i$
1	5	2	2	1	0	2
2	9	2	1	2	1	2

Table 1: An SCPD instance with two items

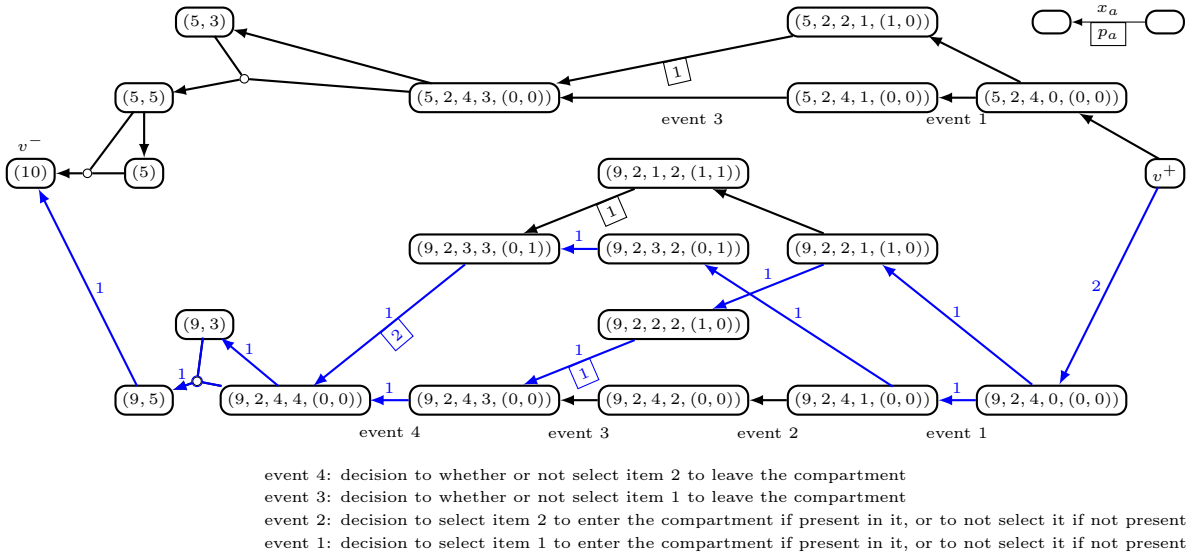


Figure 4: Decision hypergraph of the instance given in Table 1 and an optimal solution

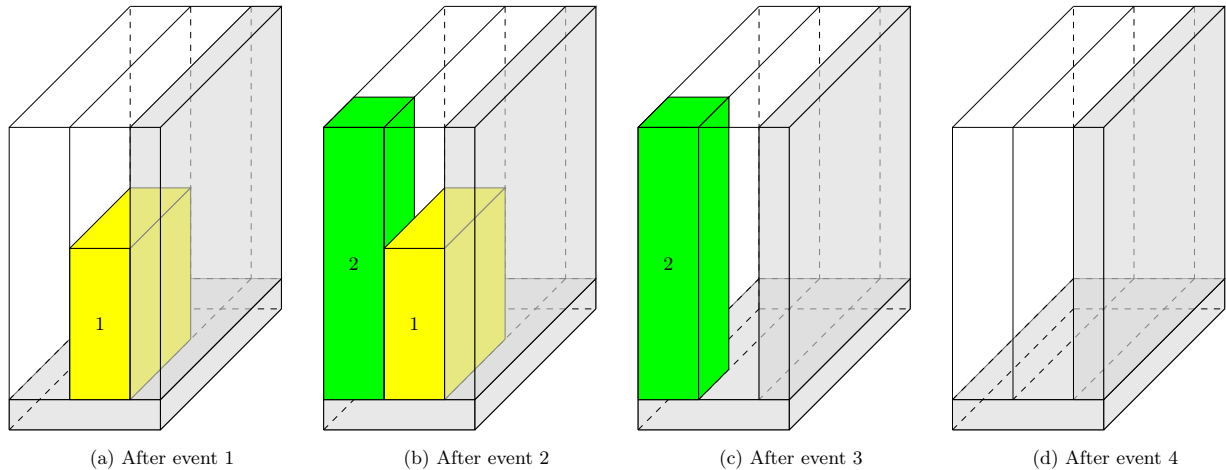


Figure 5: Solution associated with the flow in blue in Figure 1

## 4 Improvements of the hypergraph-based formulation

The aim of this section is to describe improvements on the arc flow formulation (5). To strengthen the quality of its linear relaxation, we start by introducing valid inequalities in §4.1. Since the formulation is non-polynomial in the size of the instance, we describe in §4.2 to §4.4 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 §S.2 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 4 (Bin Packing (BP)).** *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  $\bar{w}_j \in \mathbb{N}$ . 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 5 (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  $\bar{w}_j \in \mathbb{N}$  and a time interval  $[\bar{s}_j, \bar{s}_j + \bar{d}_j)$  with  $\bar{s}_j \in \mathbb{N}$  and  $\bar{d}_j \in \mathbb{N}$ , during which the item is present. The TBP problem is to find an assignment of the items to the bin 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.*

We note  $z_{\text{BP}(\mathcal{J}, C)}^*$  (resp.  $z_{\text{TBP}(\mathcal{J}, C)}^*$ ) the value of an optimal solution to the BP (resp. TBP) problem with capacity  $C$  and set of items  $\mathcal{J}$ . We recall that the sets  $\mathcal{I}_{\leq h}$ ,  $\mathcal{I}_{\leq h, w}$  and  $\mathcal{I}_{h, w}$  are the set of items with a height less than or equal to  $h$ , the set of items with a height less than or equal to  $h$  and a width equal to  $w$ , and the set of items with a height 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  $z_{\text{TBP}(\mathcal{J}, C)}^*$  by denoting as  $z_{\text{TBP}(\mathcal{I}_{\leq h, w}, L_{\max})}^*$  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  $\ell_i$  and time interval  $[s_i, s_i + d_i)$ .

### 4.1 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.

We want to address a problem that arises when a specific structure is met in the solution of the linear relaxation to (5). We illustrate this situation in Figure 6. We consider an instance with a cabinet of size (2, 4, 1) and a single item with size (2, 1, 1). In this figure, there are 0.25 units of flow passing through the leftmost hyperarc, corresponding to the creation of a shelf with height 2. Then, four compartments with identical width equal to 1 are built inside this shelf and, due to flow conservation, there are 0.25 units of flow going through the corresponding rightmost hyperarcs. Then there is 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 inside 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.

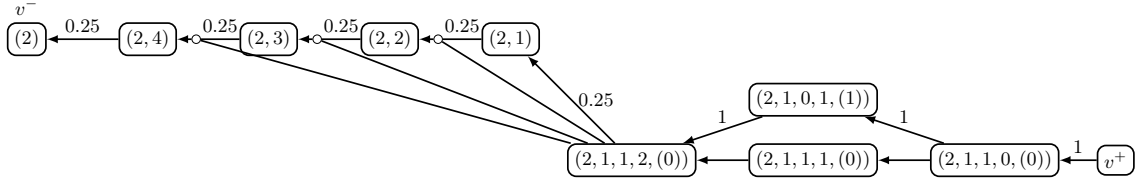


Figure 6: Example of a solution to the linear relaxation of model (5)

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

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

*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 positivity of the variables. If  $\sum_{a \in \mathcal{A}(i)} x_a = 1$  (i.e., item  $i$  is selected), then there must be an 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.  $\square$

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 there is only 0.25 units of flow traversing the single hyperarc associated with the design of a shelf with height greater than or equal to the height of the item.

## 4.2 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). 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, and then 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 the 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 an hyperarc corresponding to the creation of a shelf of height  $h$  must be traversed to reach the sink  $v^-$ . Denoting  $h(v)$  the height of the stage 1 state represented by  $v$ , this value can be computed in a recursive way as follows:

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

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.

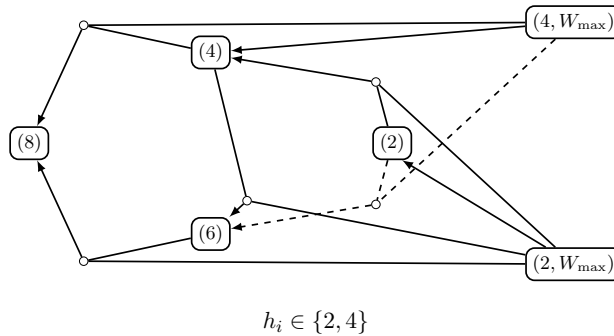


Figure 7: Illustration of how to exploit dominance rule 2 in the hypergraph generation

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  $z_{\text{BP}(\mathcal{L}, L_{\text{max}})}^*$  where  $\mathcal{L}$  contains  $z_{\text{TBP}(\mathcal{I}_{\leq h, w}, L_{\text{max}})}^*$  items with weight  $w$  for each width  $w \in \mathcal{W}_h$ . The validity of this procedure comes from the fact that items assigned to the same compartment must 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” 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  $h \in \mathcal{H}$  and  $U_1(h) = \min \left\{ \lfloor \frac{H_{\text{max}}}{h} \rfloor, N_1(\mathcal{I}_{\leq h}), \sum_{w \in \mathcal{W}_h} \min \left\{ z_{\text{TBP}(\mathcal{I}_{\leq h, w}, L_{\text{max}})}^*, |\mathcal{I}_{h, w}| \right\} \right\}$ . 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  $H_{\text{max}}$  cannot contain more than  $\lfloor H_{\text{max}}/h \rfloor$  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  $k \in \mathbb{N}^*$ . 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 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  $h$ .  $\square$

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 within 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 the dominance rule 3 to compartments built inside 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  $U_2(h, w) = \min\{\lfloor \frac{W_{\max}}{w} \rfloor, N_2(\mathcal{I}_{\leq h, w})\}$ . There exists an optimal solution to the SCPD problem where in 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 in the shelf, hence  $U_2(h, w) \leq \lfloor \frac{W_{\max}}{w} \rfloor$ . 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.  $\square$

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 an 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:

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

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 height equal to  $h$ .*

*Proof.* Let  $S^*$  be an optimal solution to the SCPD problem and let  $\psi$  be a shelf in  $S^*$ . The shelf  $\psi$  can be transformed in two steps into a shelf where the leftmost compartment contains an item with height equal to  $h_\psi$ . Since the compartments in  $\psi$  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  $\psi$ . If  $\tilde{h} < h$ , then the height of shelf  $\psi$  can be reduced to  $\tilde{h}$ .  $\square$

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.

### 4.3 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 clearly 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), originally 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 3 indicates that merging vertices associated with equivalent states does not change the optimal value of the arc flow formulation (5).

**Proposition 3.** *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 items selection are added by this procedure.  $\square$

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

Proposition 4 is inspired by a classical technique used in packing problems, and is related to the notion of *raster points* (Terno et al., 1987).

**Proposition 4.** *Let  $H_1, H_2 \in \{0, \dots, H_{\max}\}$ . 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  $\eta(v_1) = \eta(v_2)$ .  $\square$

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 which is 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 4 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 5.

**Proposition 5.** *Let  $h \in \mathcal{H}$  and  $W_1, W_2 \in \{0, \dots, W_{\max}\}$ . 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  $\eta(v_1) = \eta(v_2)$ .  $\square$

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.

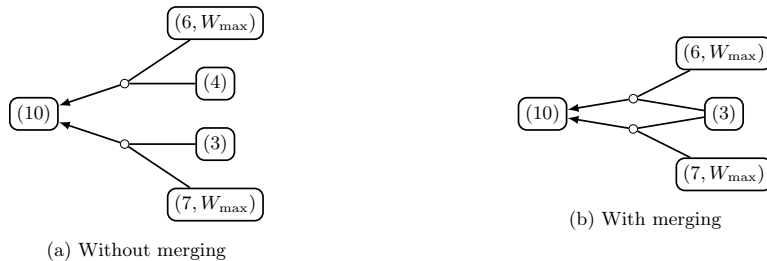


Figure 8: Example of equivalent stage 2 states

Proposition 6 gives a sufficient condition for the equivalence of stage 3 states.

**Proposition 6.** *Let  $h_1, h_2 \in \mathcal{H}$ ,  $w \in \mathcal{W}$ ,  $L \in \{0, \dots, L_{\max}\}$ ,  $e_1, e_2 \in \mathcal{E}$ , 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 the set of items that can enter or leave the compartment if considering the events in descending order are exactly the same from  $e_1$  as from  $e_2$ . Therefore,  $\eta(v_1) = \eta(v_2)$ .  $\square$

We use Proposition 6 to merge vertices associated with equivalent stage 3 states. For each event  $e \in \mathcal{E}$ , we compute  $H^-(e, h) = \max \{h_{i(\tilde{e})} : \tilde{e} \in \mathcal{E}, \tilde{e} \leq e, h_{i(\tilde{e})} \leq h, w_{i(\tilde{e})} = w_{i(e)}\}$  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.

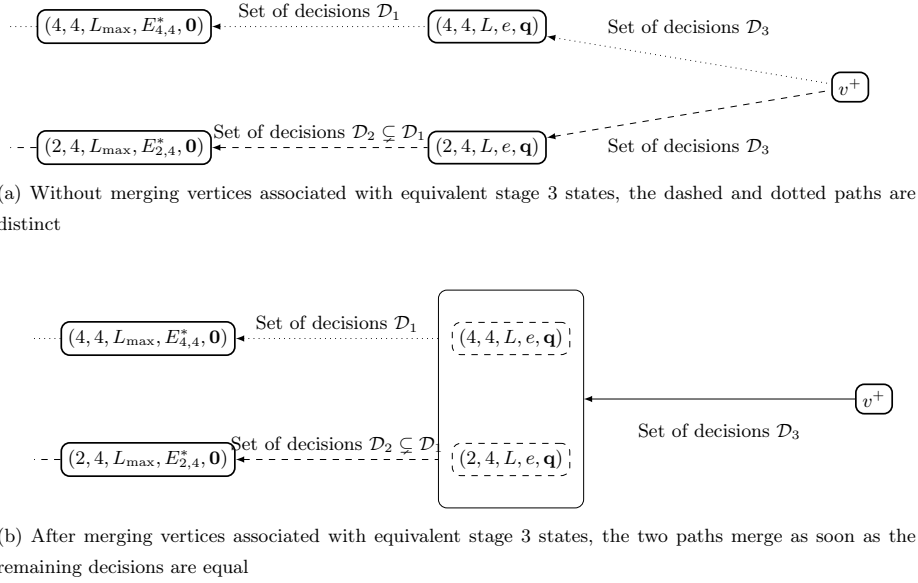


Figure 9: Example of equivalent stage 3 states

#### 4.4 Exploiting trivial subproblems

In the SCPD problem, there may exist states that 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 into 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  $(W)$  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  $\eta(v) = \mathbb{P}(\{i \in \mathcal{I}_{\leq h, w} : \exists \tilde{e} \in \mathcal{E}^{out}, \tilde{e} \leq e, i(\tilde{e}) = i\})$

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 in the hypergraph 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, as 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  $I_{\leq h,w}$  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  $I_w[k]$  (resp.  $I_{\leq h,w}[k]$ ) be the item at index  $k$  in  $I_w$  (resp.  $I_{\leq h,w}$ ).

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

$$\beta^\ddagger(h, w, k) = \begin{cases} 0 & \text{if } k = 0 \\ \max \{p_{I_{\leq h,w}[k]} + \beta^\ddagger(h_{I_{\leq h,w}[k]}, w, k-1), \beta^\ddagger(h_{I_{\leq h,w}[k]}, w, k-1)\} & \text{otherwise} \end{cases} \quad (7)$$

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 supproblem, 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  $I_{\leq h,w}$  related to the first outgoing event occurring before or exactly at  $e$ , i.e.  $k_e = \max\{k \in \{1, \dots, |I_{\leq h,w}|\} : e^{\text{out}}(I_{\leq h,w}[k]) \leq e\}$  where  $e^{\text{out}}(i)$  is the outgoing event of an item  $i \in \mathcal{I}$ . Then we do the following substitution:

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

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  $k \in \{1, \dots, |I_{\leq h,w}|\}$ .

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  $k \in \{0, \dots, |I_w|\}$ . The forward recursion is as follows:

$$\beta^\dagger(w, k) = \begin{cases} 0 & \text{if } k = 0 \\ \max \{p_{I_w[k]} + \beta^\dagger(w, k-1), \beta^\dagger(w, k-1)\} & \text{otherwise} \end{cases} \quad (9)$$

For each height  $H \in \{0, \dots, H_{\max}\}$  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.  $k_w(H) = \max\{k \in \{0, \dots, |I_w|\} : h_{I_w[k]} \leq H\}$ . If a stage 2 state  $(h, W)$  or a stage 1 state  $(H)$  is found to be trivial, we make the following substitutions:

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

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

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  $k \in \{1, \dots, |I_w|\}$ .

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.

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

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 exists 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.  $\square$

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.

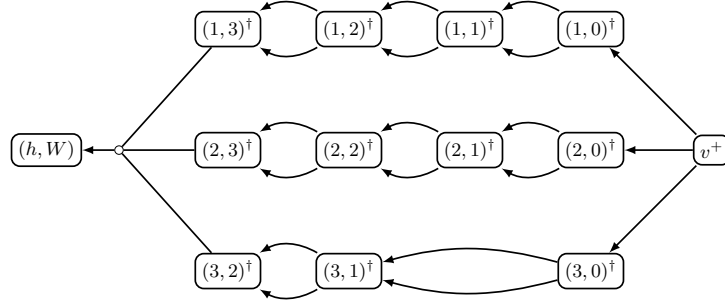


Figure 10: Illustration of the hyperarcs created from a trivial stage 2 state  $(h, W)$

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

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

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

This equation is derived by separating a stage 1 state into two cases. Either a shelf with a height greater than  $\bar{h}$  is built, in which case the recursion does not change from the initial one, or a shelf of height less than or equal to  $\bar{h}$  is built, in which case, by Dominance rule 2 only shelves of height  $\bar{h}$  or less will be subsequently used. Since by assumption the problem is trivial by using only shelves of height  $\bar{h}$  or less, the right-hand part of the equation, which allows to select any subset of items of height less than  $\bar{h}$ , 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  $\bar{h}$ -trivial with  $H = 4$  and  $\bar{h} = 2$ . In this example, the items have a height in the set  $\{1, 2, 3\}$ , a width in the set  $\{1, \dots, 7\}$  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.

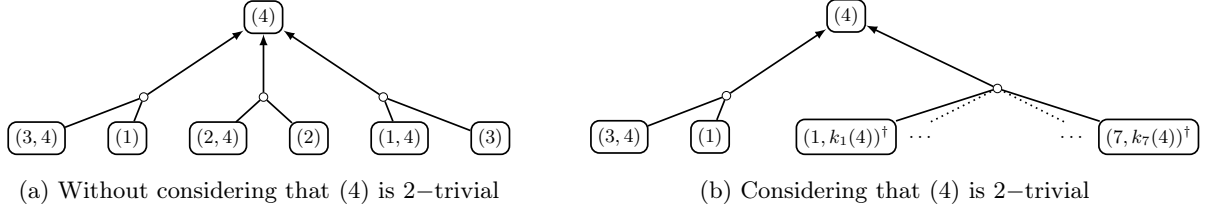


Figure 11: Example of  $h$ -trivial stage 1 problems reduction

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.

**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 7. 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 7.** *A stage 3 state  $v = (h, w, L, e, \mathbf{q})$  is trivial if and only if the following conditions hold:*

$$\forall \tilde{e} \in \mathcal{E}_{h, w}, \tilde{e} < e, \quad L + \sum_{\hat{e}=\tilde{e}}^e \tilde{\ell}(\hat{e}, \mathbf{q}) \geq 0 \quad (14)$$

*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 the condition (14) holds.  $\square$

We determine whether a stage 2 problem is trivial by computing an upper bound on the minimum width required to select each admissible item in 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 8.** *A stage 2 state  $v = (h, W)$  is trivial if the following condition holds:*

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

*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 in the shelf.  $\square$

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  $h$ . Solving the BP and the TBP problems to optimality is not required to compute a valid upper bound using this procedure.

**Proposition 9.** *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.

## 5 Computational experiments

In this section, we first explain in §5.1 how we generated instances to the SCPD problem. We then describe in §5.2 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 §4.1, §4.2, §4.3 and §4.4 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 4.

### 5.1 Data sets

Our experiments are conducted using randomly generated instances<sup>1</sup>. For each instance, we set  $(H_{\max}, W_{\max}, L_{\max}) = (300, 300, 300)$ , and for each item, the height, width, length and profit are all uncorrelated and uniformly randomly distributed in the interval  $[50, 100]$ . To ensure that the compartments represent relevant problems, the number of different widths is set equal to  $m \in \mathbb{N}^*$  and the generation algorithms are adapted to this property. For the time intervals, we separate the instances into two classes, using the same methodology as Caprara et al. (2013).

For the first class of instances, denoted U, we generate the starts of time intervals uniformly at random in the interval  $[0, s_{\max}]$  and the durations uniformly at random in the interval  $[d_{\min}, d_{\max}]$ . Algorithm 12 in §S.3 of the supplementary material summarizes the generation algorithm for the instances of class U.

For the second 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 overlapping intervals. In the clique-based framework, we consider each width independently of the others. For each width, we generate C cliques, where each time step corresponds to a different clique of items. The generation algorithm 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 removes  $b\%$  of the items, (i.e. selects  $b\%$  items from the previous clique and sets  $d_i = t - s_i$  for these items), and adds a new items (i.e. items such that  $s_i = t$ ), to form the clique of time step  $t$ . The parameters  $b$  and  $a$  are uniformly randomly drawn in  $[b_{\min}, b_{\max}]$  and  $[a_{\min}, a_{\max}]$ , respectively. The algorithm stops when the last clique has been generated. Algorithm 13 in §S.3 of the supplementary material summarizes the generation algorithm for the instances of class I.

<sup>1</sup>available online from <https://gitlab.inria.fr/edge/scpd/scpd-artifacts>



We now define the different instance groups we created and explain the relevance of the chosen generation parameters. For each size of the input, i.e. the number of items, we create three instance groups where the only varying parameter is the number of different widths. The values we have chosen for  $s_{\max}$ ,  $d_{\min}$ ,  $d_{\max}$ ,  $a_{\min}$ ,  $a_{\max}$ ,  $b_{\min}$  and  $b_{\max}$  ensure that there is no instance where there is at most one item per width at each time step, i.e. the instance only requires solving the design phase. Finally, the parameter  $m$  plays an important role in the structure of our instances, with a lower value usually meaning an easier design phase and a more difficult assignment phase. Therefore, we use different values depending on the instance group to study the impact it has on the difficulty of our instances. Tables 2a and 2b summarize the generation parameters for the instances of classes U and I, respectively. In these tables, the column  $|\mathcal{I}|$ , which is the mean number of items per instance in the group, is given as an additional information and does not impact the instances generation. We generate 10 instances per instance group for a total of 180 instances.

Group	$ \mathcal{I} $	$s_{\max}$	$d_{\min}$	$d_{\max}$	$m$
U70-3	70	1000	100	500	3
U70-5	70	1000	100	500	5
U70-10	70	1000	100	500	10
U100-3	100	1000	100	500	3
U100-5	100	1000	100	500	5
U100-10	100	1000	100	500	10
U200-3	200	1000	100	500	3
U200-5	200	1000	100	500	5
U200-10	200	1000	100	500	10

(a) Class U

Group	$ \mathcal{C} $	$ \mathcal{I} $	$a_{\min}$	$a_{\max}$	$b_{\min}$	$b_{\max}$	$m$
I5-3	5	38	2	4	90	90	3
I5-5	5	63	2	4	90	90	5
I5-10	5	125	2	4	90	90	10
I10-3	10	72	2	4	90	90	3
I10-5	10	126	2	4	90	90	5
I10-10	10	253	2	4	90	90	10
I15-3	15	114	2	4	90	90	3
I15-5	15	191	2	4	90	90	5
I15-10	15	375	2	4	90	90	10

(b) Class I

Table 2: Generation parameters for the instances

To ensure that the clique-based instances in the same group generated by a probabilistic method do not unexpectedly differ in the number of items, we generate many instances but keep only a subset of them, chosen randomly so that the difference in the number of items of the minimum and maximum size instances is less than or equal to 5.

## 5.2 Numerical results

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. We solved the compact formulation of the TBP and BP problems (see Dell’Amico et al. (2020)) associated with the improvements described in §4.2 and §4.4 with CPLEX 20.1 with default parameters and a time limit of 10 seconds. The detailed results for every single tested instance are available online<sup>2</sup>.

In the remainder, we note  $\mathbf{C}$  the compact formulation (1),  $\mathbf{AF}$  the arc flow formulation (5) and  $\mathbf{AF}^{\mathbf{+ALL}}$  the arc flow formulation (5) with the improvements described in section 4. We consider an instance to be solved if the solver returns a proven optimal solution.

<sup>2</sup><https://gitlab.inria.fr/edge/scpd/scpd-artifacts>

### 5.2.1 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 **AF<sup>+ALL</sup>**. For this purpose, we classify the improvements into four families, each family corresponding to one of the subsections §4.1 (valid inequalities, **VI**), §4.2 (pruning hyperarcs, **PH**), §4.3 (merging equivalent states, **MES**) and §4.4 (trivial subproblems, **TS**). First, considering the formulation **AF**, for each family of improvements, we consider all the improvements listed in the corresponding subsection, creating the formulations **AF<sup>+VI</sup>**, **AF<sup>+PH</sup>**, **AF<sup>+MES</sup>** and **AF<sup>+TS</sup>**, respectively. Second, considering the formulation **AF<sup>+ALL</sup>**, for each family of improvements, we remove the improvements listed in the corresponding subsection, creating the configurations **AF<sup>+ALL-VI</sup>**, **AF<sup>+ALL-PH</sup>**, **AF<sup>+ALL-MES</sup>** and **AF<sup>+ALL-TS</sup>**, respectively. For each configuration, we report in Table 3 the size of the model and the number of instances solved. Note that, due to memory constraints, the formulation **AF** could not be created for 9 instances of the group U200-3 and for 1 instance of the group I15-10 (the largest formulation created contains around 12 millions of variables and 11 millions of constraints).

Formulation	Variables	Constraints	Solved	Difference
<b>AF</b>	1.0	1.0	49	—
<b>AF<sup>+VI</sup></b>	1.0	1.0	47	-2
<b>AF<sup>+PH</sup></b>	1.0	1.0	58	+9
<b>AF<sup>+MES</sup></b>	0.4	0.4	58	+9
<b>AF<sup>+TS</sup></b>	0.9	0.9	56	+7

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**.

Formulation	Variables	Constraints	Solved	Difference
<b>AF<sup>+ALL</sup></b>	1.0	1.0	84	—
<b>AF<sup>+ALL-VI</sup></b>	1.0	1.0	81	-3
<b>AF<sup>+ALL-PH</sup></b>	1.0	1.0	72	-12
<b>AF<sup>+ALL-MES</sup></b>	2.3	2.3	60	-24
<b>AF<sup>+ALL-TS</sup></b>	1.1	1.0	66	-18

Variables/Constraints: The average ratio between the number of variables and the number of constraints of the formulation and the formulation **AF<sup>+ALL</sup>**.

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

Difference: The difference in the number of instances solved between the formulation and the formulation **AF<sup>+ALL</sup>**.

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

Table 3: Impact of the improvements on the size and the number of the instances solved using the arc flow formulation

When considered independently, merging equivalent states (**MES**) has the highest added value, followed by using dominance rules (**PH**) to prune hyperarcs and exploiting trivial subproblems (**TS**). This is highly correlated with reducing the size of the model. If only valid inequalities are introduced in the formulation **AF**, they have a negative effect on the results obtained with the solver. However, removing these inequalities from the formulation **AF<sup>+ALL</sup>** results in three less instances being solved. To conclude this analysis, the best configuration is obtained by using all the improvements described in this paper. Note that the number of variables and constraints in the formulation **AF<sup>+ALL</sup>** is reduced by about 59% on average compared to the formulation **AF**.

### 5.2.2 Comparing the results obtained with the compact and the arc flow formulations

We now compare in Table 4, Table 5, and Figure 12, the arc flow formulations to the formulation **C** in terms of integrality gap, number of instances solved, and number of best primal and dual bounds found.

We first study the linear relaxations of the different models. We consider only the instances for which the linear relaxation of each formulation was solved in less than 70 hours using CPLEX 20.1 and for which the optimal value is known (i.e., the instance has been solved during at least one of our tests). Table 4 reports for each formulation the average integrality gap over each instance group. The column “Number of instances”

reports the number of instances considered in the comparison. In this table, the formulation  $\mathbf{AF}^{+\mathbf{VI}}$  denotes the arc flow formulation with the valid inequalities of Subsection 4.1. 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} $	Number of instances	Integrality gap			
			$\mathbf{C}$	$\mathbf{AF}$	$\mathbf{AF}^{+\mathbf{VI}}$	$\mathbf{AF}^{+\mathbf{ALL}}$
I5-3	38	10	0	0	0	0
I5-5	63	7	1.13	1.13	1.13	1.13
I5-10	125	9	26.25	6.19	6.17	5.94
I10-3	72	4	0.70	0.70	0.70	0.70
I10-5	126	0	—	—	—	—
I10-10	253	3	46.48	3.61	3.61	3.55
I15-3	114	0	—	—	—	—
I15-5	191	0	—	—	—	—
I15-10	375	0	—	—	—	—
U70-3	70	10	0	0	0	0
U70-5	70	9	0.35	0.35	0.35	0.35
U70-10	70	10	3.34	2.92	2.92	2.73
U100-3	100	3	0	0	0	0
U100-5	100	5	6.97	5.26	5.20	4.95
U100-10	100	9	11.22	4.93	4.92	4.70
U200-3	200	0	—	—	—	—
U200-5	200	0	—	—	—	—
U200-10	200	5	32.52	3.98	3.96	3.79
Mean	—	84	8.59	2.38	2.37	2.28

Table 4: Integrality gap of the formulations

We observe that the arc flow formulations have a smaller integrality gap than the compact formulation. The valid inequalities we add help to improve the linear relaxation of the formulation  $\mathbf{AF}$ . Specifically, the formulation  $\mathbf{AF}^{+\mathbf{VI}}$  has a better linear relaxation than the formulation  $\mathbf{AF}$  for 36 out of the 180 instances. The relative difference between the integrality gap obtained with and without the valid inequalities is at most 0.3%. We observe that the other improvements of Section 4 further improve the linear relaxation of the formulation  $\mathbf{AF}$ . For 39 instances, the integrality gap is equal to 0. In our experiments, this corresponds to instances where all the items are selected in the solution. These instances all have at most 100 items.

We now examine for which formulation the solver returns the largest number of proven optimal solutions or gives the best solutions among the formulations  $\mathbf{C}$  and  $\mathbf{AF}^{+\mathbf{ALL}}$ . 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 5 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	$\mathcal{I}$	<b>C</b>			<b>AF+ALL</b>		
		Primal best	Dual best	Solved	Primal best	Dual best	Solved
I5-3	38	10 /10	10 /10	10	10 /10	10 /10	10
I5-5	63	10 /10	5 /10	4	10 /10	10 /10	7
I5-10	125	8 /10	0 /10	0	10 /10	10 /10	9
I10-3	72	8 /10	5 /10	3	10 /10	10 /10	4
I10-5	126	6 /10	0 /10	0	7 /10	10 /10	0
I10-10	253	8 /10	0 /10	0	9 /10	10 /10	3
I15-3	114	7 /10	3 /10	0	2 /10	9 /10	0
I15-5	191	9 /10	0 /10	0	1 /10	10 /10	0
I15-10	375	3 /10	0 /10	0	7 /10	10 /10	0
U70-3	70	10 /10	10 /10	10	10 /10	10 /10	10
U70-5	70	10 /10	10 /10	9	10 /10	10 /10	9
U70-10	70	10 /10	10 /10	10	10 /10	10 /10	10
U100-3	100	9 /10	8 /10	3	4 /10	10 /10	3
U100-5	100	10 /10	3 /10	0	10 /10	10 /10	5
U100-10	100	10 /10	9 /10	9	10 /10	10 /10	9
U200-3	200	10 /10	7 /10	0	0 /3	3 /3	0
U200-5	200	3 /10	0 /10	0	7 /10	10 /10	0
U200-10	200	10 /10	1 /10	1	10 /10	9 /10	5
Total	—	151 /180	81 /180	59	137 /173	171 /173	84

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 5: Results obtained by the solver within 3600 seconds for the formulations **C** and **AF+ALL**

We observe that 25 more instances are solved with the formulation **AF+ALL** than with the formulation **C**. Note that every instance solved with the formulation **C** is also solved with the formulation **AF+ALL**. Of the 84 solved instances, the 39 instances where all the items are selected are all solved in less than 7 seconds with the formulation **AF+ALL** and in less than 66 seconds with the formulation **C**. The solver finds 90 better dual solutions with the formulation **AF+ALL** than with the formulation **C**. However, it returns 14 fewer best primal solutions with the former than with the latter and no feasible solution is found with the former within the time limit for seven large instances of group U200-3. This comes from the fact that the formulation **AF+ALL** has a comparable number of constraints to the formulation **C**, but has more than 25 times as many variables as the latter. For the 39 instances (resp. 28) instances where the solver returns a better primal solution with formulation **C** (resp. **AF+ALL**) than with the formulation **AF+ALL** (resp. **C**), the gap between the value of the primal solution obtained by the two formulations is on average equal to 4.21% (resp. 0.92%). 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+ALL**) and  $z$  is the primal bound for the formulation **AF+ALL** (resp. **C**).

We finally examine the solution time and the gap obtained for the formulations **C**, **AF** and **AF+ALL**. Figure 12a shows for each formulation the number of instances solved as a function of time. We observe that as many instances are solved in 10 minutes with the formulation **C** as with the formulation **AF** in 60 minutes.

More interestingly, we observe that as many instances are solved in 10 minutes with the formulation **AF+ALL** as with the formulation **C** in 60 minutes, which confirms the superiority of the formulation **AF+ALL**. 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+ALL**, with the exception of the seven instances with no feasible solution computed within the time limit. The average optimality gap is equal to 6.32%, 9.98% and 4.38% for formulations **C**, **AF** and **AF+ALL**, respectively. For the 25 instances solved with the formulation **AF+ALL** but not with the formulation **C**, the average optimality gap returned by the solver for the latter is 8.78%. All the results in this section indicate that on the instances we have generated, the results obtained with the formulation **AF+ALL** are significantly better than those obtained with the formulation **C**.

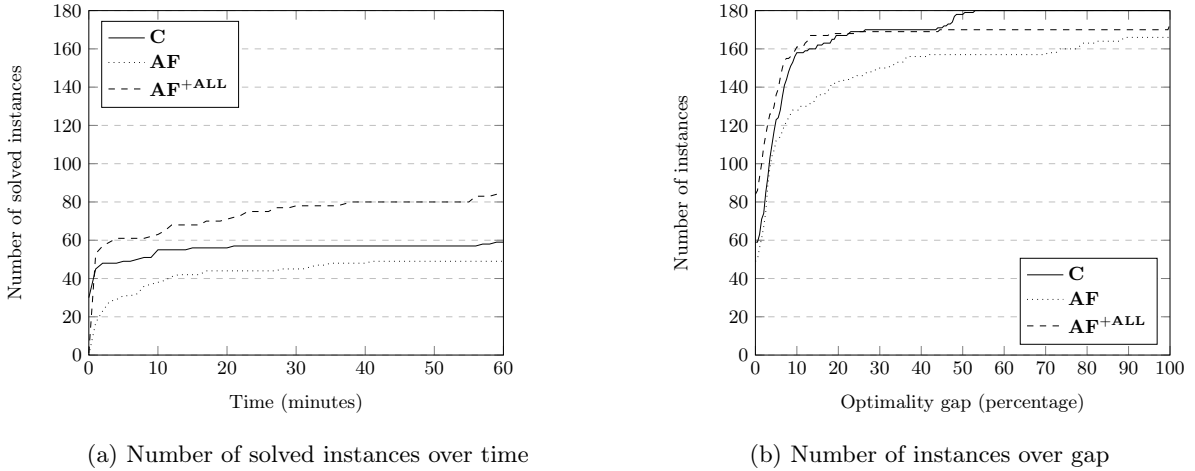


Figure 12: Performance profiles for the three MILP formulations

## 6 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 this arc flow formulation as it is does not lead to any additional solved instances compared to using a compact formulation, although its linear relaxation is of better quality. We presented several improvements to reduce the size of the hypergraph while maintaining the validity of the arc flow formulation. With the introduced refinements, we manage to reduce the hypergraph to approximately 41 % of its original size. The arc flow formulation based on this reduced hypergraph and containing valid inequalities produces significantly better results than those obtained with the original hypergraph, solving 35 more instances and providing better primal and dual bounds. Compared to the compact formulation, 25 more instances are solved using the arc flow formulation on the reduced hypergraph and with valid inequalities, and the average gap is reduced from 6.32% to 4.38%.

A possible way for solving more instances of the SCPD problem with our arc flow formulation is to give a primal solution computed by problem-specific heuristics to the solver. Generalizing our filtering techniques to problems than can be formulated using decision hypergraphs is also of particular interest.

## 7 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 Hoes, 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.
- 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. (2019). How to determine slot sizes in a unit-load warehouse. *IIE transactions*, 51(4):355–367.
- 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.
- 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.
- Gilmore, P. C. and Gomory, R. E. (1965). Multistage cutting stock problems of two and more dimensions. *Operations research*, 13(1):94–120.
- Gschwind, T. and Irnich, S. (2017). Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities. *OR Spectrum*, 39(2):541–556.
- Gu, J., Goetschalckx, M., and McGinnis, L. F. (2010). Research on warehouse design and performance evaluation: A comprehensive review. *European journal of operational research*, 203(3):539–549.
- 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.
- Iori, M., De Lima, V. L., Martello, S., Miyazawa, F. K., and Monaci, M. (2021). Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research*, 289(2):399–415.
- 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.
- Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362.
- Terno, J., Lindemann, R., and Scheithauer, G. (1987). *Zuschnittprobleme und ihre praktische Lösung: mathematische Modelle von Layout-Problemen*. Deutsch.
- 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. 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

```

1  $\mathcal{I}' \leftarrow \emptyset$  ; // Set of items in the solution
2  $\mathcal{L}^1 \leftarrow \{(H_{\max})\}$  ; // stage 1 states to process
3  $\mathcal{L}^2 \leftarrow \emptyset$  ; // pairs (stage 2 state, shelf) to process
4  $\mathcal{L}^3 \leftarrow \emptyset$  ; // pairs (stage 3 state, compartment) to process
5 while  $\mathcal{L}^1 \neq \emptyset$  do
6    $v = (H) \leftarrow$  arbitrary element of  $\mathcal{L}^1$  ;
7    $\mathcal{L}^1 \leftarrow \mathcal{L}^1 \setminus \{v\}$  ;
8   Choose arbitrarily  $\mathcal{F}$  such that  $x_{(\mathcal{F},v)} \geq 1$  ;
9    $x_{(\mathcal{F},v)} \leftarrow x_{(\mathcal{F},v)} - 1$  ;
10  for  $u \in \mathcal{F}$  do
11    if  $u$  is a stage 2 state  $(h_u, W_{\max})$  then
12       $\psi \leftarrow (h_u, W_{\max}, L_{\max})$  ; // creation of a shelf with height  $h_u$ 
13       $\Psi \leftarrow \Psi \cup \{\psi\}$  ;
14       $\mathcal{L}^2 \leftarrow \mathcal{L}^2 \cup \{(u, \psi)\}$  ; // add the shelf information to  $u$ 
15    else
16       $\mathcal{L}^1 \leftarrow \mathcal{L}^1 \cup \{u\}$  ;
17  while  $\mathcal{L}^2 \neq \emptyset$  do
18     $(v, \psi) = ((h_\psi, W), \psi) \leftarrow$  arbitrary element of  $\mathcal{L}^2$  ;
19     $\mathcal{L}^2 \leftarrow \mathcal{L}^2 \setminus \{(v, \psi)\}$  ;
20    Choose arbitrarily  $\mathcal{F}$  such that  $x_{(\mathcal{F},v)} \geq 1$  ;
21     $x_{(\mathcal{F},v)} \leftarrow x_{(\mathcal{F},v)} - 1$  ;
22    for  $u \in \mathcal{F}$  do
23      if  $u$  is a stage 3 state  $(h_\psi, w_u, L_{\max}, E_{h_\psi, w_u}^*, \mathbf{0})$  then
24         $\phi \leftarrow (h_\psi, w_u, L_{\max})$  ; // creation of a compartment with height  $h_\psi$ , width  $w_u$ 
25         $\Phi \leftarrow \Phi \cup \{\phi\}$ ,  $\mu(\phi) \leftarrow \psi$  ;
26         $\mathcal{L}^3 \leftarrow \mathcal{L}^3 \cup \{(u, \phi)\}$  ; // add the compartment information to  $u$ 
27      else
28         $\mathcal{L}^2 \leftarrow \mathcal{L}^2 \cup \{(u, \psi)\}$  ;
29  while  $\mathcal{L}^3 \neq \emptyset$  do
30     $(v, \phi) = ((h_\phi, w_\phi, L, e, \mathbf{q}), \phi) \leftarrow$  arbitrary element of  $\mathcal{L}^3$  ;
31     $\mathcal{L}^3 \leftarrow \mathcal{L}^3 \setminus \{(v, \phi)\}$  ;
32    Choose arbitrarily  $\mathcal{F}$  such that  $x_{(\mathcal{F},v)} \geq 1$  ;
33     $x_{(\mathcal{F},v)} \leftarrow x_{(\mathcal{F},v)} - 1$  ;
34     $u = (h_\phi, w_\phi, \tilde{L}, \tilde{e}, \tilde{\mathbf{q}}) \leftarrow$  the single element of  $\mathcal{F}$  ;
35    if  $e \in \mathcal{E}^{out}$  and  $\tilde{\mathbf{q}}_{i(e)} = 1$  then
36       $\rho(i(e)) \leftarrow \phi$  ;
37       $\mathcal{I}' \leftarrow \mathcal{I}' \cup \{i(e)\}$  ;
38    if  $\tilde{e} \neq \mathbf{0}$  then
39       $\mathcal{L}^3 \leftarrow \mathcal{L}^3 \cup \{(u, \phi)\}$ 
40 return  $(\Psi, \Phi, \mathcal{I}', \mu, \rho)$ 

```

---

## S.2. Hypergraph generation algorithms with all improvements of Section 4

---

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

---

**Input:** An SCPD instance  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$   
**Output:** A decision hypergraph associated with the given instance

```

1  $\mathcal{V} \leftarrow \{s, (H_{\max})\}$  ; // set of vertices of the hypergraph
2  $\mathcal{A} \leftarrow \emptyset$  ; // set of hyperarcs
3  $\mathcal{S}^1 \leftarrow \{(H_{\max})\}$  ; // stage 1 states to process
4  $\mathcal{S}^2 \leftarrow \emptyset$  ; // stage 2 states to process
5  $\mathcal{S}^3 \leftarrow \emptyset$  ; // stage 3 states to process
6  $\mathcal{S}^\dagger \leftarrow \emptyset, \mathcal{S}^\ddagger \leftarrow \emptyset$  ; // trivial states to process
7 while  $\mathcal{S}^1 \neq \emptyset$  do
8    $v = (H) \leftarrow$  element of  $\mathcal{S}^1$  that maximizes  $h$  ;
9    $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \setminus \{v\}$  ;
10  Apply Algorithm 2 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}), v, \mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^\dagger$  and  $(\mathcal{V}, \mathcal{A})$  as inputs ;
11 while  $\mathcal{S}^2 \neq \emptyset$  do
12    $v = (h, W) \leftarrow$  element of  $\mathcal{S}^2$  that maximizes lexicographically  $h$  and  $w$  ;
13    $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \setminus \{v\}$  ;
14   Apply Algorithm 3 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}), v, \mathcal{S}^2, \mathcal{S}^3, \mathcal{S}^\dagger$ , and  $(\mathcal{V}, \mathcal{A})$  as inputs ;
15 while  $\mathcal{S}^3 \neq \emptyset$  do
16    $v = (h, w, L, e, \mathbf{q}) \leftarrow$  element of  $\mathcal{S}^3$  that maximizes lexicographically  $h, w, e$  ;
17    $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \setminus \{v\}$  ;
18   Apply Algorithm 4 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}), v, \mathcal{S}^3, \mathcal{S}^\dagger$  and  $(\mathcal{V}, \mathcal{A})$  as inputs ;
19 while  $\mathcal{S}^\dagger \neq \emptyset$  do
20    $v = (w, k)^\dagger \leftarrow$  element of  $\mathcal{S}^\dagger$  that maximizes lexicographically  $w, k$  ;
21    $\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \setminus \{v\}$  ;
22   Apply Algorithm 10 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}), v, \mathcal{S}^\dagger$  and  $(\mathcal{V}, \mathcal{A})$  as inputs ;
23 while  $\mathcal{S}^\ddagger \neq \emptyset$  do
24    $v = (h, w, k)^\ddagger \leftarrow$  element of  $\mathcal{S}^\ddagger$  that maximizes lexicographically  $h, w, k$  ;
25    $\mathcal{S}^\ddagger \leftarrow \mathcal{S}^\ddagger \setminus \{v\}$  ;
26   Apply Algorithm 11 with  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I}), v, \mathcal{S}^\ddagger$  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  $((H_{\max}, W_{\max}, L_{\max}), \mathcal{I})$ , a stage 1 state  $v = (H)$ , three sets  $\mathcal{S}^1$ ,  $\mathcal{S}^2$  and  $\mathcal{S}^\dagger$ , an hypergraph  $(\mathcal{V}, \mathcal{A})$

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

```

1  $\bar{h} \leftarrow$  the maximum height such that  $v$  is  $\bar{h}$ -trivial, 0 if it does not exist ;           //  $h$ -triviality of  $v$ 
2  $\tilde{h} \leftarrow +\infty$  ;                                                                                       // Dominance rule 2
3 if  $\{\pi : c_1(v, \pi) = 1\} \neq \emptyset$  then
4    $\tilde{h} \leftarrow \min\{\pi : c_1(v, \pi) = 1\}$  ;
5  $\tilde{h} \leftarrow \min\{\tilde{h}, h\}$  ;
6  $C_2(h) \leftarrow$  True if  $h \leq \tilde{h}$ , False otherwise ;                                                       // Dominance rule 2
7  $C_3(h) \leftarrow$  True if  $c_1(v, h) < U_1(h)$ , False otherwise ;                                           // Dominance rule 3
8  $C_8(h) \leftarrow$  False if  $h \geq \bar{h}$ , True otherwise ;                                                 //  $h$ -triviality of  $v$ 
9 for  $h \in \mathcal{H}, h \leq H$  and  $C_2(h)$  and  $C_3(h)$  and not  $C_8(h)$  do
10    $\hat{H} \leftarrow \max\{\pi \in \mathcal{RH}_h : \pi \leq H - h\}$  ;                                                 // Proposition 4
11    $\hat{W} \leftarrow \max\{\pi \in \mathcal{RW}_h : \pi \leq W_{\max}\}$  ;                                             // Proposition 5
12    $v_1 \leftarrow (h, \hat{W})$  ;                                                                           // generate a stage 2 state
13    $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_1\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}, \mathcal{F} \leftarrow \{v_1\}$  ;
14   if  $\exists i \in \mathcal{I}, h_i \leq \hat{H}$  then
15      $v_2 \leftarrow (\hat{H})$  ;                                                                           // generate a stage 1 state
16      $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \cup \{v_2\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}, \mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$  ;
17    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$  ;                                                         // generate an hyperarc
18 if  $\bar{h} \neq 0$  then                                                                                       //  $h$ -triviality of  $v$ 
19   for  $w \in \mathcal{W}_{\bar{h}}$  do
20      $k_w(\bar{h}) \leftarrow \max\{k \in \{0, \dots, |I_w|\} : h_{I_w[k]} \leq \bar{h}\}$  ;
21      $v_1 \leftarrow (w, k_w(\bar{h}))^\dagger$  ;
22      $\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \cup \{v_1\}, \mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$  ;
23      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$  ;                                                         // generate an hyperarc

```

---

---

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

---

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

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

```

1 if  $v$  is trivial then
2   for  $w \in \mathcal{W}_h$  do
3      $k_w(h) \leftarrow \max \{k \in \{0, \dots, |\mathcal{I}_w|\} : h_{1_w[k]} \leq h\}$  ;
4      $v_1 \leftarrow (h, w, k_w(h))^\ddagger$  ;
5      $\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \cup \{v_1\}$ ,
6      $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$  ;
7      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$  ; // generate an hyperarc
8 else
9    $C_5(w) \leftarrow \text{True if } c_2(v, w) < U_2(h, w), \text{ False otherwise}$  ; // Dominance rule 5
10   $C_6(w) \leftarrow \text{True if } (((W = W_{\max}) \wedge (\exists i \in \mathcal{I}_{\leq h} : h_i = h, w_i = w)) \vee W \neq W_{\max}), \text{ False otherwise}$  ;
    // Dominance rule 6
11  for  $w \in \mathcal{W}_h, w \leq W$  and  $C_5(w)$  and  $C_6(w)$  do
12     $\hat{h} \leftarrow h_{\max}(\mathcal{I}_{\leq h, w})$  ; // Dominance rule 4
13     $\hat{W} \leftarrow \max\{\pi \in \mathcal{RW}_{h_{\max}(\mathcal{I}_{\leq h, w})} : \pi \leq W - w\}$  ; // Proposition 5
14     $v_1 \leftarrow (\hat{h}, w, L_{\max}, E_{\hat{h}, w}^*, \mathbf{0})$  ;
15     $v_2 \leftarrow (\hat{h}, \hat{W})$  ;
16     $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$ ,  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$ ,  $\text{tail} \leftarrow \mathcal{F} \cup \{v_1\}$  ;
17    if  $\exists i \in \mathcal{I} : h_i \leq h, w_i \leq w - \tilde{w}$  then
18       $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup \{v_2\}$ ,  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$ ,  $\mathcal{F} \leftarrow \mathcal{F} \cup \{v_2\}$  ;
19     $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \mathcal{F}, 0)\}$  ; // generate an hyperarc

```

---

---

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

---

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

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

```

1 if  $v$  is trivial then
2   if  $e = E_{h,w}^*$  then
3      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v^+\}, \sum_{i \in \mathcal{I}_{\leq h,w}} p_i), (v, \{v^+\}, 0)\}$ ;
4   else
5      $k_e \leftarrow \max\{k \in \{1, \dots, |\mathcal{I}_{\leq h,w}|\} : e^{\text{out}}(\mathcal{I}_{\leq h,w}[k]) \leq e\}$ ;
6      $v_1 \leftarrow (h, w, k_e)^\ddagger$ ;
7      $\mathcal{S}^\ddagger \leftarrow \mathcal{S}^\ddagger \cup \{v_1\}$ ,  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$ ;
8      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$ ; // generate an hyperarc
9 else
10  if  $e = 0$  then
11     $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{s\}, 0)\}$ ; // connect the source with this state
12  else if  $e \in \mathcal{E}^{\text{out}}$  then
13     $\hat{h} \leftarrow H^-(E_{h,w}^-(e), h)$ ; // Proposition 6
14     $v_1 \leftarrow (\hat{h}, w, L, E_{h,w}^-(e), \mathbf{q})$ ; // skipping the item
15     $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$ ,  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$ ;
16     $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$ ; // generate an hyperarc
17    if  $L \geq \ell_{i(e)}$  then
18       $v_2 \leftarrow (\hat{h}, w, L - \ell_{i(e)}, E_{h,w}^-(e), \mathbf{q} + \boldsymbol{\varepsilon}_{i(e)})$ ; // selecting the item
19       $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_2\}$ ,  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_2\}$ ;
20       $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_2\}, p_{i(e)})\}$ ; // generate an hyperarc
21  else if  $e \in \mathcal{E}^{\text{in}}$  then
22     $\hat{h} \leftarrow H^-(E_{h,w}^-(e), h)$ ; // Proposition 6
23    if  $\mathbf{q}_{i(e)} = 0$  then
24       $v_1 \leftarrow (\hat{h}, w, L, E_{h,w}^-(e), \mathbf{q})$ ; // skipping the item
25    else
26       $v_1 \leftarrow (\hat{h}, w, L + \ell_{i(e)}, E_{h,w}^-(e), \mathbf{q} - \boldsymbol{\varepsilon}_{i(e)})$ ; // selecting the item
27     $\mathcal{S}^3 \leftarrow \mathcal{S}^3 \cup \{v_1\}$ ,  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_1\}$ ;
28     $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v, \{v_1\}, 0)\}$ ; // generate an hyperarc

```

---

**Algorithm 10:** Create trivial states

---

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

```

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

```

---



### S.3. Instance generation

---

**Algorithm 12:** Generation algorithm for the instances of class U

---

**Input:**  $n$  the number of items,  $m$  the number of different widths

**Output:** An instance of the SCPD problem

```
1  $\mathcal{I} \leftarrow \emptyset$  denotes the set of items ;
2  $\mathcal{W} \leftarrow \emptyset$  denotes the set of different widths ;
3 for  $k \in \{0, \dots, m\}$  do
4    $w \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
5    $\mathcal{W} \leftarrow \mathcal{W} \cup \{w\}$  ;
6  $n_w \leftarrow \lceil \frac{n}{|\mathcal{W}|} \rceil$  ;
7 for  $w \in \mathcal{W}$  do
8   for  $k \in \{0, \dots, n_w\}$  do
9      $i$  denotes an item ;
10     $h_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
11     $w_i \leftarrow w$  ;
12     $\ell_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
13     $p_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
14     $s_i \leftarrow$  value from  $\mathcal{U}[0, s_{\max}]$  ;
15     $d_i \leftarrow$  value from  $\mathcal{U}[1, d_{\max}]$  ;
16 return  $((300, 300, 300), \mathcal{I})$ 
```

---

---

**Algorithm 13:** Generation algorithm for the instances of class I

---

**Input:**  $C$  the number of cliques

**Output:** An instance of the SCPD problem

```
1  $\mathcal{I} \leftarrow \emptyset$  denotes the set of items ;
2  $\mathcal{W} \leftarrow \emptyset$  denotes the set of different widths ;
3 for  $k \in \{0, \dots, |\mathcal{W}|\}$  do
4    $w \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
5    $\mathcal{W} \leftarrow \mathcal{W} \cup \{w\}$  ;
6 for  $w \in \mathcal{W}$  do
7    $a \leftarrow$  value from  $[a_{\min}, a_{\max}]$  ;
8    $c \leftarrow \emptyset$  denotes a clique of items ;
9   for  $k \in \{1, \dots, a\}$  do
10     $i$  denotes an item ;
11     $h_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
12     $w_i \leftarrow w$  ;
13     $\ell_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
14     $p_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
15     $s_i \leftarrow 0$  ;
16     $d_i \leftarrow -1$  ;
17     $c \leftarrow c \cup \{i\}$  ;
18  for  $t \in \{1, \dots, C\}$  do
19     $b \leftarrow$  value from  $\mathcal{U}[b_{\min}, b_{\max}]$  ;
20    for  $i \in c$  do
21       $p \leftarrow$  value from  $\mathcal{U}[0, 100]$  ;
22      if  $p > b$  then
23         $d_i \leftarrow t - s_i$  ;
24         $I \leftarrow I \cup \{i\}$  ;
25         $c \leftarrow c \setminus \{i\}$  ;
26     $a \leftarrow$  value from  $[a_{\min}, a_{\max}]$  ;
27    for  $k \in \{1, \dots, a\}$  do
28       $i$  denotes an item ;
29       $h_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
30       $w_i \leftarrow w$  ;
31       $\ell_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
32       $p_i \leftarrow$  value from  $\mathcal{U}[r_{\min}, r_{\max}]$  ;
33       $s_i \leftarrow t$  ;
34       $d_i \leftarrow -1$  ;
35       $c \leftarrow c \cup \{i\}$  ;
36 return  $((300, 300, 300), \mathcal{I})$ 
```

---