Robust storage loading problems with stacking and payload constraints

Marc Goerigk[a,*], Sigrid Knust[b], Xuan Thanh Le[b]

[a]*Department of Management Science, Lancaster University, United Kingdom*

[b]*Department of Mathematics and Computer Science, University of Osnabrück, Germany*

Abstract

We consider storage loading problems where items with uncertain weights have to be loaded into a storage area, taking into account stacking and payload constraints. Following the robust optimization paradigm, we propose strict and adjustable optimization models for finite and interval-based uncertainties. To solve these problems, exact decomposition and heuristic solution algorithms are developed. For strict robustness, we also propose a compact formulation based on a characterization of worst-case scenarios. Computational results for randomly generated data with up to 300 items are presented showing that the robustness concepts have different potential depending on the type of data being used.

Keywords: OR in maritime industry; storage loading; stacking problems; payload restrictions; robust optimization

1. Introduction

Storage loading problems appear in several practical applications, e.g., in the context of container terminals, container ships, warehouses or steel yards, see [33]. Especially container transportation plays a vital role in today's global economy. For convenient transportation by different modalities, containers are standardized in term of twenty-feet-equivalent-units (TEUs), which refers to containers of twenty-feet lengths. According to an executive summary in [42], the throughput of container ports all over the world was 651.1 million TEUs in 2013 (corresponding to 1524 millions tons of cargo), in which 160 million TEUs were transported by ships. Therefore, it can be seen that the transportation of seaborne containers contributes an important part in international trade. Furthermore, there has been a progressive increase in maximum container-ship size, from 500 TEU ships by the mid 1950's to 18000 TEU vessels by 2013 (see [32], page 6). Such numbers make professional and efficient operations essential in container ships. This motivates an active research topic on finding efficient solution methods for maritime container transportation. An up-to-date collection of planning and scheduling problems in large maritime container yards, together with latest solution approaches, can be found in [34]. Many other topics on maritime container transportation, such as tactical and operational management in shipping liners, empty container repositioning, etc., are collected in [32].

In this paper, we focus on storage loading problems where incoming items arrive at a storage area and have to be assigned to stacks so that certain constraints are respected. Usually, only the topmost item of each stack can be directly retrieved, i.e., the items are accessed in last-in-first-out order. The items are often relocated by cranes moving above the stacks, which imposes

[*]Corresponding author

Email addresses: `m.goerigk@lancaster.ac.uk` (Marc Goerigk), `sigrid.knust@uni-osnabrueck.de` (Sigrid Knust), `xuanthanh.le@uni-osnabrueck.de` (Xuan Thanh Le)

a restriction on the maximum height of a stack. Additionally, certain stacking constraints have to be respected, i.e., not every item may be stacked on every other item. For example, larger items are not allowed to be put on smaller ones, or items with a later departure time may not be stacked on items with an earlier departure time. For a survey and a classification scheme of such problems we refer to [33]. Complexity results for stacking problems taking into account a limited height of the stacks and stacking constraints are provided in [18].

In many practical loading problems, additional stability issues are crucial. In load planning of trains (cf. [16, 17]) containers have to be loaded onto wagons so that the stability of each wagon is guaranteed. In one-dimensional balanced loading problems (cf. [6, 36]), one has to pack a set of homogeneous blocks of given length and weight in an one-dimensional container so that the center of gravity of the packed blocks is as close to a target point as possible. An extension of this problem to two dimensions can be found in air cargo load planning (cf. [43]), where a set of cargo has to be loaded on an aircraft minimizing the deviation between the aircraft's center of gravity and a given point (in both the longitudinal and lateral directions) to improve stability of the aircraft and reduce fuel consumption. In many studies on three-dimensional container loading problems (see [14] and references therein), the aim is to find a best three-dimensional packing pattern for loading a subset of rectangular boxes into a container maximizing the total value. Here, stability issues arise due to the strength of the boxes' faces or the maximum number of boxes that can be stacked one above each other.

Also, when loading items onto a ship, the stability of the ship is an important issue that needs to be taken into account. It follows from a physical principle (see [25], Chapter 12) that the position of the gravity center of the loaded ship affects the ship's stability in the following sense: the lower the gravity center, the more stable the ship is. Consequently, as shown in Appendix A, to obtain the best stability of a ship (i.e., to have the lowest gravity center of the loaded ship), the items should be stored in such a way that heavier items are assigned to lower levels. Therefore, in the existing literature about storage loading problems in containerships, stability issues of the ships are mostly handled by imposing hard stacking constraints on the weights of the containers (i.e., heavier containers must be put below lighter ones). For example, such stacking constraints appear in the context of the master bay plan problem (MBPP) (cf. [4, 5, 40]). Formally, this problem is to determine a plan of minimum operating time for stowing a set of containers of different types into available locations of a containership, with respect to some structural and operative constraints (e.g., a restriction on the maximum weight of the containership, containers retrieved later may not be stored on top of containers that are retrieved earlier). For the equilibrium of ships, the weights of containers are classified into three groups (light, medium, heavy), and the following restrictions are considered. First, the total weight of three consecutive containers in a stack cannot be greater than an a priori established value. Second, the weight on the right side of the ship should not differ much from the weight on the left side (for cross equilibrium). Finally, the stacking constraints on weights are applied to guarantee horizontal equilibrium. In [4] the authors propose a binary linear programming model for the problem, present a heuristic approach, and give some prestowage rules for being able to solve the model. Another solution approach for the same problem is presented in [5] based on a three-phase algorithm using the idea of splitting the ship into different parts and assigning the containers to them on the basis of the containers' destinations. In [40] a heuristic method based on a relationship of the MBPP to three-dimensional bin packing problems is proposed.

In real-world containership loading problems, the hard stacking constraints on the containers' weights might be too conservative due to their interaction with other practical constraints. Moreover, the lowest center of gravity of the loaded ship caused by imposing the hard stacking constraints on weights might make the ship become too rigid, which may be bad for the ship when hit by waves. To get rid of these issues, a simple approach is to assume that the total weight of the containers allocated in a stack is limited by a given bound. For example, this approach is applied in [20, 21] to generate optimal stowage plans for container vessel bays by using constraint programming techniques. Another approach is to impose a limited area for the gravity center of the ship. One may find the use of this approach in literature dealing with the multi-port master bay plan problem (MP-MBPP), which is an extended version of the (one-port) MBPP mentioned above. In the MP-MBPP, the whole route of a ship is considered, and different sets of containers are loaded at each port of the route for shipping to successive ports. Various objective functions are considered, as well as different solution methods are proposed, (see e.g. [1, 2, 3, 28, 38, 39]). There, the containers are sorted according to their departure ports. In turn, the containers of the same departure port are classified into different weight groups, where the average weight of each group is assigned to every container belonging to the group. The ship is divided into different parts called bays, each bay in turn is partitioned into sub-sections. Then the containers are stored into these sub-sections in such a way that the center of gravity of the ship is within a limited area.

In the paper at hand, we use another approach to tackle the mentioned drawbacks of hard stacking constraints on weights and to control the stability of a ship by imposing additional constraints on the payload of the items. More precisely, we assume that the total weight that can be put on top of an item i with weight w_i must be limited by aw_i, where a is a given positive parameter which may depend on the stacks. With this additional constraint on the payload of the items, the height of the highest possible position of the gravity center of each stack is an increasing function with respect to a (see Appendix A). The payload parameter a can therefore be chosen to achieve the desired position of the gravity center of each stack, and consequently it can be used to control the stability of the ship.

In practice, it might also be possible that payload violations are allowed and the gravity center of the ship may be shifted to a higher position. To achieve the desired stability of the ship, an amount of ballast corresponding to the total payload violation is put at the bilge of the ship so that the gravity center of the whole ship is adjusted to a safe position (cf. [45]). By minimizing the total payload violation over all stacks, the amount of ballast needed is minimized, and consequently, the shipping cost is reduced while the ship's stability is guaranteed.

Since in real-world applications, often not all data are exactly known during the planning stage, we consider storage loading problems under data uncertainty in this paper. In particular, we assume that the weight of each item is uncertain and may come either from a finite set of possible scenarios or from an interval of potential outcomes. We consider two approaches to include robustness in this setting: strict robustness (see [10]), where the location of each item needs to be fixed before its actual weight becomes known, and adjustable robustness (see [9]), where each item must only be assigned to a stack, but its position within the stack can be decided once the weight is known. Our approach using payload constraints takes advantage of using knowledge of all possible weight outcomes of the items, rather than binning the items into groups and assigning to each item the average weight of its group.

For general surveys on robust optimization, we refer to [8, 12, 24, 30]. Other applications of robust optimization include disaster management [7, 23], load planning of trains [16], shunting problems [19], empty container repositioning [22], and many more. Storage loading problems with uncertain data have, for example, been studied in [29]. There, the weights of the items are classified into three groups and the weight group of each item is not known before its arrival. In the storage area, the heavier items should be stored in higher levels of the stacks, since they have to be retrieved earlier to put them in the bottom of a vessel according to the hard stacking constraints on weight. In [27] the authors propose a simulated annealing algorithm to find a good stacking strategy for a similar problem where the incoming items have uncertain weights. In [31], incoming items arriving at a partly filled storage area have to be assigned to stacks regarding that not every item may be stacked on top of every other item and taking into account uncertain data of items arriving later. Strict and adjustable robust solutions are calculated using different MIP formulations.

The remainder of the paper is organized as follows. In Section 2, we describe the deterministic stacking problem and formally introduce the uncertainty sets. The strictly robust counterpart of this uncertain problem is considered for both finite and interval uncertainty sets in Section 3, while adjustable counterparts are discussed in Section 4. Computational experiments are presented in Section 5. Finally, conclusions can be found in Section 6.

2. Problem formulation

In this section, we give a formal definition of the studied storage problem, formulate its deterministic version as a mixed-integer program (MIP), and introduce the considered uncertainties.

2.1. Nominal problem

In the following, we describe the nominal (deterministic) problem in more detail. Let m be the number of stacks where for each stack a position in the storage area is fixed. Each stack can hold at most b items. The set of all items is denoted by $I = \{1, 2, \ldots, n\}$ where normally the inequality $m < n$ holds, i.e., some items have to be stacked on others. Since all items have to be stacked, we assume that $n \leq bm$; otherwise the problem is infeasible.

As a hard constraint we assume that not every item may be stacked on every other item (for example, a larger item may not be stacked on top of a smaller one or an item with a later departure time may not be stacked on top of one with an earlier departure time). Such stacking constraints may be encoded by a 2-dimensional binary matrix $S = (s_{ij})_{n\times n}$, where $s_{ij} = 1$ if i can be stacked onto j and $s_{ij} = 0$ otherwise. Stacking constraints may be transitive (i.e., if item i is stackable on top of item j and item j is stackable on top of item h, then also i is stackable on top of h) or may have an arbitrary structure. For example, restrictions coming from lengths or departure times of the items are transitive, while restrictions induced by materials may be non-transitive. In this paper we develop models which are capable to deal with stacking constraints of an arbitrary structure.

Items stored in a stack are defined by a tuple $(i_k, \ldots, i_1)$, where i_l denotes the item stacked at level l and $l = 1$ corresponds to the ground level. Such a tuple is feasible if $k \leq b$ and $s_{i_{l+1},i_l} = 1$ for all $l = 1, \ldots, k-1$.

Additionally, we assume that each item $i \in I$ has a weight w_i and that the total weight of items put on top of item i should not be larger than aw_i with a given payload factor $a \in \mathbb{R}_+$. If the total weight W of all items above i exceeds aw_i, a payload violation of $W - aw_i$ occurs. The total payload violation of a stacking configuration is defined as the sum of the payload violations over all items in all stacks of the configuration. Note that all our models are also valid for the more general situation where the payload factor may depend on the assigned stack. In this paper, the payload constraints are assumed to be soft, i.e., payload violations are allowed, but have to be minimized. Our discussion also includes the situation of hard payload constraints, since in this case a feasible solution exists if and only if the minimal total payload violation is equal to zero.

The simplest version of a storage loading problem is the feasibility problem (cf. [18]) which asks whether all items can be feasibly allocated to the storage area respecting all hard constraints, i.e., the stack capacity b and the stacking constraints s_{ij}. If this is possible, the objective is to assign each item to a feasible location (specified by a stack number and a level in the stack). In [18] it was shown that deciding whether a feasible solution exists is strongly NP-complete even for

$b = 3$ and transitive stacking constraints. In an optimization version of the problem additionally some objective function (e.g., the total number of used stacks or the number of items stacked above the ground level) may be minimized. In this paper we concentrate on minimizing the total payload violation as the objective function. This problem is strongly NP-hard for $b \geq 3$, since it generalizes the feasibility problem.

2.2. A MIP formulation

In the following, we present a MIP formulation for the nominal problem where $w \in \mathbb{R}^n_+$ is the vector of the nominal weights of all items. Furthermore, let $Q := \{1, \dots, m\}$ be the set of stacks and $L := \{1, \dots, b\}$ be the set of levels. We use the notation $[\alpha]_+$ to indicate $\max\{\alpha, 0\}$. Let x_{iql} for $i \in I, q \in Q, l \in L$ be binary variables with

$$x_{iql} = \begin{cases} 1, & \text{if item } i \text{ is stored in stack } q \text{ at level } l, \\ 0, & \text{otherwise.} \end{cases}$$

For a stacking configuration encoded by x, the payload violation of an item in stack $q \in Q$ at level $l \in L \setminus \{b\}$ is

$$\left[\sum_{j \in I} \sum_{h=l+1}^{b} w_j x_{jqh} - a \sum_{i \in I} w_i x_{iql} \right]_+$$

and hence the total payload violation of the configuration is given by

$$f(x, w) := \sum_{q \in Q} \sum_{l \in L \setminus \{b\}} \left[\sum_{j \in I} \sum_{h=l+1}^{b} w_j x_{jqh} - a \sum_{i \in I} w_i x_{iql} \right]_+$$

To linearly represent the objective function $f(x, w)$, we use additional non-negative variables v_{ql} for $q \in Q, l \in L$ to compute the payload violation of the item stored in stack q at level l. Then the problem can be formulated as follows.

$$\min \sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v_{ql} \tag{1}$$

$$\text{s.t.} \sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 \qquad \forall i \in I \tag{2}$$

$$\sum_{i \in I} x_{iql} \leq 1 \qquad \forall q \in Q, l \in L \tag{3}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x_{jq,l-1} \geq x_{iql} \qquad \forall i \in I, q \in Q, l \in L \setminus \{1\} \tag{4}$$

$$\sum_{j \in I} \sum_{h=l+1}^{b} w_j x_{jqh} - a \sum_{i \in I} w_i x_{iql} \leq v_{ql} \qquad \forall q \in Q, l \in L \setminus \{b\} \tag{5}$$

$$x_{iql} \in \{0, 1\} \qquad \forall i \in I, q \in Q, l \in L \tag{6}$$

$$v_{ql} \geq 0 \qquad \forall q \in Q, l \in L \setminus \{b\} \tag{7}$$

According to (1) the sum of all payload violations is minimized. Constraints (2) guarantee that all items are stored. Constraints (3) ensure that at most one item is stored at each level of each stack. Due to (4) the stacking constraints s_{ij} are satisfied and no item is placed to a location where no item is stacked below. Inequalities (5) ensure that the payload violations v_{ql} are computed correctly.

We refer to this problem as (P). Note that (P) uses $\mathcal{O}(nmb)$ variables and constraints. In the following we denote by

$$\mathcal{X} := \left\{ x \in \{0,1\}^{|I|\times|Q|\times|L|} \mid x \text{ satisfies (2)-(4)} \right\}$$

the set of feasible solutions respecting all hard constraints of the stacking problem.

2.3. Uncertainties

In this paper, we consider two kinds of uncertainties for the item weights that affect the payload constraints:

- In the first case, we assume that for every item i, we are provided with lower and upper bounds $\underline{w}_i$, $\overline{w}_i$ on the possible outcome of item weights. We write

$$\mathcal{U}^I = [\underline{w}_1, \overline{w}_1] \times \ldots \times [\underline{w}_n, \overline{w}_n]$$

  to denote an interval-based uncertainty set. An element $w \in \mathcal{U}^I$ is called a scenario. The lower and upper bounds stem from empirical observations or expert knowledge. We do not assume knowledge of any probability distribution over $\mathcal{U}^I$.

- In the second case, we assume that we are given a list of N possible scenarios, where as before a scenario consists of a weight for each item. We write

$$\mathcal{U}^F = \{w^1, \ldots, w^N\}$$

  for the uncertainty set containing all possible outcomes. Such a description of scenarios may either be based on the expertise of practitioners (e.g., an experienced storage loading manager is able to enumerate typical outcomes of uncertain weights), or may stem from a probabilistic analysis and represents the most likely outcomes. We write $\mathcal{N} = \{1, \ldots, N\}$.

In case that we do not need to distinguish these two kinds of uncertainty sets, we simply write $\mathcal{U}$ to denote the uncertainty set. Note that $\mathcal{U}^F$ is a finite set, while $\mathcal{U}^I$ contains infinitely many possible outcomes. This leads to different solution approaches for the robust models we consider in this paper.

3. Strict robustness

In this section, we consider the problem setting where a complete stacking solution has to be fixed in advance before the actually realized scenario becomes known. Such an approach is required if the storage plan has to be announced before the actual weights of the items are known and the plan cannot be changed later on. This means that the planner has to find a complete stacking solution, i.e., to decide for each item to which stack and level it is assigned, based on incomplete knowledge. Following the approach of [10, 30], we focus on strictly robust solutions where the worst-case payload violation over all scenarios is minimized. The strictly robust counterpart (SR, $\mathcal{U}$) of the optimization storage loading problem (P) under affection of uncertainty set $\mathcal{U}$ is

$$(\text{SR}, \mathcal{U}) \qquad \min_{x\in\mathcal{X}} \max_{w\in\mathcal{U}} f(x,w).$$

We first consider the case of finite uncertainty in Section 3.1, afterwards the more elaborate case of interval-based uncertainty is discussed in Section 3.2.

3.1. Finite uncertainty

In the following, we modify the problem formulation (P) to include a finite uncertainty set. First, we introduce new variables $v_{ql}^k \geq 0$ for $q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N}$ measuring the payload violation of the item stored in stack q at level l in the solution for scenario k. Additionally, an auxiliary variable $v \geq 0$ is introduced to measure the total payload violation in the worst-case over all scenarios. We denote this problem as (SRF), though we may also write (SR, $\mathcal{U}^F$) when the usage of uncertainty set $\mathcal{U}^F$ should be emphasized, and obtain the following MIP formulation:

$$
\begin{aligned}
&\text{(SRF)} \qquad \min v && && (8)\\
&\sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 && \forall i \in I && (9)\\
&\sum_{i \in I} x_{iql} \leq 1 && \forall q \in Q, l \in L && (10)\\
&\sum_{j \in I \setminus \{i\}} s_{ij} x_{jq,l-1} \geq x_{iql} && \forall i \in I, q \in Q, l \in L \setminus \{1\} && (11)\\
&\sum_{j \in I} \sum_{h=l+1}^{b} w_j^k x_{jqh} - a \sum_{i \in I} w_i^k x_{iql} \leq v_{ql}^k && \forall q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} && (12)\\
&\sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v_{ql}^k \leq v && \forall k \in \mathcal{N} && (13)\\
&x_{iql} \in \{0,1\} && \forall i \in I, q \in Q, l \in L && (14)\\
&v_{ql}^k \geq 0 && \forall q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} && (15)\\
&v \geq 0 && && (16)
\end{aligned}
$$

The objective (8) is to minimize the largest total payload violation over all scenarios. As in the nominal problem, constraints (9) ensure that every item is stored, constraints (10) model that at most one item is assigned to every location, and constraints (11) take care of the stacking constraints s_{ij}. Furthermore, according to constraints (12) the payload violations v_{ql}^k are correctly computed for every scenario k. Finally, constraints (13) (together with the minimization in (8)) ensure that the variable v equals the maximum of these values. Compared to the nominal model (which uses $\mathcal{O}(nmb)$ variables and constraints), this formulation requires $\mathcal{O}((n+N)mb)$ variables and constraints.

3.2. Interval uncertainty

We now consider (SR, $\mathcal{U}^I$), i.e., the strictly robust model with interval-based uncertainty sets $\mathcal{U}^I$. We denote this problem as (SRI) for short. Due to the continuous nature of the uncertainty set $\mathcal{U}^I$, there are infinitely many scenarios and we cannot include all these scenarios into a MIP formulation as we have done with (SRF). Therefore, we discuss approaches that iteratively choose scenarios from $\mathcal{U}^I$ to include them in a finite uncertainty set (see also [15, 37, 44], where similar ideas have successfully been applied). The general procedure is shown in Algorithm 1. We start with an arbitrary scenario $w^0 \in \mathcal{U}^I$. In each iteration k, we find a best stacking configuration x^k with respect to the current (finite) set of scenarios $\mathcal{U}^k$, i.e., x^k is a solution to (SR, $\mathcal{U}^k$). Then we determine a worst-case scenario $w^{k+1} \in \mathcal{U}^I$ corresponding to this stacking configuration (i.e., a scenario of item weights maximizing the total payload violation $f(x^k, w)$ for configuration x^k). The worst-case scenario found in each step is added to the current set of scenarios. This is repeated until the objective value of the robust problem and the objective value of the worst-case problem coincide.

Algorithm 1 Exact algorithm for (SRI)

```
Require: An instance of (SRI).
 1: $k \leftarrow 0$
 2: Take an arbitrary scenario $w^1 \in \mathcal{U}^I$ and let $\mathcal{U}^1 \leftarrow \{w^1\}$.
 3: repeat
 4:     $k \leftarrow k+1$
 5:     Solve (SR, $\mathcal{U}^k$). Let $x^k$ be the resulting stacking solution and $LB^k$ its objective value.
 6:     Find a scenario $w^{k+1} \in \mathcal{U}^I$ that maximizes the total payload violation for $x^k$.
 7:     Let $UB^k$ be the corresponding (worst-case) total payload violation.
 8:     $\mathcal{U}^{k+1} \leftarrow \mathcal{U}^k \cup \{w^{k+1}\}$
 9: until $UB^k = LB^k$
10: return optimal stacking solution $x^k$
```

Theorem 1. *Algorithm 1 terminates after a finite number of iterations and yields an optimal solution x to (SRI).*

Proof: Let f^* be the optimal objective value of (SRI), i.e.,

$$f^* = \min_{x \in \mathcal{X}} \max_{w \in \mathcal{U}^I} f(x, w).$$

Since $\mathcal{U}^k \subseteq \mathcal{U}^{k+1} \subset \mathcal{U}^I$, we have

$$\begin{aligned} LB^k &= \min_{x \in \mathcal{X}} \max_{w \in \mathcal{U}^k} f(x, w) \\ &\leq \min_{x \in \mathcal{X}} \max_{w \in \mathcal{U}^{k+1}} f(x, w) = LB^{k+1} \\ &\leq \min_{x \in \mathcal{X}} \max_{w \in \mathcal{U}^I} f(x, w) = f^*. \end{aligned}$$

On the other hand, by definition of UB^k in Step 7 of the algorithm, we have

$$UB^k = f(x^k, w^{k+1}) = \max_{w \in \mathcal{U}^I} f(x^k, w) \geq \min_{x \in \mathcal{X}} \max_{w \in \mathcal{U}^I} f(x, w) = f^*.$$

This means that LB^k is a lower bound on f^* and UB^k is an upper bound on f^*. Therefore, if $LB^k = UB^k$, then (x^k, w^{k+1}) is an optimal solution to (SRI) and $LB^k = f^*$ is the optimal objective value of (SRI).

We now show that if $LB^k \neq UB^k$, then $w^{k+1} \notin \mathcal{U}^k$, so that the algorithm never enters a cyclic loop. Indeed, assume to the contrary that $w^{k+1} \in \mathcal{U}^k$. As defined in Step 6 of the algorithm, we have

$$\begin{aligned} & w^{k+1} = \operatorname*{argmax}_{w \in \mathcal{U}^I} f(x^k, w) \\ \Leftrightarrow\quad & UB^k = f(x^k, w^{k+1}) \geq f(x^k, w) \quad \forall w \in \mathcal{U}^I \\ \Rightarrow\quad & UB^k = f(x^k, w^{k+1}) \geq f(x^k, w) \quad \forall w \in \mathcal{U}^k && \text{(since } \mathcal{U}^k \subset \mathcal{U}^I\text{)} \\ \Leftrightarrow\quad & UB^k = f(x^k, w^{k+1}) = \max_{w \in \mathcal{U}^k} f(x^k, w) && \text{(since } w^{k+1} \in \mathcal{U}^k\text{).} \end{aligned}$$

Moreover, as defined in Step 5 of the algorithm, we have $LB^k = \max_{w \in \mathcal{U}^k} f(x^k, w)$. Therefore, we again obtain $LB^k = UB^k$ under the assumption that $w^{k+1} \in \mathcal{U}^k$. This means that if $LB^k \neq UB^k$, then we must have $w^{k+1} \notin \mathcal{U}^k$.

The termination of the algorithm after a finite number of iterations follows immediately from the two following claims: (a) the number of possible stacking configurations x^k generated by the algorithm is finite, (b) the number of possible worst-case scenarios w^k generated by the algorithm is also finite. Indeed, since there is a finite number of items, also the number of possible stacking configurations for these items is finite, and claim (a) follows. By Step 6 of the algorithm, we generate only one worst-case scenario w^{k+1} corresponding to stacking configuration x^k, so claim (b) follows from claim (a). □

How to solve Step 5 of the algorithm was shown in Section 3.1. We now discuss how Step 6 can be realized. Given a stacking configuration x, we need to find a vector $w \in \mathcal{U}^I$ of item weights maximizing the total payload violation (i.e., the sum of payload violations over all levels of all stacks). Since there is no payload violation in stacks containing only one item, we have to compute the maximum total payload violation in all stacks containing at least two items.

For any stack $q \in Q$ in the given configuration x, let $I(q)$ be the set of items contained in it and $L(q) := \{1, \ldots, |I(q)|\}$. We assume that $|I(q)| \geq 2$. For the sake of simplicity, we denote the weight of the item at level $l \in L(q)$ by $w_{[l]}$.

Lemma 2. *The total payload violation of any stack $q \in Q$, given by*

$$v_q(w) = \sum_{l \in L(q)} \left[\sum_{h>l} w_{[h]} - a w_{[l]} \right]_+$$

is a convex function.

Proof: We have that $\sum_{h>l} w_{[h]} - a w_{[l]}$ is linear in w, and taking the maximum of two convex functions is again a convex function. Since the sum of convex functions is also convex, the claim follows. □

We denote the problem of finding item weights w that maximize the payload violation $v_q(w)$ for a stack $q \in Q$ as (V). Due to the convexity of v_q, we can make use of the fact that there is always an optimal solution to (V) where each item weight is at its lower or upper bound. This gives rise to the following formulation as a mixed-integer program. For all $l \in L(q)$, we introduce continuous variables $w_{[l]}$ determining the weight of the item at level l in q. Furthermore, we use binary variables β_l with $\beta_l = 0$ if $w_{[l]} = \underline{w}_{[l]}$ and $\beta_l = 1$ if $w_{[l]} = \overline{w}_{[l]}$. Finally, variables α_l are used to correctly compute the payload violation $[\sum_{h>l} w_{[h]} - a w_{[l]}]_+$ in the objective function. We have $\alpha_l = 1$ if $\sum_{h>l} w_{[h]} - a w_{[l]} \geq 0$ and $\alpha_l = 0$ if $\sum_{h>l} w_{[h]} - a w_{[l]} < 0$. Then, problem (V) can be formulated as follows.

$$\text{(V)} \quad \max \sum_{l \in L(q)} \left(\sum_{h>l} w_{[h]} - a w_{[l]} \right) \alpha_l \tag{17}$$

$$\text{s.t.} \quad w_{[l]} = \underline{w}_{[l]} + (\overline{w}_{[l]} - \underline{w}_{[l]}) \beta_l \qquad \forall l \in L(q) \tag{18}$$

$$\alpha_l, \beta_l \in \{0, 1\} \qquad \forall l \in L(q) \tag{19}$$

$$w_{[l]} \geq 0 \qquad \forall l \in L(q) \tag{20}$$

Note that this formulation is non-linear, due to the product of α and w in the objective function. We can remove variables $w_{[l]}$ by inserting equations (18) in the objective function, and get the equivalent model

$$\max \sum_{l \in L(q)} \sum_{h>l} \left(\underline{w}_{[h]} + \left(\overline{w}_{[h]} - \underline{w}_{[h]} \right) \beta_h \right) \alpha_l$$

$$- \sum_{l \in L(q)} a \left(\underline{w}_{[l]} + \left(\overline{w}_{[l]} - \underline{w}_{[l]} \right) \beta_l \right) \alpha_l \tag{21}$$

$$\text{s.t. } \alpha_l, \beta_l \in \{0,1\} \qquad \forall l \in L(q) \tag{22}$$

By introducing new variables $\gamma_{lh} = \alpha_l \cdot \beta_h$ for all $l, h \in L(q)$, we obtain the binary linear program

$$\max \sum_{l \in L(q)} \left(\sum_{h>l} \underline{w}_{[h]} - a\underline{w}_{[l]} \right) \alpha_l$$

$$+ \sum_{l \in L(q)} \left(\sum_{h>l} (\overline{w}_{[h]} - \underline{w}_{[h]})\gamma_{lh} - a(\overline{w}_{[l]} - \underline{w}_{[l]})\gamma_{ll} \right) \tag{23}$$

$$\text{s.t. } \alpha_l + \beta_h - 1 \leq \gamma_{lh} \leq \frac{1}{2}(\alpha_l + \beta_h) \qquad \forall l, h \in L(q) \tag{24}$$

$$\alpha_l, \beta_l \in \{0,1\} \qquad \forall l \in L(q) \tag{25}$$

$$\gamma_{lh} \in \{0,1\} \qquad \forall l, h \in L(q) \tag{26}$$

The objective function (23) is the same as in (21) after substituting and reordering terms. The additional constraints (24) are used to ensure that γ_{lh} is one if and only if both α_l and β_h are one. Solving problem (23)-(26) independently for each stack $q \in Q$ hence gives the desired solution to Step 6 of Algorithm 1.

Note that maximizing a convex function over a convex domain in general is an NP-hard problem [11]. However, we can show that for this special case an efficient solution algorithm exists.

Theorem 3. *For a given stacking solution and any value of the payload parameter a, the maximum total payload violation of each stack $q \in Q$ can be found by evaluating $\mathcal{O}(|I(q)|^{2\delta-1})$ scenarios, where $\delta := \min\{\lceil a \rceil, \lfloor \frac{|I(q)|}{2} \rfloor\}$.*

Proof: As mentioned above, due to the convexity of v_q, to find an optimal solution to (V) it is sufficient to consider only scenarios where the weights of all items in $I(q)$ are either on their lower or upper bounds. For a choice of item weights w, we say that at level $l < |I(q)|$ a solution has a *break* if $w_{[l]} = \underline{w}_{[l]}$ and $w_{[l+1]} = \overline{w}_{[l+1]}$, and an *anti-break* if $w_{[l]} = \overline{w}_{[l]}$ and $w_{[l+1]} = \underline{w}_{[l+1]}$. An example with six items, two breaks and one anti-break is depicted in Figure 1, left. Items having their upper-bound weights are painted in gray, while items having their lower-bound weights are painted in white. There are breaks at levels two and five, and an anti-break at level three.

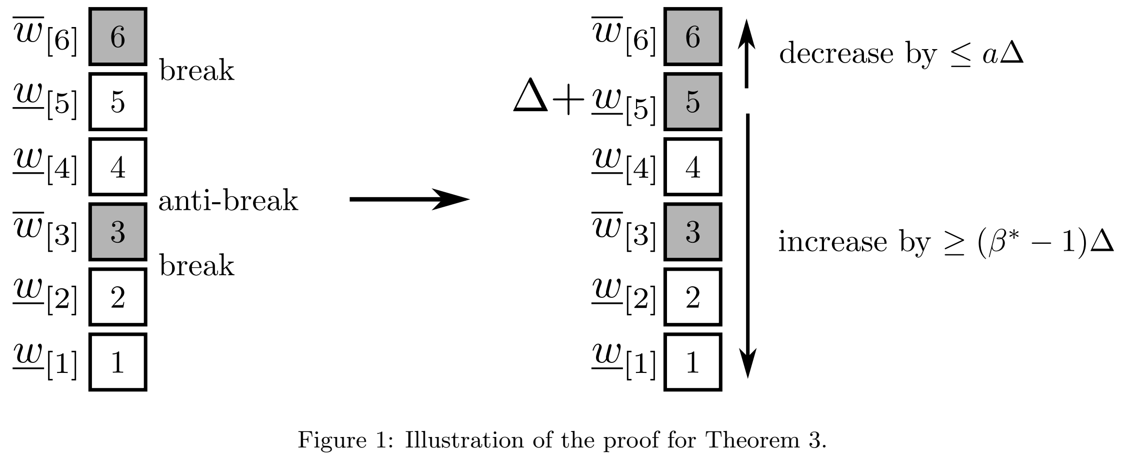

Figure 1: Illustration of the proof for Theorem 3.

Note that there is alway an optimal solution where the bottom item is as light as possible, and the top item is as heavy as possible; therefore, there is always an optimal solution with at least one break. Whenever there is a break at some level l of an optimal solution to (V), without loss of generality we can assume that $\sum_{h>l} w_{[h]} - aw_{[l]} \geq 0$. Indeed, if this was not the case, then we have $\sum_{h>l} w_{[h]} < aw_{[l]}$, i.e., there is no payload violation at level l of stack q. Therefore there is still no payload violation at this location if we increase $w_{[l]}$ by a positive amount. In other words, we could increase the weight of the item at level l without decreasing $v_q(w)$.

We now show that there is an optimal solution to (V) with at most δ breaks. Firstly, we note that each break occupies two consecutive levels in the stack, and different breaks occupy different levels. Therefore, there are no more than $\lfloor \frac{|I(q)|}{2} \rfloor$ breaks in stack q. Secondly, there exists an optimal solution to (V) with at most $\lceil a \rceil$ breaks. Indeed, let w^* be an optimal solution of (V) with $\beta^* > \lceil a \rceil$ breaks and let l^* be the level of the topmost break. If we increase the weight of the item at level l^* by $\Delta = \overline{w}_{[l^*]} - \underline{w}_{[l^*]}$, the payload violation at level l^* decreases by at most $a\Delta$. However, there are at least $\beta^* - 1$ more payload violations beneath level l^*, which result in an increase of $v_q(w)$ by at least $(\beta^* - 1)\Delta$ (see Figure 1, right). Due to $\beta^* > \lceil a \rceil$, the total payload violation could be increased. Therefore, we can remove the break at level l^* without decreasing the violation $v_q(w)$. Repeating this argument until the next break level, we find that there is an optimal solution with at most $\lceil a \rceil$ breaks.

We now count the number of possible scenarios with $\beta \in \{1, \dots, \delta\}$ breaks. In such a scenario, between any two consecutive breaks there must be exactly one anti-break. Therefore, each of such scenarios corresponds to a choice of $2\beta - 1$ levels for β breaks together with $\beta - 1$ anti-breaks in between. Since stack q contains $|I(q)|$ items, there are $|I(q)| - 1$ levels that can have breaks or anti-breaks. This leads to $\binom{|I(q)|-1}{2\beta-1}$ possible scenarios having β breaks. Enumerating all these possibilities for $\beta = 1, 2, \dots, \delta$ gives $\mathcal{O}(|I(q)|^{2\delta-1})$ possible scenarios that have to be tested. □

Note that if a is a fixed value, the complexity $\mathcal{O}(|I(q)|^{2\delta-1})$ is polynomially bounded in the input length of the problem and hence (V) can be solved in polynomial time.

Due to Theorem 3, Step 6 of Algorithm 1 can alternatively be realized by enumerating all relevant item weights per level. However, the result can also be used to avoid the iterative algorithm and to formulate a compact model that includes all relevant scenarios directly. Note that these are not scenarios in the sense that a specific item gets some weight; instead, we assign to a specific level either the lower or upper weight of an item. The result is a formulation similar to the one used in Section 3.1.

We present this compact formulation of (SRI) for the case $a \leq 1$ where we know that for each stack only a single break has to be considered. Hence, only $|I(q)| - 1$ scenarios are relevant for stack q, where in the solution for scenario k the break occurs at level k. We introduce auxiliary variables $w_{ql}^k \geq 0$ denoting the weight of the item in stack q at level l in the solution for scenario k with a break at level k. Modifying the formulation (SRF), we get the following MIP formulation for (SRI):

$$\min \sum_{q \in Q} v_q \tag{27}$$

$$\text{s.t.} \quad \sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 \qquad \forall i \in I \tag{28}$$

$$\sum_{i \in I} x_{iql} \leq 1 \qquad \forall q \in Q, l \in L \tag{29}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x_{jq,l-1} \geq x_{iql} \qquad \forall i \in I, q \in Q, l \in L \setminus \{1\} \tag{30}$$

$$w_{ql}^k = \sum_{i \in I} \underline{w}_i x_{iql} \qquad \forall q \in Q, k, l \in L, l \leq k \tag{31}$$

$$w_{ql}^k = \sum_{i\in I} \overline{w}_i x_{iql} \qquad \forall q \in Q, k, l \in L, l > k \tag{32}$$

$$\sum_{h>l} w_{qh}^k - a w_{ql}^k \leq v_{ql}^k \qquad \forall q \in Q, k, l \in L \setminus \{b\} \tag{33}$$

$$\sum_{l\in L\setminus\{b\}} v_{ql}^k \leq v_q \qquad \forall q \in Q, k \in L\setminus\{b\} \tag{34}$$

$$x_{iql} \in \{0,1\} \qquad \forall i \in I, q \in Q, l \in L \tag{35}$$

$$v_{ql}^k \geq 0 \qquad \forall q \in Q, k, l \in L \setminus \{b\} \tag{36}$$

$$w_{ql}^k \geq 0 \qquad \forall q \in Q, k, l \in L \tag{37}$$

$$v_q \geq 0 \qquad \forall q \in Q \tag{38}$$

In the solution for scenario k, all items up to level k are assumed to be as light as possible (constraints (31)), while all items at higher levels are as heavy as possible (constraints (32)). The payload violation in the solution for scenario k for stack q and level l is measured by the variable v_{ql}^k in constraints (33). The worst-case over all scenarios is computed with the help of the variables v_q and constraints (34).

For $a > 1$ a similar formulation can be used by enumerating all possible scenarios via the number of breakpoints, using $\mathcal{O}(b^{2\delta-1})$ relevant scenarios per stack. We present the formulation for $a \leq 2$ in Appendix B.

Note that in the case of hard payload constraints (i.e., no payload violations are allowed which implies that the total payload violation must be equal to zero), a more compact formulation can be given. For this, we consider the single robust payload constraint

$$\sum_{j\in I} \sum_{h=l+1}^{b} w_j x_{jqh} - a \sum_{i\in I} w_i x_{iql} \leq 0 \qquad \forall q \in Q, l \in L \setminus \{b\}, w \in \mathcal{U}^I$$

For a fixed location $(q,l) \in Q \times (L \setminus \{b\})$, the first term of the left hand side in this inequality is equal to the total weight of all items stored above this location, while the sum in the second term equals the weight of the item stored at the location. The maximum difference over all $w \in \mathcal{U}^I$ between the former and the latter term is therefore attained when the item stored at the location has minimum weight, while the items stored above have maximum weights. As we have pointed out, there exists a worst-case scenario $w \in \mathcal{U}^I$ which dominates all other scenarios from $\mathcal{U}^I$ with respect to this constraint. Hence, this robust payload constraint can be equivalently written as

$$\sum_{j\in I} \sum_{h=l+1}^{b} \overline{w}_j x_{jqh} - a \sum_{i\in I} \underline{w}_i x_{iql} \leq 0 \qquad \forall q \in Q, l \in L \setminus \{b\} \tag{39}$$

Now, any stacking solution has zero payload violation if and only if all these worst-case inequalities are fulfilled, i.e., the robust counterpart of the problem is given by (1)-(4),(39),(6).

4. Adjustable robustness

Following the ideas first introduced in [9], we now consider a robust model where not all stacking decisions need to be fixed in advance, but some can be made after the realized scenario becomes known. In our setting, we follow the idea that a planner needs to determine in advance to which stack an item is assigned ("here-and-now" decision); however, he is allowed to choose the level of the item within the stack depending on the weight scenario of all items later ("wait-and-see" decision). This gives the planner more flexibility in his decision making and potentially better results with less payload violations.

Such a setting occurs in practice if special subareas (stacks) must be reserved for the items in advance (for example, according to the different destination of the items). As another example we refer to the following setting from [41]: In the hatch overstow problem, containers need to be loaded onto a ship with several hatches, where different areas of the ship have to be filled separately. In our setting, in the first stage we assign containers to subsets of stacks in the terminal (corresponding to the subareas on the ship). In the second stage, these subsets are then loaded onto the ship, and the precise locations of the items in the stacks of the corresponding subarea are determined using the weights that are now known. For the sake of simplicity, we restrict our presentation to the setting that items are assigned to single stacks in the first stage (and not to subsets of stacks). However, models and solution algorithms can be extended to this setting (see Appendix C).

In Section 4.1, we first consider finite uncertainty sets before the more complex interval-based models are discussed in Section 4.2.

4.1. Finite uncertainty

As in Section 3, we would like to optimize the worst-case performance of a stacking solution over all possible weight realizations. There are two kinds of decisions that need to be made: Here-and-now decisions independent of realized item weights, which determine for every item the stack it is assigned to; and wait-and-see decisions depending on the scenario, which decide the level of each item at which it should be stored.

We introduce binary here-and-now variables z_{iq} for $i \in I, q \in Q$ where $z_{iq} = 1$ if item i is assigned to stack q. Furthermore, wait-and-see variables x^k_{iql} depend on the realized scenario k and determine if item i is stored in stack q at level l in the solution corresponding to scenario k. Then the problem can be formulated as follows.

$$\text{(ARF)} \quad \min\ v \tag{40}$$

$$\text{s.t.} \sum_{q \in Q} z_{iq} = 1 \qquad \forall\, i \in I \tag{41}$$

$$\sum_{i \in I} z_{iq} \le b \qquad \forall\, q \in Q \tag{42}$$

$$\sum_{q \in Q} \sum_{l \in L} x^k_{iql} = 1 \qquad \forall\, i \in I, k \in \mathcal{N} \tag{43}$$

$$\sum_{i \in I} x^k_{iql} \le 1 \qquad \forall\, q \in Q, l \in L, k \in \mathcal{N} \tag{44}$$

$$\sum_{l \in L} x^k_{iql} = z_{iq} \qquad \forall\, q \in Q, i \in I, k \in \mathcal{N} \tag{45}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x^k_{jq,l-1} \ge x^k_{iql} \qquad \forall\, i \in I, q \in Q, l \in L \setminus \{1\}, k \in \mathcal{N} \tag{46}$$

$$\sum_{j \in I} \sum_{h=l+1}^{b} w^k_j x^k_{jqh} - a \sum_{i \in I} w^k_i x^k_{iql} \le v^k_{ql} \qquad \forall\, q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} \tag{47}$$

$$\sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v^k_{ql} \le v \qquad \forall\, k \in \mathcal{N} \tag{48}$$

$$z_{iq} \in \{0, 1\} \qquad \forall\, i \in I, q \in Q \tag{49}$$

$$x_{iql}^k \in \{0,1\} \qquad \forall\, i \in I, q \in Q, l \in L, k \in \mathcal{N} \tag{50}$$

$$v_{ql}^k \geq 0 \qquad \forall\, q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} \tag{51}$$

$$v \geq 0 \tag{52}$$

Constraints (41) model that every item has to be assigned to some stack, while every stack contains at most b items (constraints (42)). Constraints (43) ensure that also for each scenario every item has to be assigned to exactly one stack and level in the corresponding solution, while constraints (44) restrict the number of items at any location to be at most one. We couple the here-and-now variables z_{iq} with the wait-and-see variables x_{iql}^k in constraints (45): If the here-and-now decisions assign item i to stack q, then also in every scenario k item i has to be assigned to some level l in stack q. Constraints (46)-(48) are used to model the stacking constraints and to compute the payload violations.

Note that this formulation is more sensitive to the number of scenarios than the strictly robust model, with $\mathcal{O}(Nnmb)$ variables and $\mathcal{O}((N+n)mb)$ constraints.

4.2. Interval uncertainty

We now consider the adjustable robust problem with interval uncertainty, denoted as (ARI). As in Section 3.2, we follow the idea of an iterative approach that considers relaxed problems with finite uncertainty sets, and a subproblem to generate worst-case scenarios. This subproblem now becomes more complex, since the worst-case generation also needs to take the possible wait-and-see decisions into account.

According to the here-and-now decisions all items are assigned to stacks, but their ordering in the stacks (i.e., the assignment to levels) is determined later when the weights of all items are known. To this end, we consider the following two subproblems for a single stack:

1. Given a subset of possible item orderings, we search for a worst-case weight scenario that maximizes the smallest total payload violation over all orderings.
2. Given a weight scenario, we search for an item ordering that minimizes the total payload violation.

Both subproblems are iteratively solved, until their objective values coincide.

We consider a fixed stack $q \in Q$ and assume that the set $I(q)$ contains all items which are assigned to q according to the here-and-now decisions z_{iq}. Furthermore, let $L(q) := \{1, \ldots, |I(q)|\}$ and $L'(q) := L(q) \setminus \{|I(q)|\}$. Let $\mathcal{P}(I_q)$ be the set of all permutations for the items in the set $I(q)$, each permutation describing an assignment of all items to levels in the stack. Furthermore, we denote by $v_q(\pi, w)$ the total payload violation for stack q with respect to the weights w if the items in $I(q)$ are ordered according to the permutation π.

In the first subproblem, for a given subset $\mathcal{P}'(I_q) \subseteq \mathcal{P}(I_q)$ of permutations for the items in stack q we search for a worst-case weight scenario $w \in \mathcal{U}^I$ that is a solution of problem

$$\text{(Max-}w\text{)} \qquad \max_{w \in \mathcal{U}^I} \min_{\pi \in \mathcal{P}'(I_q)} v_q(\pi, w).$$

In the following, we assume that the set $\mathcal{P}'(I_q)$ contains K permutations $\pi^1, \ldots, \pi^K$ which are described by binary values p_{il}^k with $p_{il}^k = 1$ if and only if in permutation π^k item $i \in I(q)$ is assigned to level $l \in L(q)$. Let $\mathcal{K} := \{1, \ldots, K\}$.

To solve problem (Max-w) for all $i \in I(q)$, we introduce continuous variables $w_i \in [\underline{w}_i, \overline{w}_i]$ determining the weight of item i. Additionally, we have variables $v_l^k \geq 0$ measuring the payload violation for the item assigned to level l in the permutation π^k and binary auxiliary variables α_l^k

to determine whether there is a payload violation at level l in the permutation π^k or not. Finally, the auxiliary variable $v \geq 0$ denotes the total payload violation of the stack. Then problem (Max-w) can be formulated as follows.

$$\begin{aligned}
& \max\ v && && (53)\\
\text{s.t.}\ & \sum_{l \in L'(q)} v_l^k \geq v && \forall k \in \mathcal{K} && (54)\\
& \left(\sum_{h>l} \sum_{i \in I(q)} p_{ih}^k w_i - a \sum_{i \in I(q)} p_{il}^k w_i \right) \alpha_l^k = v_l^k && \forall k \in \mathcal{K}, l \in L'(q) && (55)\\
& \underline{w}_i \leq w_i \leq \overline{w}_i && \forall i \in I(q) && (56)\\
& \alpha_l^k \in \{0,1\} && \forall k \in \mathcal{K}, l \in L'(q) && (57)\\
& v_l^k \geq 0 && \forall k \in \mathcal{K}, l \in L'(q) && (58)\\
& v \geq 0 && && (59)
\end{aligned}$$

According to (53) the total payload violation of the stack is maximized. Constraints (54) ensure that v equals the smallest payload violation over all item permutations. Due to (55), the payload violations are correctly computed. Finally, (56) guarantees that the weight variables w_i are contained in the given intervals.

Note that constraints (55) are non-linear, due to the product of α with w. To remove this non-linearity, we introduce new variables $\beta_{il}^k = w_i \alpha_l^k$ and require $0 \leq \beta_{il}^k \leq M\alpha_l^k$ and $w_i - M(1-\alpha_l^k) \leq \beta_{il}^k \leq w_i$ for a suitable large constant M (note that $M \geq \overline{w}_i$ suffices). This gives rise to the following new formulation:

$$\begin{aligned}
(\text{Max-}w)\quad & \max\ v && \\
\text{s.t.}\ & \sum_{l \in L'(q)} \sum_{h>l} \sum_{i \in I(q)} p_{ih}^k \beta_{il}^k - a \sum_{l \in L'(q)} \sum_{i \in I(q)} p_{il}^k \beta_{il}^k \geq v && \forall k \in \mathcal{K}\\
& \beta_{il}^k \leq \overline{w}_i \alpha_l^k && \forall k \in \mathcal{K}, l \in L'(q), i \in I(q)\\
& w_i + \overline{w}_i(\alpha_l^k - 1) \leq \beta_{il}^k \leq w_i && \forall k \in \mathcal{K}, l \in L'(q), i \in I(q)\\
& \underline{w}_i \leq w_i \leq \overline{w}_i && \forall i \in I(q)\\
& \alpha_l^k \in \{0,1\} && \forall k \in \mathcal{K}, l \in L'(q)\\
& \beta_{il}^k \geq 0 && \forall k \in \mathcal{K}, i \in I(q), l \in L'(q)\\
& v \geq 0 &&
\end{aligned}$$

We denote this problem as (AV1, $I(q)$, $\mathcal{P}'$) and its optimal objective value by $\overline{v}_q(\mathcal{P}')$.

We now consider the second subproblem, which consists of finding a (possibly new) permutation for the items in stack q minimizing the total payload violation with respect to the current weights w, i.e., a solution of problem

$$(\text{Min-}\pi) \qquad \min_{\pi \in \mathcal{P}(I_q)} v_q(\pi, w).$$

For $i \in I(q), l \in L(q)$ we introduce binary variables p_{il} determining the item permutation, i.e., $p_{il} = 1$ if and only if item i is assigned to level l. Variables $v_l \geq 0$ for $l \in L'(q)$ are used to determine the payload violation at level l. Then, problem (Min-π) can be formulated as follows.

$$(\text{Min-}\pi) \quad \min \sum_{l \in L'(q)} v_l \qquad (60)$$

$$\text{s.t.} \sum_{\substack{h\in L(q)\\h>l}} \sum_{j\in I(q)} w_j p_{jh} - \sum_{i\in I(q)} a w_i p_{il} \leq v_l \qquad \forall l \in L'(q) \tag{61}$$

$$\sum_{i\in I(q)} p_{il} = 1 \qquad \forall l \in L(q) \tag{62}$$

$$\sum_{l\in L(q)} p_{il} = 1 \qquad \forall i \in I(q) \tag{63}$$

$$p_{i,l+1} + p_{jl} \leq 1 \qquad \forall i,j \in I(q) \text{ with } s_{ij} = 0, l \in L'(q) \tag{64}$$

$$p_{il} \in \{0,1\} \qquad \forall i \in I(q), l \in L(q) \tag{65}$$

$$v_l \geq 0 \qquad \forall l \in L'(q) \tag{66}$$

Due to (60) the total payload violation is minimized. Constraints (61) are used to determine the violations v_l at the different levels. The assignment constraints (62) and (63) ensure that at each level exactly one item is stored and that each item is assigned to exactly one level, respectively. Due to constraints (64) the item permutation is feasible with respect to the stacking constraints s_{ij}. We denote this problem as (AV2, $I(q)$, w) and its optimal objective value by $v^*_q(w)$.

These two subproblems are then solved, until their objective values coincide and the worst possible total payload of a fixed stack assignment is determined, which also yields a new worst-case weight scenario. This scenario is added to the main adjustable problem, and the process is repeated. We summarize this approach in Algorithm 2.

Algorithm 2 Exact Algorithm for (ARI)

```
Require: An instance of (ARI).
 1: $k \leftarrow 0$
 2: Take an arbitrary scenario $w^1 \in \mathcal{U}^I$ and let $\mathcal{U}^1 \leftarrow \{w^1\}$.
 3: repeat
 4:     $k \leftarrow k + 1$
 5:     Solve (ARF) with uncertainty set $\mathcal{U}^k$. Let $x^k$ be the resulting stacking solution and $LB^k$
        its objective value.
 6:     $UB^k \leftarrow 0$
 7:     for all $q \in Q$ do
 8:         $\ell \leftarrow 0$
 9:         Let $I(q)$ be the set of items assigned to stack $q$ in $x^k$.
10:         $\mathcal{P}^1 \leftarrow \{\pi^1\}$ for some permutation $\pi^1$ of all items in $I(q)$ which is feasible w.r.t. $s_{ij}$.
11:         repeat
12:             $\ell \leftarrow \ell + 1$
13:             Solve (AV1, $I(q)$, $\mathcal{P}^\ell$).
14:             Let $w$ be the resulting item weights and $\overline{v}_q(\mathcal{P}^\ell)$ the resulting objective value.
15:             Solve (AV2, $I(q)$, $w$).
16:             Let $\pi^\ell$ be the resulting item permutation, and $v^*_q(w)$ the resulting objective value.
17:             $\mathcal{P}^{\ell+1} \leftarrow \mathcal{P}^\ell \cup \{\pi^\ell\}$
18:         until $\overline{v}_q(\mathcal{P}^\ell) = v^*_q(w)$
19:         $UB^k \leftarrow UB^k + v^*_q(w)$
20:         $w^k_i \leftarrow w_i$ for all items $i \in I(q)$
21:     end for
22:     $\mathcal{U}^{k+1} \leftarrow \mathcal{U}^k \cup \{w^k\}$
23: until $UB^k = LB^k$
24: return optimal stacking solution $x^k$
```

4.3. Heuristic solution approaches

In the following, we present three heuristics to solve (ARI). Each is provided with a feasible starting solution, whose generation is explained at the end of this section.

4.3.1. Pattern generation

In our first approach, we generate a set of candidate patterns (corresponding to subsets of items assigned to the same stack). To this end, we consider an extended problem formulation for (ARI). Let $\mathcal{C}$ denote all possible subsets $C \subseteq I$ of items that are feasible with respect to the stacking constraints and the stack capacity b (i.e., these items can be assigned together to the same stack). We introduce a binary variable y_C for each such subset C to decide if subset C is used. For each $C \in \mathcal{C}$, let

$$v(C) = \max_{w \in \mathcal{U}^I} \min_{\pi \in \mathcal{P}(C)} v(\pi, w)$$

be the worst-case payload violation of the set C (which can be determined using (AV1) and (AV2)).

In the following, χ_j^C indicates whether $j \in I$ is contained in C with $\chi_j^C = 1$ if and only if $j \in C$, and zero otherwise. Then, we get the following equivalent formulation (ARI-PG) for (ARI) based on the concept of pattern generation:

$$\text{(ARI-PG)} \quad \min \sum_{C \in \mathcal{C}} v(C) y_C \tag{67}$$

$$\text{s.t.} \sum_{C \in \mathcal{C}} \chi_j^C y_C = 1 \qquad \forall j \in I \tag{68}$$

$$\sum_{C \in \mathcal{C}} y_C \leq m \tag{69}$$

$$y_C \in \{0,1\} \qquad \forall C \in \mathcal{C} \tag{70}$$

Constraints (68) ensure that every item is contained in one pattern, and constraint (69) bounds the number of available patterns.

As there are potentially exponentially many relevant sets $C \in \mathcal{C}$, we use the following algorithm to solve (ARI-PG) heuristically. We keep a working set of patterns, which is initially filled with the patterns that are given by some feasible starting solution. In every iteration, we generate a set $\mathcal{C}'$ of subsets $C \subseteq I$ at random, where the cardinality of each such subset C is chosen between 1 and b in a way that the expected cardinality equals n/m. These candidate patterns are made up of new, random patterns, by the working set of patterns, and by randomly merging patterns from the working set. Items that have once been part of the working set are never removed. Hence, we generate new patterns in the fashion of a genetic algorithm.

We evaluate each subset using (AV1) and (AV2) iteratively to compute $v(C)$. Then, we solve (ARI-PG) using the heuristic set $\mathcal{C}'$ instead of the full set $\mathcal{C}$. Patterns of the resulting solution that are not part of the working set yet are added to it. We then begin with the next algorithm iteration by generating new patterns.

Note that problems of type (ARI-PG) are easy to solve with current commercial MIP solvers (constraints (68) are special ordered set constraints, and constraint (69) is a single knapsack constraint), which means that a large number of sets C can be used in the heuristic with still small computation times. Thus, one can expect this approach to give better solutions than Algorithm 2 within the same amount of time; however, no quality bounds (or even a proof of optimality) are produced.

4.3.2. Local search

In our second approach, we follow an iterative improvement local search heuristic. Given a feasible assignment of items to stacks, we consider the following two moves: An item is moved from one stack to another stack with sufficient capacity; or two items from two stacks are swapped. All feasible moves are evaluated in random order (again, by solving (AV1) and (AV2) iteratively) until an improving move is found, which is then performed. The process is repeated until either a local optimum or some time limit is reached.

4.3.3. Destroy-and-repair heuristic

Finally, in our third heuristic, we follow a destroy-and-repair approach, which can also be considered as a variable neighborhood heuristic.

Given a feasible assignment, we randomly select one stack with a high objective value and one stack with a low objective value. We then solve a problem of type (ARI) restricted to only these two stacks, i.e., we find a new, optimal assignment for these two stacks using the iterative procedure described in Algorithm 2. This process is repeated until a time limit is reached.

To increase the number of iterations, we limit the time for a repair operation by stopping Algorithm 2 once a gap of 3% is reached or ten seconds have elapsed, whichever comes first. We also provide the solver with the current stack order as a starting solution.

Note that the neighborhood this algorithm investigates is larger than for the local search, i.e., every move the local search evaluates is contained in the set of possible moves for this destroy-and-repair heuristic. However, every iteration is computationally more costly, as we do not only evaluate an assignment of items to two stacks, but also find a best possible assignment.

4.3.4. Constructive heuristic

To provide the heuristics from above with a feasible starting solution, one could simply solve the nominal problem. As this turns out to be too computationally difficult for large-scale problems, we present a different model where the payload violation is not considered.

This model is inspired by network flows. Consider a directed graph, where there exists one node i for every item $i \in I$, and an arc (i, j) connecting $i, j \in I$ iff $s_{ij} = 1$. Every arc can carry an integer amount of flow between 0 and b. The topmost item within a stack receives b unit of flow, and in every subsequent node, loses one unit of flow. Every node needs to be provided with at least one unit of flow.

We use variables $x_{ij} \in \{0, \dots, b\}$ to denote the flow along arc (i, j), binary variables z_{ij} to model if arc (i, j) carries any flow, and binary variables y_i to determine whether item i is the topmost item of a stack. The corresponding feasibility problem can be formulated as follows:

$$\sum_{i \in I} y_i \leq m \tag{71}$$

$$\sum_{\substack{j \in I \\ s_{ji}=1}} x_{ji} - \sum_{\substack{j \in I \\ s_{ij}=1}} x_{ij} + b y_i \geq 1 \qquad \forall\, i \in I \tag{72}$$

$$x_{ij} \leq b z_{ij} \qquad \forall i, j \in I \tag{73}$$

$$\sum_{\substack{j \in I \\ s_{ij}=1}} z_{ij} \leq 1 \qquad \forall i \in I \tag{74}$$

$$\sum_{\substack{j \in I \\ s_{ji}=1}} z_{ji} \leq 1 \qquad \forall i \in I \tag{75}$$

$$x_{ij} \in \{0,\dots,b\} \qquad \forall i,j \in I : s_{ij} = 1 \tag{76}$$

$$z_{ij} \in \{0,1\} \qquad \forall i,j \in I : s_{ij} = 1 \tag{77}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{78}$$

Constraints (71) ensure that at most m stacks can be used, while constraints (72) model the integer flow. Constraints (73) are used to determine whether an arc is used or not, while constraints (74) and (75) ensure that every node has at most one predecessor and at most one successor. Note that a solution found this way is feasible for (ARI), as only the payload violation is ignored, which is part of the objective.

We solve this problem once using a MIP solver to provide any of the three heuristics from above with a feasible starting solution.

5. Computational experiments

To test the performance of the models and algorithms introduced in this paper, we performed four experiments with different sets of instances. The first three experiments use smaller instances with up to 30 items to compare exact and heuristic algorithms, and to analyze the impact of different parameters. The fourth experiment considers heuristic solutions for larger instances with up to 300 items.

5.1. Small instances

5.1.1. Setup

Recall that an uncertain stacking problem is parameterized by: The number of items n, the number of available stacks m, the maximum height of stacks b, and the payload violation parameter a. Additionally, a stacking matrix S is required as well as either an interval-based uncertainty $\mathcal{U}^I$ or a finite uncertainty set $\mathcal{U}^F$.

We randomly generated stacking matrices S by using a density parameter $d \in [0,1]$, which is the relative number of ones within the non-diagonal elements of the matrix (i.e., for $d = 1$, all items can be stacked onto each other, and for $d = 0.5$, there are $n(n-1)/2$ randomly distributed allowed pairings). Furthermore, we generated lower and upper bounds $\underline{w}_i$ and $\overline{w}_i$ on item weights in the following way: There are two types of items, which both occur with the same probability. The first type of items has $\underline{w}_i \in [9,10]$ and $\overline{w}_i \in [10,11]$; the second type of items has $\underline{w}_i \in [0,10]$ and $\overline{w}_i \in [10,20]$. Thus, the expected average of lower and upper bound is 10 in both cases, but the variance is different. This reflects the case that items have on average a similar weight, but different variance. For finite uncertainty sets, we sample scenarios uniformly.

For the three experiments, we modified problem parameters such that the first experiment considers few high stacks, and the second experiment considers many small stacks. The third experiment, which is described in the appendix, analyzes the impact of the payload parameter a.

For each parameter choice, we generated 20 instances (all of them were noted to be feasible). For each instance, we solve:

- The nominal model, where nominal weights are the midpoints of the respective intervals. We refer to this solution as "Nom".
- The strictly robust model with finite uncertainty, sampling 10 and 20 scenarios; we denote these solutions as "S-10" and "S-20", respectively.

- The strictly robust model with interval-based uncertainty using Algorithm 1. This is denoted as “SI”.
- The strictly robust model with interval-based uncertainty using the compact model. This is denoted as “SIC”.
- The adjustable model with finite uncertainty, sampling 5 and 10 scenarios; we denote these solutions as “A-5” and “A-10”, respectively.
- The adjustable model with interval-based uncertainty using the exact Algorithm 2. This is denoted as “AI”.
- The adjustable model with interval-based uncertainty, based on the formulation (ARI-PG), using the pattern generation heuristic with 1,000 candidate sets. We denote this heuristic solution as “AIPG”.
- The adjustable model with interval-based uncertainty using local search, which is denoted as “AILS”.
- The adjustable model with interval-based uncertainty using the destroy-and-repair heuristic, which is denoted as “AIDR”.

We used CPLEX v.12.6 ([26]) to solve all MIPs. All experiments were conducted on a computer with a 16-core Intel Xeon E5-2670 processor, running at 2.60 GHz with 20MB cache, and Ubuntu 12.04. Processes were pinned to one core. To restrict computation times, a time limit of 15 minutes was imposed on every solution approach.

5.1.2. Few high stacks

In this experiment, we fix $a = 1$, $d = 0.5$ and $m = 3$, and vary the number of items n from 9 to 30 in steps of 3. The stack height b is equal to $n/3$, that is, this experiment considers relatively few but high stacks.

We present the median computation times (in seconds) in Table 1. Nominal solution times are very small, while the iterative approaches hit the time limit of 15 minutes already for small n. In particular, comparing SI and SIC shows that the compact model for strict robustness is considerably more efficient than the iterative approach, and needs only slightly longer computation times than the nominal model.

n	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI
9	0.0	0.1	0.2	4.3	0.0	0.4	1.4	24.5
12	0.0	0.9	1.4	229.0	0.1	2.3	8.4	900.0
15	0.1	4.1	14.0	900.0	0.2	4.9	29.0	900.0
18	0.1	14.0	36.7	900.0	0.3	12.0	253.3	900.0
21	0.2	27.7	58.5	900.0	0.5	47.1	551.4	900.0
24	0.3	73.3	167.4	900.0	0.7	99.1	889.4	900.0
27	0.4	62.7	391.5	900.0	1.6	405.5	900.0	900.0
30	0.6	147.5	900.0	900.0	1.9	583.9	900.0	900.0

Table 1: Experiment 1, median computation times in seconds.

Comparing S-10 with S-20 and A-5 with A-10, we note that already a small increase in scenarios results in a large increase in computation times.

We now consider strict objective values, i.e., the worst-case payload violation when stacks cannot be adjusted. Note that strict objective values are not well-defined for adjustable solutions, as they do not specify a complete stacking in their first-stage. The average gaps for strict objective values are presented in Table 2, which are computed as $(UB - LB)/UB$ using the best-known lower bound on every instance (which is the objective value of SIC, as it solves all instances to optimality).

n	Nom	S-10	S-20	SI	SIC
9	5.07	6.65	4.92	0.00	0.00
12	17.12	8.85	7.21	0.00	0.00
15	17.62	10.09	6.87	0.93	0.00
18	14.32	9.04	6.14	2.91	0.00
21	12.83	9.55	7.21	4.76	0.00
24	10.13	8.14	5.86	4.83	0.00
27	12.28	8.06	6.32	4.64	0.00
30	12.39	7.80	7.34	3.87	0.00

Table 2: Experiment 1, average gaps for strict objective values in percent.

Values are given as percentages; i.e., the nominal solution has an average gap of 12.39% on instances with size $n = 30$. Note that the iterative approach SI still produces solutions that are close to optimality. The increased number of scenarios for S-20 results in better solutions than for S-10, which is in turn better than the nominal solution. Note that even though the nominal solution takes comparatively little computational effort, it leads to large gaps in the worst-case, underlining the value of using a robust approach here.

The average gaps for adjustable objective values are presented in Table 3. Adjustable objective values are given as the worst-case payload violation when only the assignment of items to stacks is fixed (i.e., it can also be computed for the nominal and strict solutions). As a lower bound, we used the largest lower bound produced by AI. Overall gaps are relatively small, and solutions tend to perform better with larger n. This is because m is kept constant, meaning that more items are assigned to the same number of stacks, which increases the chances to correct mistakes made in the assignment during the recovery step.

n	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI	AIPG	AILS	AIDR
9	16.69	15.84	13.29	15.66	13.60	6.40	4.55	0.00	0.00	0.16	0.16
12	8.58	8.34	7.81	4.76	4.75	6.36	5.45	1.95	1.86	2.46	2.09
15	4.70	5.37	4.71	3.76	3.58	4.24	3.82	2.09	1.78	2.16	2.07
18	5.44	4.33	3.82	3.48	2.72	4.63	4.78	2.24	1.82	2.09	1.90
21	3.73	3.64	3.01	3.07	1.73	3.68	3.00	1.30	1.32	1.16	1.20
24	2.25	2.75	2.73	1.87	1.21	2.26	2.19	1.01	1.12	0.87	0.86
27	2.35	2.13	2.32	1.70	0.92	2.52	2.01	0.86	1.03	0.73	0.75
30	2.27	2.55	2.33	2.02	1.10	2.37	2.30	1.28	1.42	0.95	1.04

Table 3: Experiment 1, average gaps for adjustable objective values in percent.

The heuristics and AI perform best, but also SIC performs well. As for strict objective values, an increased number of scenarios for the sampling algorithms improves their performance.

5.1.3. Many small stacks

In this second experiment, we fix $a = 1$, $d = 0.5$ and $b = 3$, and vary the number of items n from 9 to 30 in steps of 3. The number of stacks m is equal to $n/3$, that is, this experiment considers relatively many but small stacks.

We present the counterparts to Tables 1, 2 and 3 as Tables 4, 5 and 6.

n	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI
9	0.0	0.1	0.2	4.9	0.0	0.4	1.4	24.4
12	0.0	0.7	2.0	182.1	0.1	1.6	9.1	900.0
15	0.0	3.2	11.1	900.0	0.1	2.7	15.2	900.0
18	0.1	50.7	272.2	900.0	0.1	5.0	34.9	900.0
21	0.1	619.0	900.0	900.0	0.2	11.2	60.1	900.0
24	0.1	900.0	900.0	900.0	0.3	22.3	150.8	900.0
27	0.2	900.0	900.0	900.0	0.4	39.9	225.8	900.0
30	0.2	900.0	900.0	900.0	0.5	61.9	435.2	900.0

Table 4: Experiment 2, median computation times in seconds.

Note that computation times for the strict models increased compared to experiment 1. This is also reflected in the solution gaps. For strict objective values, gaps are considerably larger than before; in particular the iterative approach SI is not competitive anymore, while SIC still solves all instances to optimality in a short amount of time.

n	Nom	S-10	S-20	SI	SIC
9	5.07	6.65	4.92	0.00	0.00
12	8.91	6.50	5.64	0.00	0.00
15	9.31	8.10	7.28	0.97	0.00
18	8.63	8.38	5.82	6.65	0.00
21	7.15	8.19	5.62	10.24	0.00
24	7.33	6.75	6.03	9.05	0.00
27	7.81	8.17	5.03	17.69	0.00
30	8.78	8.24	7.21	21.00	0.00

Table 5: Experiment 2, average gaps for strict objective values in percent.

Also the adjustable objective gaps increase, and show larger differences between the solution approaches than before. On these instances, the heuristic approaches tend to perform best; in particular AIPG outperforms all other algorithms. In this setting, SIC is not competitive anymore. The nominal solution performs poorly in both strict and adjustable objective values.

n	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI	AIPG	AILS	AIDR
9	16.69	15.84	13.29	15.66	13.60	6.40	4.55	0.00	0.00	0.01	0.15
12	20.84	19.76	20.87	15.68	17.46	13.55	10.72	4.42	4.04	5.76	5.73
15	30.23	21.95	27.77	26.23	23.40	20.56	14.99	10.70	9.41	10.68	10.57
18	35.91	33.00	32.93	36.87	33.86	28.76	26.12	21.06	17.86	19.55	19.32
21	35.56	32.06	30.90	33.59	31.45	26.50	21.55	18.54	15.40	16.71	16.57
24	31.54	29.76	28.03	27.09	26.50	22.33	19.08	15.61	12.65	13.61	13.45
27	32.69	33.55	33.02	34.32	31.44	26.39	23.60	19.43	15.74	16.63	16.67
30	37.29	39.76	36.24	41.63	34.22	30.69	28.81	25.31	20.19	20.77	20.86

Table 6: Experiment 2, average gaps for adjustable objective values in percent.

5.1.4. Summary for small instances

We summarize the differences between the results of experiments 1 and 2 from a more general perspective. We note that for experiment 1, where few but large stacks are used, all solutions tend to perform relatively well for adjustable robustness. This is because there are more possibilities to rearrange items once the scenario becomes known, and the first-stage decision where to put items becomes less important.

However, when many but smaller stacks are used as in experiment 2, adjustable objective values significantly differ between solution approaches, which shows the increased potential of using an adjustable approach.

This is not the case for strict robustness, where we note that the nominal solution performs worse when there are few but high stacks, and better when there are many but low stacks.

Thus, from a practical perspective, it depends on the instance which approach to follow makes most sense. For few and high stacks, the planner should be concerned about strict objective values, but is fine using a nominal solution if adjustments are possible. On the other hand, if there are many and low stacks, the nominal solution is fine if no recovery is possible, but more effort needs to be taken if adjustments are possible.

5.2. Large instances

In this experiment, we tested the performance of our heuristic algorithms on larger datasets. The considered instance sizes are given in Table 7, they were chosen such that $m \cdot b = 1.2n$, i.e., there is always a spare capacity of 20% available. Other parameters were generated as described in Section 5.1.1, using $d = 0.6$ and $a = 1$.

n	m	b
100	30	4
125	38	4
150	36	5
175	42	5
200	48	5
225	54	5
250	50	6
275	55	6
300	60	6

Table 7: Experiment 4, instance sizes.

We use the heuristic methods AIPG, AILS and AIDR with a time limit of 15 minutes, and record their adjustable objective values. All heuristics are provided with a feasible starting solution by solving model (71)-(78), which is denoted as “Start”. To better evaluate their quality, a lower bound is computed for each instance by solving the nominal problem without stacking matrix constraints.

Resulting average gaps for adjustable objective values are shown in Table 8, in the left columns.

While AIPG showed the most promising performance for smaller instances, it tends to be outperformed by AILS and AIDR on larger instances. Note that the gap is relatively unaffected by the number of items n, but depends more on the stack size b. This is because all three heuristic methods need to evaluate stacks, and the required computation time to evaluate a single stack scales with b. Hence, the improvements the heuristics can make compared to the feasible starting solution are reduced with increased b within the time limit.

Overall, gaps tend to decrease with larger b. This can be explained as in the comparison between experiments 1 and 2, where larger stacks indicate that the error made in the first decision stage tend to be less important.

	Gap				To best			
n	Start	AIPG	AILS	AIDR	Start	AIPG	AILS	AIDR
100	45.95	27.77	27.71	26.98	35.69	1.13	1.05	0.01
125	47.02	29.44	28.96	28.71	34.90	1.04	0.35	0.01
150	36.32	30.02	23.62	22.50	21.75	10.79	1.48	0.01
175	34.05	28.20	21.47	21.41	19.25	9.49	0.10	0.02
200	36.70	31.09	23.94	23.75	20.53	10.68	0.26	0.02
225	35.59	29.59	22.71	22.17	20.89	10.59	0.72	0.01
250	29.20	24.97	18.61	18.26	15.52	8.98	0.43	0.00
275	28.49	24.76	18.06	17.73	15.10	9.37	0.41	0.01
300	28.81	25.03	17.65	17.26	16.27	10.39	0.49	0.01

Table 8: Experiment 4, average gaps and comparison to best for adjustable objective values in percent.

Note that the lower bound used for Table 8 is simple, and actual gaps can be assumed to be considerably smaller. To give a more detailed view on algorithm performance, we also present the average difference in percent to the best solution found on every instance in the right columns of Table 8.

6. Conclusions

We considered stacking problems with two kinds of constraints: The first is given by a general stacking matrix encoding which items can be stacked onto each other. This can be used to model practical requirements such as item departure times, or incompatible item dimensions. The second are payload constraints, which ensure that not more weight is stacked on top of an item than the stability of this item allows. However, as item weights are uncertain, we introduced robust stacking problems.

Two robust models were tackled: One where the complete item stacking needs to be fixed in advance before item weights are known; and one where adjustments in the item order can be made afterwards. Finite and interval-based uncertainty sets were considered and different solution approaches presented. In an extensive computational study on randomly generated instances, the impact of the number of items, the payload violation parameter, and the stacking matrix were analyzed.

We briefly review possible problem extensions in the following. Further uncertainty sets, such as interval-based uncertainty sets with additional restrictions may be considered. An example for such sets include the model of of Bertsimas and Sim (see [13]), where the total relative deviation of item weights from their nominal values is bounded by some parameter Γ. Using such an uncertainty set, Algorithms 1 and 2 are still applicable, with only slight differences in the computation of worst-case scenarios.

Finally, the adjustable approach presented in this paper can be extended by considering restrictions on the rearrangements that are allowed once the scenario becomes known. This is similar to the idea of recoverable robustness (see, e.g., [35]). We count the number of operations which are necessary to rearrange a stack. As an example, for a single stack, possible recovery cost measurements between two solutions x, x' include: The Hamming distance (i.e., the number of differently positioned items, given by $\sum_{i,l} |x_{iql} - x'_{iql}|$); or the number of items from top which have to be removed to transform x to x' or vice versa. In both cases, the recoverable robust counterpart for a finite number of scenarios can be modeled as a mixed-integer linear program similar to (ARF).

References

[1] D. Ambrosino, M. Paolucci, and A. Sciomachen. Computational evaluation of a MIP model for multi-port stowage planning problems. *Soft Computing*, pages 1–11, 2015. DOI 10.1007/s00500-015-1879-y.

[2] D. Ambrosino, M. Paolucci, and A. Sciomachen. Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. *Flexible Services and Manufacturing Journal*, 27(2):263–284, 2015.

[3] D. Ambrosino, M. Paolucci, and A. Sciomachen. A MIP heuristic for multi port stowage planning. *Transportation Research Procedia*, 10:725–734, 2015.

[4] D. Ambrosino, A. Sciomachen, and E. Tanfani. Stowing a containership: the master bay plan problem. *Transportation Research Part A*, 38(2):81–99, 2004.

[5] D. Ambrosino, A. Sciomachen, and E. Tanfani. A decomposition heuristics for the container ship stowage problem. *Journal of Heuristics*, 12(3):211–233, 2006.

[6] S. V. Amiouny, J. J. Bartholdi, J. H. Vande Vate, and J. Zhang. Balance loading. *Operations Research*, 40(2):238–246, 1992.

[7] A. Ben-Tal, B. D. Chung, S. R. Mandala, and T. Yao. Robust optimization for emergency logistics planning: Risk mitigation in humanitarian relief supply chains. *Transportation Research Part B: Methodological*, 45(8):1177–1189, 2011.

[8] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization.* Princeton University Press, Princeton and Oxford, 2009.

[9] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Math. Programming A*, 99(2):351–376, 2003.

[10] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

[11] H. P. Benson. Concave minimization: Theory, applications and algorithms. In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, volume 2 of *Nonconvex Optimization and Its Applications*, pages 43–148. Springer US, 1995.

[12] D. Bertsimas, D. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[13] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[14] A. Bortfeldt and G. Wäscher. Constraints in container loading–a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013.

[15] P.C. Bouman, J.M. Akker, and J.A. van den Hoogeveen. Recoverable robustness by column generation. In *European Symposium on Algorithms*, volume 6942 of *Lecture Notes in Computer Science*, pages 215–226. Springer, 2011.

[16] F. Bruns, M. Goerigk, S. Knust, and A. Schöbel. Robust load planning of trains in intermodal transportation. *OR Spectrum*, 36(3):631–668, 2014.

[17] F. Bruns and S. Knust. Optimized load planning of trains in intermodal transportation. *OR Spectrum*, 34(3):511–533, 2012.

[18] F. Bruns, S. Knust, and N. Shakhlevich. Complexity results for storage loading problems with stacking constraints. *European Journal of Operational Research*, 249(3):1074–1081, 2016.

[19] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Robust algorithms and price of robustness in shunting problems. In *Proceeding of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS07)*, 2007.

[20] A. Delgado, R. M. Jensen, K. Janstrup, T. H. Rose, and K. H. Andersen. A constraint programming model for fast optimal stowage of container vessel bays. *European Journal of Operational Research*, 220(1):251–261, 2012.

[21] A. Delgado, R. M. Jensen, and C. Schulte. Generating optimal stowage plans for container vessel bays. In *Principles and Practice of Constraint Programming - CP09*, volume 5732 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2009.

[22] A. L. Erera, J. C. Morales, and M. Savelsbergh. Robust optimization for empty repositioning problems. *Operations Research*, 57(2):468–483, 2009.

[23] M. Goerigk and B. Grün. A robust bus evacuation model with delayed scenario information. *OR Spectrum*, 36(4):923–948, 2014.

[24] M. Goerigk and A. Schöbel. Algorithm engineering in robust optimization. *LNCS State-of-the-Art Surveys Springer*, 2015. To appear.

[25] D. Halliday, R. Resnick, and J. Walker. *Principles of Physics, 10th Edition International Student Version.* John Wiley & Sons, 2014.

[26] IBM. *IBM ILOG CPLEX 12.6 User's Manual*, 2013.

[27] J. Kang, K. R. Ryu, and K. H. Kim. Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent Manufacturing*, 17(4):399–410, 2006.

[28] J. G. Kang and Y. D. Kim. Stowage planning in maritime container transportation. *Journal of the Operational Research Society*, 53(4):415–426, 2002.

[29] K. H. Kim, Y. M. Park, and K. R. Ryu. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124(1):89–101, 2000.

[30] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications.* Kluwer Academic Publishers, 1997.

[31] X.T. Le and S. Knust. MIP-based approaches for robust storage loading problems with stacking constraints. Technical report, University of Osnabrück, 2015. Submitted.

[32] C.-Y. Lee and Q. Meng. *Handbook of ocean container transportation logistic: Making global supply chains effective.* Springer, 2015.

[33] J. Lehnfeld and S. Knust. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, 239(2):297–312, 2014.

[34] W. Li, Y. Wu, and M. Goh. *Planning and scheduling for maritime container yards: supporting and facilitating the global supply network.* Springer, 2015.

[35] C. Liebchen, M. Lübbecke, R. H. Möhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In R. K. Ahuja, R.H. Möhring, and C.D. Zaroliagis, editors, *Robust and online large-scale optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2009.

[36] K. Mathur. An integer-programming-based heuristic for the balanced loading problem. *Operations Research Letters*, 22(1):19–25, 1998.

[37] R. Montemanni. A Benders decomposition approach for the robust spanning tree problem with interval data. *European Journal of Operational Research*, 174(3):1479–1490, 2006.

[38] D. Pacino, A. Delgado, R. M. Jensen, and T. Bebbington. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. In *Computational logistics*, volume 6971 of *Lecture Notes in Computer Science*, pages 286–301. Springer, 2011.

[39] D. Pacino, A. Delgado, R. M. Jensen, and T. Bebbington. An accurate model for seaworthy container vessel stowage planning with ballast tanks. In *Computational logistics*, volume 7555 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2012.

[40] A. Sciomachen and E. Tanfani. The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem. *IMA Journal of Management Mathematics*, 14(3):251–269, 2003.

[41] K. Tierney, D. Pacino, and R. M. Jensen. On the complexity of container stowage planning problems. *Discrete Applied Mathematics*, 169:225–230, 2014.

[42] UNCTAD/RMT/2014. Review of maritime transport 2014. United Nations conference on Trade and Development, eISBN 978-92-1-056861-6.

[43] W. Vancroonenburg, J. Verstichel, K. Tavernier, and G. V. Berghe. Automatic air cargo selection and weight balancing: A mixed integer programming approach. *Transportation Research Part E*, 65:70–83, 2014.

[44] B. Zeng and L. Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

[45] M. Zeng, M. Y. H. Low, W. J. Hsu, S. Y. Huang, F. Liu, and C. A. Win. Automated stowage planning for large containerships with improved safety and stability. In *Proceedings of the 2010 Winter Simulation Conference*, B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds., pages 1976–1989. Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ, 2010.

Appendix A. Gravity center of a stack and the payload factor

In the following we show in more detail how the stacking configuration of items in a stack should be to attain the lowest gravity center of the stack, and how the stack's gravity center is influenced by the payload parameter a.

Consider a single stack consisting of n items of positive weights stored from level 1 to level n. Assume that the items have a common height $h > 0$ and the weight of each item is uniformly distributed over the item's volume. By this assumption, the gravity center of each item is exactly in the middle of the item. More precisely, if we start measuring the height from the ground (height 0) where all items are put on, then the height of the gravity center of the item stored at level i is $h_i := \left(i - \frac{1}{2}\right) h$ (see Figure A.2). Let G be the gravity center of the whole stack.

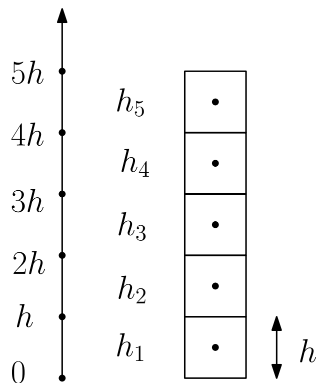

Figure A.2: Stack with 5 items.

By "hard stacking constraints on weights" we mean that heavier items must be put below lighter ones. The following lemma shows how we should store the items with given weights in order to have the lowest position for G.

Lemma 4. *Among all configurations of stacking items with given weights, a configuration satisfying the hard stacking constraints on weights has the lowest position of the gravity center G.*

Proof: Let w_i be the weight of the item stored at level i. According to [25], Chapter 12, the height of G is computed by

$$h_G = \frac{\sum_{i=1}^{n} w_i h_i}{\sum_{i=1}^{n} w_i}. \tag{A.1}$$

A global minimizer of the numerator is then to sort items non-increasingly according to weight. $\square$

We now assume that the weights of the items can vary, but always satisfy the payload constraints with a given payload parameter $a > 0$. This means that we always have

$$\sum_{i=0}^{k-1} w_{n-i} \leq a w_{n-k} \qquad (k = 1, \ldots, n-1), \tag{A.2}$$

where again the weight of the item stored at level i is denoted by w_i. Denote by $\bar{h}_G$ the height of the highest possible position of G. The behavior of $\bar{h}_G$ when the value of payload parameter a varies is shown in the following lemma.

Lemma 5. $\bar{h}_G(a)$ *is a monotonically increasing function.*

Proof: Regarding (A.2), it follows from (A.1) that the highest possible gravity center of the set consisting of the two topmost items is attained when $w_n = aw_{n-1}$. Similarly, the highest possible gravity center of the set consisting of the three topmost items is attained when

$$\begin{aligned} w_n &= aw_{n-1}, \\ w_n + w_{n-1} &= aw_{n-2}. \end{aligned}$$

By induction on n we can deduce that the highest possible position of G is attained when

$$\begin{aligned} w_n &= aw_{n-1}, \\ w_n + w_{n-1} + \ldots + w_{n-k+1} &= aw_{n-k} \qquad \forall\ k = 2, \ldots, n-1, \end{aligned}$$

or equivalently,

$$\begin{aligned} w_{n-1} &= \frac{1}{a} w_n, \\ w_{n-k} &= \frac{(a+1)^{k-1}}{a^k} w_n \qquad (k = 2, \ldots, n-1). \end{aligned}$$

Therefore, we get

$$\begin{aligned} \sum_{i=1}^{n} w_i &= \left(\frac{(a+1)^{n-2}}{a^{n-1}} + \ldots + \frac{a+1}{a^2} + \frac{1}{a} + 1 \right) w_n \\ &= A w_n, \end{aligned}$$

where

$$A = \frac{(a+1)^{n-2}}{a^{n-1}} + \ldots + \frac{a+1}{a^2} + \frac{1}{a} + 1.$$

As we have

$$\frac{a+1}{a} A - A = \frac{(a+1)^{n-1}}{a^n},$$

it holds that

$$A = \left(\frac{a+1}{a} \right)^{n-1}.$$

By applying the formula (A.1) we have

$$\begin{aligned} \bar{h}_G = \frac{\displaystyle\sum_{i=1}^{n} w_i h_i}{\displaystyle\sum_{i=1}^{n} w_i} &= \frac{\displaystyle\sum_{i=1}^{n} (i - \frac{1}{2}) h w_i}{\displaystyle\sum_{i=1}^{n} w_i} = \frac{\displaystyle\sum_{i=1}^{n} i h w_i}{\displaystyle\sum_{i=1}^{n} w_i} - \frac{1}{2} h \\ &= \frac{\displaystyle\sum_{i=1}^{n-1} ih \frac{(a+1)^{n-i-1}}{a^{n-i}} w_n + nhw_n}{\left(\frac{a+1}{a}\right)^{n-1} w_n} - \frac{1}{2} h \\ &= \sum_{i=1}^{n-1} i \frac{1}{a} \left(\frac{a}{a+1} \right)^{i} h + n \left(\frac{a}{a+1} \right)^{n-1} h - \frac{1}{2} h. \end{aligned} \tag{A.3}$$

If we set $u := \frac{a}{a+1}$, then $0 < u < 1$ and $a = \frac{u}{1-u}$. Moreover, we can rewrite (A.3) as follows:

$$\begin{aligned}
\bar{h}_G &= \sum_{i=1}^{n-1} i \frac{1-u}{u} u^i h + n u^{n-1} h - \frac{1}{2} h \\
&= \sum_{i=1}^{n-1} i(1-u) u^{i-1} h + n u^{n-1} h - \frac{1}{2} h \\
&= \sum_{i=1}^{n-1} i(u^{i-1} - u^i) h + n u^{n-1} h - \frac{1}{2} h \\
&= \left(\frac{1}{2} + \sum_{i=1}^{n-1} u^i \right) h. \qquad \text{(A.4)}
\end{aligned}$$

It is an immediate consequence of (A.4) that $\bar{h}_G$ is monotonically increasing with respect to u. Obviously, $u = \frac{a}{a+1} = 1 - \frac{1}{a+1}$ is monotonically increasing with respect to a. It follows that $\bar{h}_G(a)$ is a monotonically increasing function. $\square$

As a consequence of Lemma 5, the smaller the value of a is, the lower $\bar{h}_G$ is, i.e., the more stable the stack is. Moreover, given a desired position for G, we can compute the payload parameter a corresponding to that position, and then use that value of a for controlling the stability of the stack during the loading process.

Appendix B. SRI with $1 < a \leq 2$

In the following we present a compact formulation for (SRI) when $1 < a \leq 2$. This contains formulation (27)-(38) for $a < 1$, which is extended by new variables $w_{ql}^{k_1,k_2,k_3}$ representing the weight of the item in stack q at level l when there is a break at positions k_1 and k_3, as well as a light item atop a heavy item in level k_2. Additionally, variables $v_{ql}^{k_1,k_2,k_3}$ are used to measure the payload violation in this scenario. We denote $L^* := \{(k_1,k_2,k_3) \in (L \setminus \{b\}) \times (L \setminus \{b\}) \times (L \setminus \{b\}) : k_1 < k_2 < k_3\}$.

$$\min \sum_{q \in Q} v_q \tag{B.1}$$

$$\text{s.t.} \quad \sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 \qquad \forall i \in I \tag{B.2}$$

$$\sum_{i \in I} x_{iql} \leq 1 \qquad \forall q \in Q, l \in L \tag{B.3}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x_{jq,l-1} \geq x_{iql} \qquad \forall i \in I, q \in Q, l \in L \setminus \{1\} \tag{B.4}$$

$$w_{ql}^k = \begin{cases} \sum_{i \in I} \underline{w}_i x_{iql} & \text{if } l \leq k \\ \sum_{i \in I} \overline{w}_i x_{iql} & \text{if } k < l \end{cases} \qquad \forall q \in Q, l \in L, k \in L \setminus \{b\} \tag{B.5}$$

$$w_{ql}^{k_1,k_2,k_3} = \begin{cases} \sum_{i \in I} \underline{w}_i x_{iql} & \text{if } l \leq k_1 \\ \sum_{i \in I} \overline{w}_i x_{iql} & \text{if } k_1 < l \leq k_2 \\ \sum_{i \in I} \underline{w}_i x_{iql} & \text{if } k_2 < l \leq k_3 \\ \sum_{i \in I} \overline{w}_i x_{iql} & \text{if } k_3 < l \end{cases} \qquad \forall q \in Q, l \in L, (k_1,k_2,k_3) \in L^* \tag{B.6}$$

$$\sum_{h > l} w_{qh}^k - a w_{ql}^k \leq v_{ql}^k \qquad \forall q \in Q, k, l \in L \setminus \{b\} \tag{B.7}$$

$$\sum_{h > l} w_{qh}^{k_1,k_2,k_3} - a w_{ql}^{k_1,k_2,k_3} \leq v_{ql}^{k_1,k_2,k_3} \qquad \forall q \in Q, l \in L, (k_1,k_2,k_3) \in L^* \tag{B.8}$$

$$\sum_{l \in L \setminus \{b\}} v_{ql}^k \leq v_q \qquad \forall q \in Q, k \in L \setminus \{b\} \tag{B.9}$$

$$\sum_{l \in L \setminus \{b\}} v_{ql}^{k_1,k_2,k_3} \leq v_q \qquad \forall q \in Q, (k_1,k_2,k_3) \in L^* \tag{B.10}$$

$$x_{iql} \in \{0,1\} \qquad \forall i \in I, q \in Q, l \in L \tag{B.11}$$

$$v_{ql}^k \geq 0 \qquad \forall q \in Q, l, k \in L \setminus \{b\} \tag{B.12}$$

$$v_{ql}^{k_1,k_2,k_3} \geq 0 \qquad \forall q \in Q, l, (k_1,k_2,k_3) \in L^* \tag{B.13}$$

$$w_{ql}^k \geq 0 \qquad \forall q \in Q, l \in L, k \in L \setminus \{b\} \tag{B.14}$$

$$w_{ql}^{k_1,k_2,k_3} \geq 0 \qquad \forall q \in Q, l \in L, (k_1,k_2,k_3) \in L^* \tag{B.15}$$

$$v \geq 0 \tag{B.16}$$

Appendix C. ARF with subsets of stacks

The adjustable models in this paper have been concerned with the setting that every item is assigned to a specific stack in the first stage. Here we show that this can be extended to assigning items to subsets of stacks in the first stage instead.

Let the set of stacks $Q = \{1, \dots, m\}$ be partitioned into subsets $Q_1, \dots, Q_R$ such that an item assigned to the subset Q_r may be assigned to any position in any stack $q \in Q_r$ in the second stage, but not to any stack outside of Q_r. We write $\mathcal{R} = \{1, \dots, R\}$.

$$\text{(ARF-Sub)} \quad \min\ v \tag{C.1}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} z_{ir} = 1 \qquad \forall\, i \in I \tag{C.2}$$

$$\sum_{i \in I} z_{ir} \leq |Q_r| b \qquad \forall\, r \in \mathcal{R} \tag{C.3}$$

$$\sum_{q \in Q_r} \sum_{l \in L} x^k_{iql} = z_{ir} \qquad \forall\, i \in I, k \in \mathcal{N}, r \in \mathcal{R} \tag{C.4}$$

$$\sum_{i \in I} x^k_{iql} \leq 1 \qquad \forall\, q \in Q, l \in L, k \in \mathcal{N} \tag{C.5}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x^k_{jq,l-1} \geq x^k_{iql} \qquad \forall\, i \in I, q \in Q, l \in L \setminus \{1\} \tag{C.6}$$

$$\sum_{j \in I} \sum_{h=l+1}^{b} w^k_j x^k_{jqh} - a \sum_{i \in I} w^k_i x^k_{iql} \leq v^k_{ql} \qquad \forall\, q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} \tag{C.7}$$

$$\sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v^k_{ql} \leq v \qquad \forall\, k \in \mathcal{N} \tag{C.8}$$

$$z_{ir} \in \{0, 1\} \qquad \forall\, i \in I, r \in \mathcal{R} \tag{C.9}$$

$$x^k_{iql} \in \{0, 1\} \qquad \forall\, i \in I, q \in Q, l \in L, k \in \mathcal{N} \tag{C.10}$$

$$v^k_{ql} \geq 0 \qquad \forall\, q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} \tag{C.11}$$

$$v \geq 0 \tag{C.12}$$

In constraints (C.2) we enforce that every item needs to be assigned to exactly one subset of stacks, while constraints (C.3) restrict the number of items assigned to every subset according to its capacity. This assignment is connected to the second-stage variables x^k via constraints (C.4), which ensure that one level in one of the appropriate stacks needs to be used for every item. The remaining constraints are the same as in (ARF): A level can only be used once, stacking matrix constraints need to be respected, and the worst-case payload violation is modeled.

Appendix D. Results of experiment 3

We now consider the impact of different values for the payload violation parameter a. We choose $a = 0.5$ to $a = 1.5$ with a stepsize of 0.1. Other parameters are fixed to $n = 24$, $m = 4$, $b = 6$, $d = 0.5$. Results are presented in Tables D.9 – D.13.

Note that computation times, presented in Table D.9, are not monotone in a; instead, problems with a being in the vicinity of 1 tend to take longer to solve for algorithms using a fixed number of sampled scenarios. Hence, it may be helpful to slightly change the value of parameter a if computation times are too high for a practical instance.

a	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI
0.5	0.2	9.3	20.1	900.0	0.4	80.4	412.2	900.0
0.6	0.2	9.8	27.4	900.0	0.4	62.9	630.1	900.0
0.7	0.2	15.8	27.0	900.0	0.5	109.4	897.1	900.0
0.8	1.1	32.8	62.9	900.0	1.7	519.4	897.8	900.0
0.9	2.0	44.5	168.9	900.0	2.4	898.0	899.3	900.0
1.0	0.2	80.0	400.1	900.0	0.6	128.9	897.6	900.0
1.1	0.2	29.7	112.7	900.0	0.7	58.4	655.0	900.0
1.2	0.2	12.5	24.7	900.0	0.8	95.6	604.6	900.0
1.3	0.2	15.9	31.3	900.0	0.8	62.2	548.7	900.0
1.4	0.2	12.5	38.7	900.0	1.0	57.3	643.3	900.0
1.5	0.2	20.1	42.9	900.0	0.9	79.0	703.3	900.0

Table D.9: Experiment 3, median computation times in seconds.

Considering the strict objective values (Table D.10), we note the gap to be roughly monotonically increasing with a for Nom, S-10 and S-20; for SI, there is a disproportional increase in gap for $a \geq 1$. Absolute objective values are smaller for increasing a, which may also add to an increased algorithm gap.

a	Nom	S-10	S-20	SI	SIC
0.5	8.77	7.13	4.89	0.37	0.00
0.6	9.30	7.60	5.26	0.43	0.00
0.7	10.00	8.28	5.99	0.34	0.00
0.8	11.01	8.54	6.43	1.26	0.00
0.9	12.11	9.66	6.86	1.00	0.00
1.0	12.73	10.05	7.15	7.57	0.00
1.1	12.57	10.20	6.74	6.16	0.00
1.2	12.69	10.62	7.35	7.36	0.00
1.3	12.73	10.57	7.58	6.53	0.00
1.4	13.25	11.64	8.44	6.43	0.00
1.5	14.07	11.61	7.81	6.98	0.00

Table D.10: Experiment 3, average gaps for strict objective values in percent.

We show the adjustable objective value gaps in Table D.11. Note that overall gaps tend to be smaller than for strict robustness, which is in agreement with the the observations made in Section 5.1.4. The heuristic algorithms perform best, in particular AIDR is slightly outperforming AIPG and AILS. As with strict objective values, gaps tend to increase with a.

In Tables D.12 and D.13, we investigate the sensitivity of solutions regarding the parameter a. In Table D.12, we evaluate solutions calculated for varying values of a as if they were solutions

a	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI	AIPG	AILS	AIDR
0.5	3.96	3.51	3.43	1.67	1.50	3.59	3.31	1.40	1.17	1.17	1.11
0.6	4.15	3.94	3.74	2.55	1.97	4.04	3.50	1.85	1.66	1.70	1.63
0.7	4.58	4.73	3.58	2.19	1.99	3.82	3.29	2.06	1.64	1.67	1.59
0.8	4.30	4.12	3.78	2.26	2.28	3.63	3.21	1.98	1.65	1.64	1.61
0.9	4.49	4.04	4.07	2.46	2.31	4.11	3.62	2.13	1.86	1.86	1.85
1.0	5.12	4.71	4.43	4.94	3.14	4.72	4.45	3.23	2.59	2.55	2.48
1.1	4.16	3.47	4.11	3.64	2.58	3.63	3.48	2.47	1.93	1.89	1.87
1.2	4.45	4.79	3.65	4.34	2.38	3.92	3.23	2.39	1.88	1.86	1.86
1.3	4.06	3.99	3.53	3.70	2.22	2.95	2.93	1.94	1.65	1.67	1.64
1.4	3.75	4.57	3.91	3.44	2.51	4.00	3.48	2.52	2.05	2.03	2.03
1.5	4.39	4.53	4.42	3.71	2.72	3.17	3.75	2.60	2.17	2.17	2.15

Table D.11: Experiment 3, average gaps for adjustable objective values in percent.

to $a = 0.5$, and to $a = 1.5$ in Table D.13. We find that those solutions that have the smallest gap in Table D.11 (i.e., AI, AIPG, AILS and AIDR), tend to be most sensitive to changes in a; however, they still outperform the other algorithms. Overall, gaps remain small, which means that solutions are relatively robust to changes in or wrong estimates of the parameter a. This also aligns well with the suggestion to slightly change parameter a when computation times are too high.

a	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI	AIPG	AILS	AIDR
0.5	3.96	3.51	3.43	1.67	1.50	3.59	3.31	1.40	1.17	1.17	1.11
0.6	3.75	3.52	3.33	2.14	1.47	3.61	3.08	1.41	1.20	1.26	1.16
0.7	4.13	4.28	3.21	1.83	1.66	3.50	2.94	1.72	1.34	1.45	1.26
0.8	3.81	3.69	3.46	1.93	1.86	3.34	3.02	1.80	1.47	1.45	1.42
0.9	3.77	3.42	3.63	2.01	1.80	3.59	3.06	1.80	1.52	1.47	1.49
1.0	3.68	3.39	3.36	3.74	1.89	3.52	3.44	2.40	1.95	2.20	1.88
1.1	3.88	2.97	3.87	3.41	2.06	3.61	2.88	2.64	2.01	2.13	2.23
1.2	4.08	4.29	3.79	3.96	2.11	3.91	3.08	2.59	2.28	2.20	2.43
1.3	3.72	3.81	3.94	3.52	2.32	3.27	3.30	2.69	2.62	2.41	2.52
1.4	3.49	4.19	3.87	3.55	2.22	3.75	3.14	2.95	2.30	2.44	2.45
1.5	3.82	3.99	3.99	3.49	2.77	3.07	3.23	2.93	2.65	2.65	2.74

Table D.12: Experiment 3, average gaps for adjustable objective values in percent, when evaluating solutions to $a = 0.5$.

a	Nom	S-10	S-20	SI	SIC	A-5	A-10	AI	AIPG	AILS	AIDR
0.5	4.28	3.99	3.55	2.76	2.73	4.05	3.73	2.48	2.66	2.51	2.52
0.6	3.94	3.94	3.83	3.30	2.89	4.16	3.41	2.44	2.48	2.44	2.42
0.7	4.68	5.02	3.67	2.83	2.57	3.73	3.19	2.44	2.42	2.47	2.40
0.8	4.71	4.42	3.99	2.93	3.14	3.65	3.01	2.43	2.39	2.42	2.32
0.9	4.53	4.26	3.78	2.68	2.75	3.92	3.48	2.47	2.35	2.40	2.34
1.0	4.44	3.90	3.57	4.39	2.58	3.70	3.60	2.59	2.31	2.27	2.31
1.1	4.02	3.63	4.10	3.46	2.74	3.47	3.53	2.48	2.31	2.21	2.23
1.2	4.67	5.14	3.79	4.54	2.65	3.83	3.40	2.65	2.22	2.20	2.20
1.3	4.56	4.60	4.00	4.17	2.74	3.30	3.34	2.46	2.19	2.19	2.16
1.4	3.80	4.70	4.02	3.56	2.65	4.12	3.60	2.62	2.17	2.15	2.15
1.5	4.39	4.53	4.42	3.71	2.72	3.17	3.75	2.60	2.17	2.17	2.15

Table D.13: Experiment 3, average gaps for adjustable objective values in percent, when evaluating solutions to $a = 1.5$.