Robust solutions to storage loading problems under uncertainty

Dissertation
zur Erlangung des Doktorgrades (Dr. rer. nat.)
des Fachbereichs Mathematik und Informatik
der Universität Osnabrück

vorgelegt von

Xuan Thanh Le

October 2016

Acknowledgements

By the occasion of presenting this dissertation, I would like to express my sincere gratitude to everybody who were helping, supporting, and encouraging me during my pleasant time of Ph.D. study.

My utmost gratitude and admiration are to my great doctor-mother: Professor Sigrid Knust. Thanks to her conscientious guidance, my research skills have been growing up. She introduced me to many workshops and short-term schools, as well as connected me with many collaborators who helped me a lot in enlarging my knowledge. She gave me the freedom to pursue my research interests and also many chances for presenting my findings in seminars and conferences, from that my presentation skills have been sharpened. Her clear and careful characteristics have a significant influence on me in conducting research and writing papers for publication. She is not only a master in advising students but also a very warm person in daily life. I truly enjoyed the concerts of the orchestra where she played in, and the joyful hiking trips to the wonderful hills of Osnabrück, from that I discovered my new passions beside mathematics.

I am also deeply grateful to my co-supervisor, Professor Matthias Reitzner, for his guidance and support. He is an expert not only in his profession but also in teaching skills. What I can always learn from his impressive lectures are the art of analyzing a mathematical problem, making it easy to understand, and having clear solution approaches.

I would like to express many thanks to my collaborators: Marc Goerigk and Christina Büsing, for their helpful cooperation, useful advices, and valuable experiences.

I wish to acknowledge the financial support provided by the Graduate School Combinatorial Structures in geometry, which in turn is financed by the German Science Foundation according to Deutsche Forschungsgemeinschaft GRK 1916/1.

I am particularly grateful for the assistance given by Institute of Mathematics (Hanoi, Vietnam). My great thank goes to Professor Ngo Viet Trung for introducing me to the Graduate School and for his continuous encouragement.

I further like to address my thanks to all my colleagues at Department of Mathematics and Informatics, Osnabrück University for creating a friendly and collaborative working environment. I specially thank my kindhearted officemate, Ulrich von der Ohe, for all of his unconditional helps and for sharing lots of joyful moments.

I should not forget to thank all of my friends, those are not only in Osnabrück but also scattered around the world, for joining with me in my off-work time and helping me balance between work and pleasure.

My dissertation would not be possible without the endless love and spiritual support from my parents and my wife. My absence from home during my time of doing Ph.D. abroad means a lot of personal sacrifices of them. I would like to say my heartfelt gratitude to my family for their understanding and patience.

Contents

Chapter 1

Introduction

For the last several decades, the rapid growth of international trade and the ongoing globalization of supply chains have been witnessed via a huge amount of goods stored in and transited through commercial storage areas all over the world. Within the context of container terminals, the annual worldwide container throughput has increased around 6.7 times from 88.150 million TEUs in 1990 to 588.905 million TEUs in 2011 (see [63] Chapter 1), and reached 651.1 million TEUs in 2013 (see [80]). Here the term TEU abbreviates a twenty-feet-equivalent unit, which is used to refer to one container of twenty-feet length. There has been also a steady increase in capacity of storage areas. As reported in [37], the average carrying capacity of container ships grew from 2259 TEUs in January 2004 to 4449 TEUs in January 2014, and the maximum capacity of container ships over the world today is as many as 19224 TEUs. Such numbers strengthen the essentiality of having efficient storage operations in management of commercial storage areas.

As classified in [64], different types of optimization problems arise due to the interim storage operations of items in a storage area. *Storage loading problems* appear when incoming items arrive at a storage area and one has to temporarily assign them to positions in a convenient way for handling afterwards. In contrast, *unloading problems* arise when outgoing items need to be retrieved from the storage area, and the goal is to decide which items will leave the storage in which order and which relocations should be performed. *Premarshalling problems* are refered to the cases when items have to be relocated within the storage area such that later on all items can be retrieved without any further relocation. *Combined loading/unloading problems* occur when incoming items need to be stored while outgoing items need to be retrieved.

This thesis focuses on storage loading problems in which the input data are subjected to uncertainty. In the following section we will present the basic setup of the storage loading problems in the scope of this thesis. We also introduce some related concepts and useful notations to formally describe the problems. Then in Section 1.2 we present the principles of some robustness concepts that can be applied to solve the storage loading problems in a robust approach. We give a short review on literature of storage loading problems in Section 1.3 before sketching the outline of this thesis in Section 1.4.

1.1 Storage loading problems

The setup of storage loading problems concerns several aspects including configuration of storage areas, how the incoming items are specified, constraints on stacking the items, and objectives of the storage loading process.

1.1.1 Configuration of storage areas

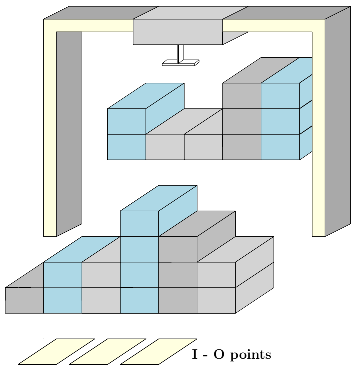
The storage areas in their natural and professional configuration are organized in parallel stacks, where items are stockpiled on top of each other within each stack. Such stack-based storage areas can be found in the following logistics applications.

- In container terminals (see e.g. [33, 82]), items are inbound and outbound containers stored in stacking yards. As illustrated in Figure 1.1a, stacking yards are usually divided into blocks, each block consists of several bays having the same number of stacks aligned side by side. The stacks have the same number of levels (also called tiers), i.e., they can store the same number of containers. Containers are exchanged between transfer vehicles and stacking yards at input/output (I/O) points located either at both ends of each block (in European layout) or along the blocks (in Asian layout). Such stacking yard layouts are designed for the use of yard cranes (also called gantry cranes), that are the most popular container handling equipments in stacking yards. The yard cranes can simultaneously move along the bays and between both ends of each block. In loading processes, incoming containers are picked up at I/O points by the yard cranes and stored at their destinations in the block. Conversely, when a container needs to be retrieved, a single yard crane picks the container up from its location and puts it at an I/O point of the corresponding block. Due to the technical design of yard cranes, within each stack the containers are stored in last-in-first-out order, i.e., only the topmost container of any stack can be directly accessed by the yard cranes.

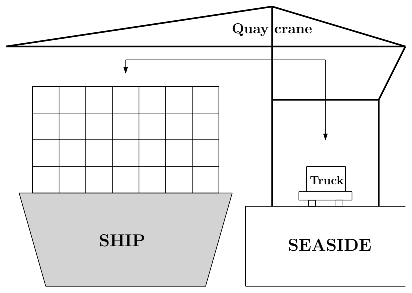
- In container ships (see e.g. [70, 71]), items are seaworthy containers stored in stacking yards of means of marine transportation (such as ships, barges, or vessels) to transport between seaports. The stacking yards are also organized in blocks of parallel stacks. Typically, for the container ships of type Lift-on Lift-off (cf. [5]), containers are loaded onto or unloaded from the top of stacks by using quay cranes located at the seasides of the ports (see Figure 1.1b). Furthermore, for safety reasons while traveling, each container ship is associated with a safe field of vision from its navigating, manoeuvring, and monitoring work stations to the sea surface. The technical restrictions from the quay cranes and the ship's field of vision impose a limit on the stacks' height, i.e., each stack can store up to a limited number of containers.

- In tram or bus depots (see e.g. [22, 28, 83]), one often encounters a yard consisting of dead-end tracks, where the trams or buses are parked in the last-in-first-out manner for purposes of maintaining or waiting for their next scheduled trips (see Figure 1.1c). Such depots can be viewed as stack-based storage areas if we think of the tracks as stacks, and the trams or buses to be parked play the role of items to be stored. The number of such vehicles that can be parked at the same time on each track is limited by the track's capacity.

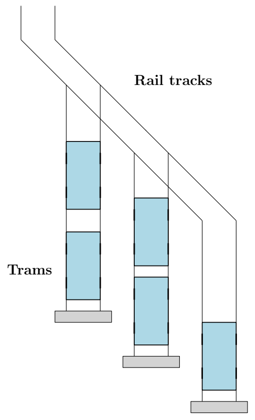
- In warehouses (see e.g. [9, 69, 72]), items are companies' products stored before being picked according to customer orders. The products are often packaged and moved in unit loads of the same size, such as carton boxes, plastic boxes, or crates. The most simple storage method in warehouses is block stacking, in which the unit loads are stored on top of each other in stacks. The stacks are supported from bottom by pallets, that are put either on the floor (for large products) or on the selves of racks (for small products). Due to the cover boxes' strength and the racks' structure, each stack can store a limited number of the unit loads.



(a) Container terminal.



(b) Container ship.



(c) Tram depot.

Figure 1.1: Stack-based storage areas in practice.

Although the storage loading problems motivated from the above practical applications are described in different terms, they share several common features, in which the following ones characterize the storage loading problems in the focus of this thesis.

- The storage area is arranged in stacks and each stack has its own fixed position in a two-dimensional area. This means that deciding where to position a stack in the area is not a matter, the importance is to decide which stack to use for putting each item.

- The number of levels in each stack is limited, and each level in each stack can be occupied by at most one item. It is a natural assumption that each item is either located on the bottom level of some stack (i.e. the item is the first one stored in its stack) or stacked directly on top of another item. Direct access is only allowed to the topmost item of each stack, i.e., the items in each stack are arranged in last-in-first-out order.

We restrict our consideration to the case that the stacks have the same number of levels. We furthermore focus on pure loading problems, in which we deal with the storage of incoming items and assume that no outgoing item is retrieved during the loading process. For modeling the configuration of storage areas, if not stated differently, throughout this thesis we will use the following notations. The set of stacks in a storage area is denoted by $Q := \{1, \dots, m\}$, where m is the number of stacks in the storage area. The set of all items is denoted by $I := \{1, \dots, n\}$, where n is the number of items concerned to the loading process. Each stack contains b levels (i.e., at most b items can be stored in each stack), and the set of levels in one stack is denoted by $L := \{1, \dots, b\}$.

1.1.2 Storage precedence of incoming items

An important feature of storage loading problems is storage precedence of incoming items, indicating in which order the items are stored. In practice this is typically done in one of the following ways (cf. [64]).

- *Store a set of incoming items:* all items of a set have to be loaded into the storage area, no matter what the sequence of storage operations is. An example is the case that all containers on an incoming train have to be stored into a stacking yard in a container terminal.

- *Store a sequence of incoming sets of items:* several sets of items have to be stored into the storage area in the set-by-set manner, i.e. all items belonging to the same set have to be stored before any item of another set can be stored (the sequence of storage operations for each set does not matter). For instance, several trains arrive consecutively at a container terminal and have to be unloaded in the order of their arrival.

- *Store a sequence of incoming items:* several items have to be stored into the storage area in the item-by-item manner, i.e. one item after another. For example, this happens when an inbound stream of trams has to be parked in rail tracks of a depot.

- *Store a set of sequences of incoming items:* several sets of items, where each set has an associated sequence of storage operations, have to be stored into the storage area so that the sequences themselves have to be regarded but can be mixed in delivery. This applies for instance in tram depots where some inbound streams of trams arrive at the same time and have to be parked in rail tracks of the depots. There, the parking precedence of trams of different streams is not important, but the sequence of parking trams in each stream has to be respected.

If not indicated differently, throughout this thesis we will use the following notations (introduced in [64]) to shortly specify the storage precedence of incoming items.

- $\mathcal{I}^{in}$: store a set of incoming items.

- $(\mathcal{I}^{in})_K$: store a sequence of K incoming sets of items.

- π^{in}: store a sequence of incoming items.

- $\{\pi^{in}\}_K$: store a set of K sequences of incoming items.

Note that $\mathcal{I}^{in}$ is a special case of $(\mathcal{I}^{in})_K$ with $K = 1$. Similarly, π^{in} is the particular case of $\{\pi^{in}\}_K$ when $K = 1$. Obviously, π^{in} is a special case of $(\mathcal{I}^{in})_K$ where each set contains exactly one item.

Especially, in some case studies where the storage precedence is of type $(\mathcal{I}^{in})_K$, if the numbering of the sets of items is given in accordance with their precedence, say $I^1, \ldots, I^K$, then we also denote the sequence by $I^1 \to \ldots \to I^K$. Apart from the incoming sets of items, there may be a set I^{fix} of items that are already stored in the storage area. In that case, we do not include I^{fix} in the notation $(\mathcal{I}^{in})_K$, but we denote the sequence by $I^{fix} \to I^1 \to \ldots \to I^K$ to emphasize the appearance of I^{fix}.

1.1.3 Stacking constraints

Normally some items have to be stacked on others. Not every item may be stacked on top of each other (for instance, the stacking process must always regard the given storage precedence of incoming items). Therefore, how the items can be put on top of each other is also an important feature of storage loading problems. Depending on the context of the loading process as well as the future plans for the items, different stacking constraints might have to be respected.

It is common that the stacking constraints are described by a binary relation on some associated parameters of items. More precisely, the following stacking constraints are most popular in practice.

- *Stacking constraints on weight in the most stable sense:* items of lighter weight may only be stacked on top of heavier items, i.e., item i may only be stacked on top of item j if $w_i \leq w_j$, where w_i (resp., w_j) is the weight of item i (resp., j). For convenience, this type of stacking constraints is denoted by $s_{ij}(w, \searrow)$. We prove in Lemma 5.1 a physical fact that, among possible configurations of storing a set of items into a stack, the one generated by applying $s_{ij}(w, \searrow)$ has the lowest gravity center of the whole stack, and consequently it is the most stable configuration. Therefore $s_{ij}(w, \searrow)$ is often applied to storage loading problems in which stability of the stacking solution is an important issue, especially in the context of container ships.

- *Stacking constraints on weight in the reverse-stable sense:* in constrast to $s_{ij}(w, \searrow)$, this type of stacking constraints, denoted $s_{ij}(w, \nearrow)$, requires that items of heavier weight may only be stacked on top of lighter items. This is often applied in the context of storing containers in an intermodal terminal for loading onto a vessel later on. In the terminal, the heavier containers should be stored in higher levels of the stacks, since they have to be retrieved earlier to put in the lower levels of the vessel (so the loaded vessel will have the best stability according to the principle of $s_{ij}(w, \searrow)$ mentioned above).

- *Stacking constraints on length* (denoted $s_{ij}(\ell)$): shorter items may only be stacked on top of longer items, i.e., item i may only be stacked on top of item j if $\ell_i \leq \ell_j$, where ℓ_i (resp., ℓ_j) is the length of item i (resp., j). For example, this is the case of stacking containers in container terminals or container ships (see e.g. [34]), where containers are standardized in length (20-foot, 40-foot, and 45-foot) and longer containers are not allowed to stacked directly on top of shorter ones.

- *Stacking constraints on departure time:* items of higher priority to retrieve may only be stacked on top of the ones with lower priority. This means that item i may only be stacked on top of item j if $d_i \leq d_j$, where d_i (resp., d_j) is the departure time of item i (resp., j). We shall denote this type of stacking constraints by $s_{ij}(d)$.

Normally, in practice the loading process of items has to satisfy a combination of several types of stacking constraints, i.e., these types of stacking constraints have to be respected simultaneously. For example, in container terminals, containers are often stored in such a way that satisfies a combination of stacking constraints on length and on departure time.

Stacking constraints on n items can also be described explicitly by a stacking matrix $S = (s_{ij}) \in \{0,1\}^{n \times n}$, in which $s_{ij} = 1$ if and only if item i can be stacked directly on top of item j. For convenience we always set $s_{ii} = 0$ for every item i (i.e., an item cannot be stacked on itself). We can also represent the stacking constraints by a directed graph with n vertices and arcs $i \to j$ if $s_{ij} = 1$. If the stacking constraints are given in form of a stacking matrix (s_{ij}) without any specified remarks, then we simply denote them by s_{ij}.

The strictness of satisfying stacking constraints during the loading phase is also an important aspect. More precisely, stacking constraints may be hard or soft: hard stacking constraints must be regarded during the loading process, while soft stacking constraints can be violated (but the violation should be avoided as much as possible). Violating soft stacking constraints leads to a number of *unordered stackings*, so-called *misordered stackings* or *mis-overlays* (cf. [31, 64]). In some references (see e.g. [27, 79]) unordered stackings are also called *blockages* or *overstowages.* Formally, given s_{ij} as soft stacking constraints and two items u, v such that $s_{uv} = 0$, putting item u on top of item v would make an unordered stacking with respect to s_{ij}. For example, it is often the case in practice that stacking constraints on departure time are considered to be soft. In that context, an item is called a *blocking item* if it has later departure time than some items (so-called *blocked items*) stacked below it. Due to the last-in-first-out order of accessing items in each stack, in order to retrieve a blocked item we need to move all blocking items above it to other stacks until the blocked item is the topmost one in its stack. Each movement of an item from its stack to another is called a *reshuffle* or *relocation.*

Different types of stacking constraints may have mutual properties. A common point of the four types of stacking constraints mentioned above ($s_{ij}(w, \searrow)$, $s_{ij}(w, \nearrow)$, $s_{ij}(\ell)$, $s_{ij}(d)$) is that they all define a total order on the set of all items. This means these stacking constraints are *transitive* (i.e., if item i is stackable on top of item j and item j is stackable on top of item k, then item i is also stackable on top of item k) and *total* (in the sense that for any pair of items i, j we can stack i on top of j or stack j on top of i). Therefore, for *total order stacking constraints* we refer to the ones that are both transitive and total, as this term is already used in [31]. Note that stacking constraints may be non-transitive, for example the ones induced by stacking restrictions on materials of items (cf. also [31]).

Apart from the stacking constraints, in Chapter 5 we consider storage loading problems with *payload constraints* that make special restrictions on stacking items. More precisely, payload constraints require that the total payload put on top of each item should not exceed a proportion of the item's weight. Stating more formally, the total weight that can be put on top of an item i having weight w_i must be limited by aw_i, where a is a given positive parameter so-called *payload parameter.* Payload constraints are motivated from stability issues of container ships. As discussed in Section 5.1, via an appropriate payload parameter, one can adjust the gravity center of the loaded ship to a desired area, and therefore control the stability of the ship.

1.1.4 Data uncertainty

To satisfy the stacking constraints, one might need to know associated data of items. These data therefore must be sent to managers of storage areas before their arrival. However, due to different reasons, in practice the data information sent in advance are not quite accurate, especially the weight of items. The exact data can only be informed when the items actually arrive, or just a short time before the scheduled loading time. Therefore one may need to take into account the uncertainty of items' data when setting up storage loading plans. In this thesis we are interested in the cases in which uncertain data of items are given in the following ways.

- *Interval uncertainty:* Each item $i \in \{1, \ldots, n\}$ may have some associated parameter a_i (e.g., corresponding to the weight or departure time of the item), in which the value of a_i may vary in an interval $[a_i^{min}, a_i^{max}]$ (without assuming any knowledge of a probability distribution). Lower and upper bounds a_i^{min}, a_i^{max} may be derived from empirical observations or expert knowledge. We denote by $\mathcal{S}_\mathcal{I}$ the set of all possible values of vector $a = (a_1, \ldots, a_n)$, i.e. the Cartesian product of the intervals. Each value $a \in \mathcal{S}_\mathcal{I}$ is called a *scenario.*

- *Discrete uncertainty:* Uncertain data of items lead to uncertainty on the values of entries of the stacking matrix. For a set consisting of n items, there are finitely many possibilities of stacking matrices (note that the diagonal entries of the stacking matrices are always assumed to be 0, and the non-diagonal entries are binary, therefore at most 2^{n^2-n} values of stacking matrices are possible). Hence, uncertain data of items can be represented via a discrete (and finite) set $\mathcal{S}_\mathcal{D} := \{(s_{ij}^1), \ldots, (s_{ij}^N)\}$ of possible scenarios of stacking matrices, where N is the number of these scenarios.

- *Finite uncertainty:* Uncertain data of items may also be given by a (finite) list of M possible scenarios $\mathcal{S}_\mathcal{F} := \{a^1, \ldots, a^M\}$, where each scenario $a^i (i = 1, \ldots, M)$ corresponds to an outcome of the vector $a = (a_1, \ldots, a_n)$ of the items' associated data such as their weights or lengths. The outcomes of vector a may either be based on the expertise of practitioners or stem from a probabilistic analysis.

- *Stochastic uncertainty:* Items' data may be distributed to some distribution function on an underlying set. For instance, the weight w_i of an item i can be uniformly distributed on an interval $[w_i^{min}, w_i^{max}]$, or belongs to some weight class (e.g. heavy, medium, light) with some probability.

1.1.5 Storage objectives

Depending on the context of the storage loading problems, various storage objectives may be involved in the loading process. A comprehensive list of objective functions in storage problems is given in [64]. In the scope of this thesis, we are interested in storage loading problems with the following objectives.

- In case that some hard stacking constraints are given, the first objective is to decide whether all items can be feasibly stored into the storage area regarding all these stacking constraints as well as the capacity of the storage area and the storage precedence of the items. If this is possible, the next step would be to assign each item to a feasible location minimizing the total number of used stacks (to have more flexibility for storing items arriving later on), or minimizing the total number of items stacked at levels above the ground level (to reduce the risk of reshuffling).

- In case that some soft stacking constraints are given, the objective is to store all items minimizing the number of unordered stackings, or minimizing the violation with respect to the soft stacking constraints. A particular case study is when stacking constraints on departure time are given as soft constraints. In that situation, as discussed in Section 1.1.3, unordered stackings lead to a number of relocations of items when retrieving the items according to the desired retrieval order, and minimizing the number of these induced relocations is also an interesting storage objective.

We use the following notations (as proposed in [64]) to denote the mentioned storage objectives:

- '$-$': Decide whether all items can be feasibly stored into the storage area regarding all given hard stacking constraints.
- $\#St$: Store all items regarding all given hard stacking constraints, minimizing the number of used stacks.
- $\#SI^{>1}$: Store all items regarding all given hard stacking constraints, minimizing the number of items stacked above the ground level.
- $\#US$: Store all items minimizing the number of unordered stackings with respect to given soft stacking constraints.

1.1.6 A three-field notation

In [64] the authors propose a three-field notation $\alpha \mid \beta \mid \gamma$ for a classification scheme of storage loading, unloading, and premarshalling problems. We apply and extend that three-field notation to briefly represent the storage loading problems under our consideration. More precisely, the content and meaning of the fields in the notation are as follows.

- The first field α contains the problem type (L for loading) and information about the parameters of the storage area (e.g. the number of stacks m or the number of levels per stack b). We denote $b = b'$ if the number of levels per stack is given by a fixed number $b' \in \mathbb{N}$. If b is given as part of the input, then we omit this subfield.
- The first part of the second field β contains the information about storage precedence of the items (as described in Section 1.1.2).
- The second subfield of β gives information about stacking constraints. We add an underline symbol under the character s in the notation of stacking constraints to indicate that the stacking constraints are soft. For hard stacking constraints the underline symbol is omitted. For representing combined stacking constraints we use the symbol '&' between the notation of its single components. If no stacking constraint is given, then we omit the second sub-field of β.
- In cases of storage loading problems with uncertain data, the third part of β specifies the type of data uncertainty by using notations introduced in Section 1.1.4. Furthermore, to indicate which parameter is uncertain, we add a tilde symbol above the character corresponding to that parameter in the first and second subfields of β. We use the notation $\tilde{s}_{ij}$ to mean that the encoded matrix of the stacking constraints s_{ij} contains 'uncertain' entries. To emphasize that the items in a set have uncertain data, we add the tilde symbol above the name of that set. For deterministic storage loading problems, the third part of β is omitted.
- The third field γ specifies the storage objective as listed in Section 1.1.5.

For example, consider the storage loading problem represented by

$$L, b = 2 \mid I^{fix} \to I^1 \to \tilde{I}^2, s_{ij}(\tilde{w}, \searrow)\&\underline{s}(\ell), \mathcal{S}_{\mathcal{I}} \mid -.$$

In this problem, each stack in the storage area can contain at most $b = 2$ items. The storage precedence of items is given by the sequence $I^{fix} \to I^1 \to I^2$, and the appearance of I^{fix} in this sequence means that there are some items already stored in the storage area. The loading process must satisfy the hard stacking constraints on weight in the most stable sense and should satisfy the soft stacking constraints on length (cf. Section 1.1.3). Here the tilde symbols mean that the weight of items in I^2 are uncertain. Furthermore, the notation $\mathcal{S}_\mathcal{I}$ in the description of the problem tells us that the weights of I^2-items vary in some intervals. According to the notation '–' in the γ field, the storage objective of this problem is to find a feasible stacking solution in every scenario of weights of I^2-items.

As another example, consider the storage loading problem represented by

$$L \mid \tilde{I}, \tilde{s}_{ij}, \mathcal{S}_\mathcal{D} \mid \#St.$$

In this case, the maximum number of items per stack b is given as a part of the input. A set of items I with uncertain data needs to be stored into the storage area. The loading process must satisfy the hard stacking constraints encoded by stacking matrix (s_{ij}). Due to the uncertain data of the items, the stacking matrix contains entries with uncertain values (therefore the tilde symbols take part in the description of the problem). Furthermore, the notation $\mathcal{S}_\mathcal{D}$ in β field here means that the uncertainty on data of items is given in form of a list of possible stacking matrices (cf. Section 1.1.4). As denoted by '$\#St$' in the γ field, the storage objective of this problem is to find a stacking solution for each stacking matrix scenario minimizing the number of used stacks.

1.2 Robust optimization

As we have mentioned in Section 1.1.4, storage managers may have to face up with the uncertainty of items' data when making optimal loading decisions. Optimization under uncertainty can therefore find its application in the context of storage loading problems. A major trend of dealing with data uncertainty in the optimization process is *robust optimization*. In this section we recall the principles and formal descriptions of some robust optimization concepts to be used in this thesis.

The main research objects of robust optimization are *uncertain optimization problems* (see e.g. [10, 19]). More precisely, such an uncertain optimization problem is a parameterized family $(P_\xi)_{\xi\in\mathcal{U}}$ of optimization problems corresponding to $\xi \in \mathcal{U} \subset \mathbb{R}^p$ (for some $p \in \mathbb{N}$):

$$\begin{array}{lll} (P_\xi) & \min & f(x) \\ & \text{s.t.} & F(x,\xi) \le 0, \\ & & x \in \mathcal{X}, \end{array}$$

where

- ξ is the parameter vector representing data elements of the problem (P_ξ),
- $\mathcal{U}$ is the set consisting of considered values of parameter ξ,
- x is the decision vector,
- $\mathcal{X} \subset \mathbb{R}^n$ is the variable space (here n is the dimension of the space),

- $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function,
- $F(\cdot, \xi) : \mathbb{R}^n \to \mathbb{R}^m$ (for some $m \in \mathbb{N}$) is the function describing the constraints of (P_ξ) for any fixed $\xi \in \mathcal{U}$.

To be precise, the vector inequality $F(x, \xi) \leq 0$ in the description of constraints of (P_ξ) means that $F_i(x, \xi) \leq 0$ for $i = 1, \dots, m$, in which the functions F_i's are the components of the vector function F. Note that in the statement of (P_ξ) the objective function is independent of data vector ξ. This is only a technical assumption. Indeed, if the objective function of (P_ξ) is represented by a data-dependent function $f(x, \xi)$, we can make the objective data-independent by adding a new variable t to be minimized and adding the constraint $f(x, \xi) - t \leq 0$ to the list of original constraints.

The set $\mathcal{U}$ is called *uncertainty set* and its elements are called *scenarios*. The uncertainty set can be given by disturbing a so-called *nominal scenario*. The nominal scenario can also refer to the most likely value of data vector ξ. The optimization problem $(P_{\bar{\xi}})$ corresponding to the nominal scenario $\bar{\xi} \in \mathcal{U}$ is called the *nominal problem* of the uncertain optimization problem $(P_\xi)_{\xi \in \mathcal{U}}$. An optimal solution to the nominal problem $(P_{\bar{\xi}})$ is called a *nominal solution* to $(P_\xi)_{\xi \in \mathcal{U}}$.

Roughly speaking, the spirit of robust optimization is to find a robust solution to the uncertain optimization problem, i.e., a solution that is immunized against uncertainty. Different robustness concepts are introduced in the literature, all of them share the following common assumptions on the decision-making environment:

- No probability distribution on the uncertainty set $\mathcal{U}$ is given.
- Constraint violation is forbidden for any realization of the data vector ξ in the uncertainty set $\mathcal{U}$, i.e., the constraints of (P_ξ) are hard for every $\xi \in \mathcal{U}$.
- The decisions are made when and only when the actual data is within the prespecified uncertainty set $\mathcal{U}$.

In the following we describe the basic ideas of some robustness concepts that are of our interest in this thesis. For convenience of the description, we denote

$$\mathcal{F}(\xi) := \{x \in \mathcal{X} \mid F(x, \xi) \leq 0\},$$

i.e., $\mathcal{F}(\xi)$ is the feasible set of (P_ξ).

1.2.1 Strict robustness

This concept was originally introduced in [13]. The key feature of the strict robustness concept is that all decision variables represent here-and-now decisions, i.e., the solution to the uncertain optimization problem must be decided before the actual data becomes known. Once the solution is decided, we are sticked with the solution, i.e., nothing changes in the solution when the actual data is realized.

To be precise, a solution $x \in \mathcal{X}$ to the uncertain optimization problem $(P_\xi)_{\xi \in \mathcal{U}}$ is called *strictly robust feasible* if it is feasible for all scenarios in the uncertainty set, i.e., if $x \in \mathcal{F}(\xi)$ for all $\xi \in \mathcal{U}$. The *strictly robust counterpart* of the uncertain optimization problem is given as

$$(SR) \qquad \min \quad f(x) \qquad \text{s.t.} \quad x \in SR(\mathcal{U}),$$

in which $SR(\mathcal{U})$ is the set consisting of all strictly robust feasible solutions with respect to the uncertainty set $\mathcal{U}$, i.e.,

$$SR(\mathcal{U}) = \bigcap_{\xi \in \mathcal{U}} \mathcal{F}(\xi).$$

An optimal solution to (SR) is called a *strictly robust optimal solution* to the uncertain optimization problem $(P_\xi)_{\xi \in \mathcal{U}}$.

By definition, the concept of strict robustness tends to be over-conservative. To overcome the conservatism of strict robustness, other robustness concepts have been introduced in the literature including adjustable robustness.

1.2.2 Adjustable robustness

This concept was originally introduced in [11]. It applies to the context in which some decision variables must be set before knowing the actual data (here-and-now variables), while the other decision variables can be adjusted when the actual data is realized (wait-and-see variables). This means that adjustable robustness is a two-stage concept of robust optimization: in the first stage the here-and-now decision is determined, then the wait-and-see decision is made in the second stage.

More precisely, the decision vector x can be split as $x = (y, z)$ with $y \in \mathcal{Y} \subset \mathbb{R}^{n_1}$ and $z \in \mathcal{Z} \subset \mathbb{R}^{n-n_1}$ for some $n_1 \in \{1, \ldots, n-1\}$, where the variables y need to be determined before knowing the actual value of $\xi \in \mathcal{U}$, while the variables z may be determined after the actual realization of ξ. We can rewrite the problem (P_ξ) as follows.

$$\begin{array}{lll} (P_\xi) \qquad & \min & f(y,z) \\ & \text{s.t.} & F(y,z,\xi) \leq 0, \\ & & (y,z) \in \mathcal{Y} \times \mathcal{Z}. \end{array}$$

Naturally, when making a decision on here-and-now variables y, one has to make sure that the optimization problem (P_ξ) is feasible once the actual data $\xi \in \mathcal{U}$ is realized. Such a value of a here-and-now decision vector y is called an *adjustable robust feasible solution* to the uncertain optimization problem $(P_\xi)_{\xi \in \mathcal{U}}$. The set of the adjustable robust feasible solutions therefore can be given by

$$\begin{aligned} AR(\mathcal{U}) &= \{y \in \mathcal{Y} \mid \forall \xi \in \mathcal{U}\ \exists z \in \mathcal{Z}\ :\ (y,z) \in \mathcal{F}(\xi)\} \\ &= \bigcap_{\xi \in \mathcal{U}} Pr_{\mathcal{Y}}(\mathcal{F}(\xi)), \end{aligned}$$

where $Pr_{\mathcal{Y}}(\mathcal{F}(\xi))$ denotes the projection of $\mathcal{F}(\xi)$ on $\mathcal{Y}$, i.e.,

$$Pr_{\mathcal{Y}}(\mathcal{F}(\xi)) := \{y \in \mathcal{Y} \mid \exists z \in \mathcal{Z}\ :\ (y,z) \in \mathcal{F}(\xi)\}.$$

The strategy when using the adjustable robustness concept is: when in doubt, assume the worst. Mathematically speaking, given an adjustable robust feasible solution $y \in AR(\mathcal{U})$, the worst-case objective value of the uncertain optimization problem over all considered data scenarios is

$$f_{ar}(y) := \sup_{\xi \in \mathcal{U}} \inf_{z \in \mathcal{F}(\xi)} f(y,z).$$

The *adjustable robust counterpart* of $(P_\xi)_{\xi\in\mathcal{U}}$ is then defined by

$$(AR) \qquad \begin{array}{rl} \min & f_{ar}(y) \\ \text{s.t.} & y \in AR(\mathcal{U}). \end{array}$$

An optimal solution to (AR) is called an *adjustable robust optimal solution* to the uncertain optimization problem $(P_\xi)_{\xi\in\mathcal{U}}$.

1.2.3 Literature review on robust optimization

The beginning of robust optimization traces back to the work of Soyster [74]. In this work the author investigates the strictly robust approach for linear programming problems under column-wise uncertainty. More formally, he considers the following *generalized linear program*

$$\begin{array}{rll} \min & c^T x & \\ s.t. & Ax \leq b & \forall A \in \mathcal{A}, \\ & x \geq 0, & \end{array}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m$ (here n and m are dimensions of the corresponding vectors), and $\mathcal{A}$ is the set of all matrices A whose column vectors a_i belong to a given compact and convex set K_i (for $i = 1, \dots, n$). The inequalities in the description of the contraints of the program are componentwise vector inequalities. The author proves that a robust optimal solution to this uncertain problem can be obtained by solving the following deterministic linear program

$$\begin{array}{rl} \min & c^T x \\ s.t. & \bar{A}x \leq b, \\ & x \geq 0, \end{array}$$

where $\bar{A} = (\bar{a}_{ij})$ with $\bar{a}_{ij} = \max_{a_j \in K_j} a_{ij}$ for $i = 1, \dots, n$ and $j = 1, \dots, m$. Subsequent works on the topic of generalized linear programs are done in [41, 75, 78].

Since the late 1990s, a strong theoretic framework for robust optimization has been built, in which the central theoretical issue is tractability of robust optimization models with different robustness concepts. As defined in [10], tractable optimization problems are the ones that can be reformulated into equivalent problems for which there are known solution algorithms with worst-case running time polynomial in a properly defined input size. For instance, it is proven in [14] that the strictly robust counterpart of an uncertain linear programming problem with ellipsoidal uncertainty set (defined also in [14]) is equivalent to a conic quadratic program (cf. [16]), which can be solved in polynomial time by interior point methods. Tractability of the robust counterpart of an uncertain optimization problem depends on the structure of the nominal problem as well as the class of uncertainty set. Beside considering ellipsoidal uncertainty sets as we have mentioned, tractability of the strictly robust counterparts of linear programs under some other types of uncertainty is also discussed in [14, 15, 20]. For the strict robustness concept, tractability and intractability of robust counterparts of different classes of convex programs under different types of uncertainty set are proven in [13, 17, 24, 44, 45]. A systematic way to convert the strictly robust counterpart of a nonlinear uncertain inequality that is concave in the uncertain parameters into an explicit and computationally tractable set of constraints is provided in [12]. For adjustable robustness concepts, tractability of robust counterparts of some uncertain optimization problems is the topic of study in e.g. [11, 23, 77].

Robust optimization is a fruitful area not only from theoretical but also from practical point of view. Even when restricted on the robustness concepts introduced in Section 1.2, there are plenty of papers on applications of robust optimization in daily decision-making problems. In [29] one can find how strictly and adjustable robust solutions improve the load planning of trains in intermodal transportation. Real-life applications of adjustable robustness can furthermore be found in e.g. network flow and design problems (see [8]), circuit design (see [66]). We refer to [10, 19, 61, 42] for comprehensive surveys on a wide spectrum of applications of robustness concepts.

1.3 Literature review on storage loading problems

As shown in the survey [64], storage loading problems in the context of stack-based storage areas are attractive and active topics of interest. Storage loading problems under deterministic setting are studied in quite a few papers. Complexity results of several deterministic storage loading problems with stacking constraints are presented in [31]. Also in this paper, polynomial time algorithms for some of these problems are provided, and the boundary to NP-hardness is established. In [27] the authors consider a case study of deterministic storage loading problems, where a sequence of incoming items has to be stored into a stack-based storage area. There, the stacks have the same number of levels, and each item has an associated value representing its priority of retrieval. The objective is to minimize the number of unordered stackings with respect to retrieval order of the items. The authors prove strong NP-hardness of this problem, and propose different exact and heuristic solution procedures.

In the literature, storage loading problems under uncertainty are often studied in the context of container terminals and container ships. There, the weights of items (containers) are often assumed to be uncertain. Most academic papers on these problems handle the uncertainty on the weight of items by categorizing the items in weight groups.

The first paper on storage loading problems having uncertain data is [59]. In this paper, the authors study the problem of storing a sequence of export containers into a stack-based stacking yard, where each container belongs to one of three weight groups: heavy, medium, light. These containers arrive one after one from arriving trucks and will be loaded onto a vessel afterwards. Therefore, the stacking constraints on weight in the reverse-stable sense $s_{ij}(w, \nearrow)$ are applied (cf. Section 1.1.3). The weight group of each item is not known at the time of its arrival, and the distribution of the weight groups is estimated from past empirical data. The objective is to determine the storage location for an incoming item minimizing the expected number of relocations needed in the vessel-loading phase. As a solution approach, a dynamic programming model is formulated, in which the number of expected relocations is computed by taking into account two values: the probability of the weight group of the next arriving item, and the marginal expected number of relocations that become necessary when an item of a specific weight group is assigned to a specific stack. To support real-time decisions, a decision tree is developed from the set of the optimal solutions to the dynamic program and a classification procedure including selection criteria of the key attributes for branching, a pruning rule, and a simplification method. An error in the derivation of the objective function in the dynamic program is shown in [86], and a correction is also provided in this paper.

A similar problem is studied in [57] under the following assumptions. First, in the loading phase of containers into the stacking yard, we only know the estimated weight group of each container. Second, the accurate information about the weight group of all containers is only known in the vessel-loading phase later on. Third, the conditional probability of the actual weight groups given the estimated weight groups is known from the past statistics of managers. The authors propose a simulated annealing algorithm and develop a decision tree learning algorithm to find a good stacking strategy for the sequence of incoming containers, in which the conditional probability is used to compute the expected number of relocations during the vessel-loading phase.

Storage loading of containers in a stacking yard with the objective of minimizing the expected number of reshuffles is the topic of interest in [46]. In this paper the authors consider a storage loading problem in which a sequence of containers has to be stored completely into a stacking yard, in order to load onto several ships later on according to a sequence of $|C|$ sets, where C is the set of all container types. The type of a container is defined by its weight group, the ship it will be loaded onto, and its port of destination. The probability that an incoming container having type $c \in C$ is proportioned to the size of the corresponding ship, and uniformly distributed over all the types. Based on the dynamic programming model proposed in [59], the authors develop a heuristic algorithm to quickly find near-optimal solutions for large-scale problems with a realistic size of the stacking yard. The algorithm maps all states of the dynamic program onto decision trees, recognizes the correlation between the decisions made by the program for different states, then simplifies and generalizes the decision trees into generalized but simpler ones. The generalized trees can quickly locate incoming containers into the stacking yard, and can be extended to include new information provided by the dynamic program.

A case study of storage loading problems, in which a set of containers needs to be loaded onto a container ship, is considered in [35]. Due to stability reasons of the ship, weight and height restrictions are given for each stack. Location restrictions for some reefer containers, that need power plugs, are given as well. The objective function to be minimized is a weighted sum of four components: penalizing unordered stackings, penalizing the storage of non-reefer containers in a power plug location, storing different kinds of containers in the same stack, and opening a stack. By a reduction from the bin packing problem, the authors prove that the sub-problem of minimizing the number of opened stacks is NP-hard. They also provide an integer programming and a constraint programming model as solution methods to the problem. It is shown by their experimental results on real-life instances that the constraint programming model has better performance.

1.4 Thesis outline

The goal of this thesis is to study some storage loading problems motivated from several practical contexts, in which the data of some items are subjected to different types of uncertainty. We focus mainly on finding and analyzing strictly and adjustable robust stacking solutions to the problems under our consideration. The main contribution of this thesis is contained in Chapters 2-5. It is a compilation of results from two published papers (coauthored with Sigrid Knust [62], and with Marc Goerigk and Sigrid Knust [49]) and a working paper (coauthored with Christina Büsing and Sigrid Knust [32]).

Chapters 2-4 concern different versions of a storage loading problem with stacking constraints. In this problem, incoming items arriving at a partly filled storage area have to be assigned to stacks under the restriction that not every item may be stacked on top of every other item and taking into account that some items will arrive later. We consider some robust versions of this problem where the actual data of the items arriving later may differ according to some uncertainty set, while the actual data of the other items are known exactly. We focus on minimizing the number of used stacks, which is a common objective in practice to have more flexibility to store items arriving later into the remaining stacks. The approach of strict robustness requires that the stacking configuration of all items has to be fixed in advance, while the approach of adjustable robustness allows that the stacking configuration of the items arriving later can be adjusted depending on their realization. In Chapter 2 we give complexity results for some particular cases of the deterministic and robust problems under discrete or interval uncertainty. Chapter 3 presents different mixed integer programming formulations for the deterministic version and the robust counterparts of the uncertain problems. Based on our numerical experiments, we give guidelines which formulation performs best under which setting, and compare the adjustable robust stacking solutions with the strictly robust ones in terms of the optimal objective values. In Chapter 4 we study a version of the storage loading problem, in which the items coming later are subjected to stochastic uncertainty. We discuss some rule-based scenario generations to derive different uncertainty sets from stochastic data of the items, and analyze the impact of building different scenario sets on the trade-off between robustness and cost of the robust stacking solutions.

Chapter 5 is devoted to studying a storage loading problem appearing in the context of container ships. Items with uncertain weights have to be loaded onto a ship regarding some stacking constraints. In addition, payload constraints are taken into account to ensure the stability of the ship. The objective is to minimize the total payload violation over all stacks in order to reduce the shipping cost. The approach of strict robustness requires that a complete stacking solution has to be fixed in advance before the actually realized scenario becomes known. The approach of adjustable robustness determines in advance which item is assigned to which stack, but allows to choose the level of the item within the stack depending on the weight scenario of all items later. We develop exact decomposition and heuristic solution algorithms to solve the robust counterparts of the problem with finite and interval-based uncertainty.

Chapter 6 closes this thesis by some conclusions, reciting the main results, and stating some open questions.

Chapter 2

Complexity

In this chapter we discuss complexity of some deterministic and robust storage loading problems. We refer the reader to [43] for a thorough introduction into complexity theory.

In Section 2.1 we consider some particular cases of the deterministic problem

$$(P) \qquad L \mid I^{fix} \to I^1 \to I^2, s_{ij} \mid \#St.$$

This problem is motivated from practical contexts where practitioners have to deal with a sequence of sets of incoming items. For instance, several trains or vessels consecutively arrive at a container terminal, each one containing a set of containers that have to be loaded into the storage yard. According to the order of arrival, all items of one set have to be completely stored before storing any item of the next set. This corresponds to the situation that one has to store a set of items into a partly filled storage area, taking into account that some items will arrive later. Here I^{fix} denotes the set of items that are already stored in the storage area, I^1 stands for the set of items to be stored now, and I^2 refers to the set of items coming later. The loading process must satisfy hard stacking constraints (s_{ij}) minimizing the number of used stacks to have more flexibility for the remaining stacks to store items arriving later on.

In Section 2.2 we consider some robust versions of (P) where data uncertainty is taken into account. More precisely, we follow the idea that the practitioner knows exactly the actual data of items that are already stored in the storage area or going to be loaded now (i.e. items in $I^{fix} \cup I^1$), while the actual data of items arriving later (i.e. I^2-items) may differ according to some possible scenarios. We will show that complexity results for solving strictly and adjustable robust counterparts of some particular cases can be derived from the ones for their corresponding deterministic versions.

2.1 Some deterministic storage loading problems

In Table 2.1 we summarize known complexity results for some particular cases of the deterministic storage loading problem (P).

No.	Problem	Complexity	Reference
1	$L, b = 2 \mid I^{fix} \to I^1 \to I^2, s_{ij} \mid \#St$	$\mathcal{O}(n^{2.5})$	Corollary 3 [31]
2	$L, b = 3 \mid I^1, s_{ij}\ transitive \mid -$	strongly NP-complete	Theorem 4 [31]
3	$L \mid I^{fix} \to I^1, s_{ij}\ total\ order \mid \#St$	$\mathcal{O}(n \log n)$	Theorem 7 [31]
4	$L \mid I^1 \to I^2, s_{ij}\ total\ order \mid \#St$	$\mathcal{O}(n \log n)$	Theorem 8 [31]

Table 2.1: Known complexity results for some deterministic storage loading problems.

In this section we discuss new complexity results for some other particular cases of (P). More precisely, in Section 2.1.1 we consider the following problem

$$(P_1) \qquad L, b = 2 \mid I^{fix} \to I^1 \to I^2, s_{ij} \textit{ total order} \mid \#St.$$

This problem arises as a subsequent problem when solving the adjustable robust counterpart of the uncertain storage loading problem with interval uncertainty $(P_{\mathcal{I}})$ in case $b = 2$ (as we will show in Corollary 2.19). Note that (P_1) is nothing but the first problem in Table 2.1 with an additional assumption that the stacking constraints s_{ij} are total order. We show that (P_1) can be solved more efficiently than the one without this additional assumption by an algorithm of complexity $\mathcal{O}(n \log n)$. In Section 2.1.2 we show that the problem

$$(P_k) \qquad L, b = k \mid I^1, s_{ij} \textit{ transitive} \mid -$$

is strongly NP-complete for each fixed $k \geq 3$, even for transitive stacking constraints s_{ij}. This problem is a generalization of the second problem in Table 2.1, which considers only the case $k = 3$.

Note that if m^* is the optimal number of used stacks in (P), then the problem is feasible if and only if $m \geq m^*$, where m is the given number of stacks in the storage area. This means that the answer to the feasibility version of (P), i.e.

$$L \mid I^{fix} \to I^1 \to I^2, s_{ij} \mid -,$$

is straightforward from an optimal solution to (P). Furthermore, if we start with a feasible solution to (P) and re-allocate as many items as possible to the ground level until all stacks contain at least one item, then we obtain an optimal solution to

$$L \mid I^{fix} \to I^1 \to I^2, s_{ij} \mid \#SI^{>1}.$$

Relocating items to the ground levels of empty stacks requires $\mathcal{O}(n)$ time. Therefore, if a particular case of (P) is polynomially solvable, then the same result holds for its corresponding versions where the storage objective $\#St$ is replaced by '–' or $\#SI^{>1}$.

2.1.1 A polynomially solvable case

As shortly mentioned above, in this subsection we present an algorithm of complexity $\mathcal{O}(n \log n)$ to solve the problem

$$(P_1) \qquad L, b = 2 \mid I^{fix} \to I^1 \to I^2, s_{ij} \textit{ total order} \mid \#St.$$

For conveniently describing our algorithm, we define a comparator $\mathrel{\circeq}$ for each pair of items (i, j) based on the total order stacking constraints (s_{ij}) as follows:

$$i \mathrel{\circeq} j := \begin{cases} i \approx j & \text{if } s_{ij} = 1 \text{ and } s_{ji} = 1, \\ i \prec j & \text{if } s_{ij} = 1 \text{ and } s_{ji} = 0, \\ i \succ j & \text{if } s_{ij} = 0 \text{ and } s_{ji} = 1. \end{cases}$$

Note that the case "$s_{ij} = 0$ and $s_{ji} = 0$" cannot happen since all items can be compared because of the total order stacking constraints (s_{ij}). Additionally, we use notation $i \preccurlyeq j$ to indicate the case in which $s_{ij} = 1$ (i.e., either $i \approx j$ or $i \prec j$). Similarly, notation $i \succcurlyeq j$ is used to indicate the case $s_{ji} = 1$ (i.e., either $i \approx j$ or $i \succ j$).

We shall use the following terminologies (with respect to the comparator $\mathrel{\mathring{=}}$):

i is equal to j	if $i \approx j$,
i is strictly smaller than j	if $i \prec j$,
i is strictly larger than j	if $i \succ j$,
i is smaller than j	if $i \preccurlyeq j$,
i is larger than j	if $i \succcurlyeq j$.

We say that an item i is the largest (resp., smallest) item in a given set J if $i \in J$ and i is larger (resp., smaller) than all other items in J.

An important procedure in our algorithm to solve (P_1) is the process of greedily storing a set of items J into a storage area that already has some items stored before. We shall denote this procedure by $G(c \to J)$, where c denotes the set of non-empty stacks in the current stacking configuration of the storage area. The stacks in c that are fully filled cannot store any items from J, therefore without loss of generality we can ignore them and assume that all stacks in c are only partly filled. Note that each stack in c can store at most one item in J due to the restriction $b = 2$. We take all items contained in stacks of c, together with the ones in J, and sort them non-decreasingly with respect to the comparator $\mathrel{\mathring{=}}$. This can be done thanks to the total order stacking constraints s_{ij}.

The key idea in $G(c \to J)$ is to adapt the algorithms presented in the proofs of Theorems 7 and 8 from [31], that are applied to solve Problems 3 and 4 in Table 2.1. Precisely, $G(c \to J)$ is executed as follows. We start with a stack in c that contains one of the smallest items i^* (in the order after sorting), fill up this stack with one of the smallest feasible items in J (i.e. an item $i \in J$ with $i \preccurlyeq i^*$ and $i \preccurlyeq j$ for all $j \in J$). If there is no such J-item, then there is no change in the stack. We continue with the remaining items in J and a stack containing one of the smallest items among the remaining items in c, and so on. When all stacks in c have been processed, let r be the number of remaining items in J. If $r = 0$, then all J-items are already stored into stacks in c. Otherwise, $r > 0$, and let $i_1 \preccurlyeq i_2 \preccurlyeq \ldots \preccurlyeq i_r$ be the remaining J-items sorted in non-decreasing order with respect to the comparator $\mathrel{\mathring{=}}$. We store these remaining J-items into empty stacks as follows. Each pair of items i_{2k-1} and i_{2k} (for $k = 1, \ldots, \lfloor \frac{r}{2} \rfloor$) is stored into an empty stack, where the larger item i_{2k} is located at the ground level with the smaller item i_{2k-1} on top. If r is odd, then the largest item i_r is stored at the ground level of some empty stack.

According to the above strategy, whenever an item i is stacked on another item j, we have $i \preccurlyeq j$ which means $s_{ij} = 1$. Therefore the stacking constraints are always regarded. Following the proofs of Theorems 7 and 8 from [31], it is obvious that this stacking strategy minimizes the number of used stacks when stacking $c \to J$. Furthermore, this stacking strategy ensures that as many stacks in c as possible are fully filled by J-items. We denote by $g(c \to J)$ the stacking result after executing $G(c \to J)$, and by $\#St(c \to J)$ the number of used stacks in $g(c \to J)$.

Our strategy to solve (P_1) to optimality can be sketched in the following steps.

Step 1: Sort all items in $I^{fix} \cup I^1 \cup I^2$ non-decreasingly according to $\mathrel{\mathring{=}}$.
Step 2: Store I^1-items keeping the largest flexibility for storing I^2-items.
Step 3: Greedily store I^2-items.

The first step can be done in $\mathcal{O}(n \log n)$. We first discuss how **Step 3** is executed. Let c^* be the stacking configuration of $I^{fix} \to I^1$ after executing **Step 2**. According to the above discussion, whatever the stacking configuration c^* is, the best strategy to store I^2-items is $G(c^* \to I^2)$. This greedy stacking strategy always satisfies the stacking constraints (s_{ij}) and requires minimum number of stacks to store I^2-items. Therefore, in **Step 3** we store I^2 according to $G(c^* \to I^2)$. Since the sorting of all items has been done in **Step 1**, the storage operations of I^2-items in **Step 3** can be implemented in $\mathcal{O}(n)$ time.

We now discuss **Step 2** in detail. At first we note that greedily storing I^1 into the storage area with the appearance of I^{fix} may not lead to an optimal stacking solution to (P_1), as shown in the following example.

Example 2.1. *Consider the case of stacking $I^{fix} \to I^1 \to I^2$ in a storage area with $b = 2$, in which $I^{fix} = \{i_1\}$, $I^1 = \{i_2, i_3\}$, $I^2 = \{i_4\}$, and $i_3 \prec i_2 \prec i_4 \prec i_1$. Since $b = 2$ and there are 4 items, we need at least 2 stacks to store these items. Figure 2.1 (a) illustrates an optimal stacking solution to this case, which requires 2 stacks. If we greedily store I^1-items according to $G(I^{fix} \to I^1)$, then the best stacking configuration for I^2 is illustrated in Figure 2.1 (b). It needs 3 stacks and therefore is not an optimal stacking solution.*

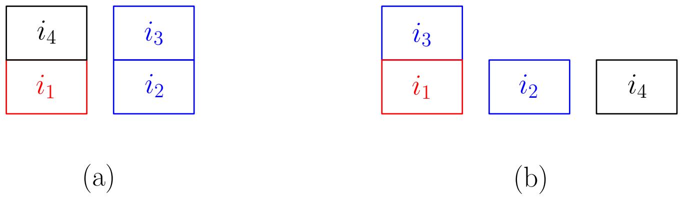


Figure 2.1: The greedy algorithm does not give an optimal solution to (P_1).

Consider the stacking configurations of $I^{fix} \to I^1$ in the two stacking solutions illustrated in Example 2.1. Denote by c_a (resp., c_b) the stacking configuration of $I^{fix} \to I^1$ in the former (resp., latter) stacking solution. Each of c_a and c_b has one empty location: for c_a that is the one above item i_1, while for c_b that is the one above item i_2. The empty location in c_a can store an I^2-item that is strictly larger than i_2 but at most as large as i_1, while the empty location in c_b cannot (since $i_2 \prec i_1$). This means that the empty location in c_a gives more possibilities for storing I^2 than the one of c_b. Therefore, in any case of I^2-item i_4, using c_a will not lead to a worse optimal objective value than using c_b, i.e.,

$$\#St(c_a \to \{i_4\}) \leq \#St(c_b \to \{i_4\}),$$

as precisely shown in the following.

- If $i_4 \succ i_1$, then $\#St(c_a \to \{i_4\}) = \#St(c_b \to \{i_4\}) = 3$.
- If $i_4 \succ i_2$ and $i_4 \preccurlyeq i_1$, then $\#St(c_a \to \{i_4\}) = 2 < \#St(c_b \to \{i_4\}) = 3$.
- If $i_4 \preccurlyeq i_2$, then $\#St(c_a \to \{i_4\}) = \#St(c_b \to \{i_4\}) = 2$.

Motivated by the above observation, we come up with the following general definition, in which we consider the context of stacking two sets of items $J^1 \to J^2$ with the total order stacking constraints s_{ij}.

Definition 2.2. *Let c_1 and c_2 be two stacking configurations of J^1. We say that c_1 is more flexible than c_2 (or c_2 is less flexible than c_1) if for all possibilities of J^2 we have*

$$\#St(c_1 \to J^2) \leq \#St(c_2 \to J^2).$$

We say that these two stacking configurations have the same flexibility (or c_1 is as flexible as c_2) if for all possibilities of J^2 we have

$$\#St(c_1 \to J^2) = \#St(c_2 \to J^2).$$

By this definition, the goal of **Step 2** mentioned above is to find the most flexible stacking configuration of $I^{fix} \to I^1$. To do that, the observations in the following lemma are helpful. All cases mentioned in this lemma are considered in the context where $b = 2$, and they are obvious by following the idea of the discussion after Example 2.1.

Lemma 2.3. *(i) Consider two partly filled stacks, each containing one item. If these two items are equal, then the two stacks have the same flexibility.*

(ii) Consider two partly filled stacks, each containing one item. Then the stack containing the larger item is more flexible than the one containing the smaller item.

(iii) An empty stack, together with a fully filled stack, is more flexible than two partly filled stacks.

We now discuss our strategy to construct a stacking configuration of $I^{fix} \to I^1$ that is most flexible for stacking I^2-items, which is also the goal of **Step 2**. The key ideas are as follows. We start with an initial stacking configuration of $I^{fix} \to I^1$, and step-by-step construct new stacking configurations with better flexibility. The construction terminates when we obtain a stacking configuration that could be proven to have the best flexibility for stacking I^2-items.

To describe precisely our strategy in **Step 2**, for convenience we introduce some concepts and notations. For a *ground item* (resp., *second-level item*) we refer to an item stored at the ground level (resp., the second level) of some stack. Since all items are comparable with respect to the defined comparator $\mathrel{\hat{=}}$, so are the items stored at the ground levels of the stacks. Therefore, given an arbitrary stacking configuration, we can sort the filled stacks non-decreasingly with respect to the comparator $\mathrel{\hat{=}}$ based on their ground items. We say that a non-empty stack q_1 is *before* (resp., *after*) another non-empty stack q_2 if the ground item of q_1 is smaller (resp., larger) than the one of q_2. If q_1 is before (resp., after) all of the other non-empty stacks, then we say that q_1 is the first (resp., the last) stack in the stacking configuration. We use the terminology *I^{fix}-stack* to indicate a stack containing an I^{fix}-item (by default this item is stored at the ground level of the stack). Similarly, for an *I^1-stack* we refer to a stack containing an I^1-item at the ground level. Since $b = 2$, a partly filled I^{fix}-stack (resp., a partly filled I^1-stack) contains only one I^{fix}-item (resp., one I^1-item) at its ground level, while a fully filled I^1-stack stores two I^1-items. Note that there is no stack containing two I^{fix}-items, therefore in an arbitrary stacking configuration of $I^{fix} \to I^1$ all second-level items must belong to I^1.

As briefly mentioned above, an important operation in our strategy for **Step 2** is to modify a stacking configuration of $I^{fix} \to I^1$ to get a new one with better flexibility. There are four types of action to do so.

- *Action 1:* relocate the second-level I^1-item in some stack q to the second level of a partly filled stack before q (as illustrated in Example 2.4).
- *Action 2:* relocate the ground I^1-item in some partly filled stack q to the second level of a partly filled stack after q (as illustrated in Example 2.5).
- *Action 3:* relocate the ground I^1-item in some partly filled stack q and the second-level I^1-item in some stack after q to one stack (as illustrated in Example 2.6).

- *Action 4:* relocate two I^1-items in some fully filled I^1-stack q to the second levels of two partly filled stacks after q (as illustrated in Example 2.7).

Note that in these actions only I^1-items are relocated (since we are not allowed to relocate any I^{fix}-item by assumption). The following examples respectively illustrate these actions.

Example 2.4. *(Action 1) Consider a stacking configuration c of $I^{fix} \to I^1$ having three items stored in two stacks (as illustrated in the left side of Figure 2.2). The fully filled stack consists of item $i_1 \in I^1$ stored at the second level and item $i_3 \in I^{fix} \cup I^1$ stored at the ground level. The partly filled stack consists of an item i_2 stored at its ground level. Assume that $i_1 \prec i_2 \prec i_3$. In this case the stack q containing i_1 and i_3 is after the one containing i_2, and we can relocate i_1 to the top of i_2 without violating the stacking constraints s_{ij}. The new stacking configuration and c have the same number of fully filled stacks and also the same number of partly filled stacks. However, since $i_2 \prec i_3$ and according to Lemma 2.3 (ii), the new stacking configuration, which is illustrated in the right side of Figure 2.2, has better flexibility than c.*

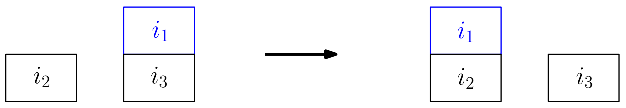


Figure 2.2: Relocate a second-level I^1-item.

Example 2.5. *(Action 2) Consider a stacking configuration c of $I^{fix} \to I^1$ having three partly filled stacks (as illustrated in the left side of Figure 2.3), each of them consists of an item in $\{i_1, i_2, i_3\}$, where $i_1 \in I^1$, $i_2, i_3 \in I^{fix} \cup I^1$, and $i_1 \prec i_2 \prec i_3$. In this case the stack containing i_2 is before the one containing i_3 and after the one containing i_1. The stack q corresponding to i_1 is the first partly filled I^1-stack. By relocating i_1 to the top of either i_2 or i_3, two partly filled stacks in c are replaced by one fully filled stack and one empty stack. Furthermore, the new stacking configuration satisfies the stacking constraints s_{ij} due to $i_1 \prec i_2 \prec i_3$. By Lemma 2.3 (iii), the new stacking configuration has better flexibility than c. Since $i_2 \prec i_3$, by Lemma 2.3 (ii) the new stacking configuration that is most flexible is obtained by moving item i_1 to the top of i_2, i.e., to the top of the first partly filled stack after q, as illustrated in the right side of Figure 2.3.*

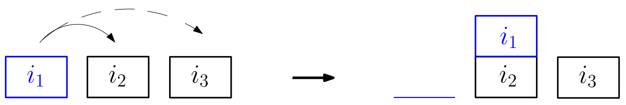


Figure 2.3: Relocate the item in the first partly filled I^1-stack.

Example 2.6. *(Action 3) Consider a stacking configuration c of $I^{fix} \to I^1$ having one partly filled I^1-stack and two fully filled stacks. These stacks contain 5 items $i_1 \preccurlyeq i_2 \preccurlyeq i_3 \preccurlyeq i_4 \preccurlyeq i_5$ as illustrated in the left side of Figure 2.4, where $i_1, i_2, i_4 \in I^1$. The stack q corresponding to i_1 is a partly filled I^1-stack. By relocating i_1 and either i_2 or i_4 to a stack, the new stacking configurations and c have the same number of partly filled stacks and also the same number of fully filled stacks. However, since $i_1 \preccurlyeq i_3 \preccurlyeq i_5$, the empty locations above i_3 or i_5 in the new stacking configurations are more flexible than the one above i_1. Therefore the new stacking configurations are more flexible than c. Furthermore,* *according to Lemma 2.3 (ii), the new stacking configuration that is most flexible is obtained by relocating items i_1 and i_4 (i.e. the second-level item in the last fully filled stack after q) into one stack. We illustrate this stacking configuration in the right side of Figure 2.4.*

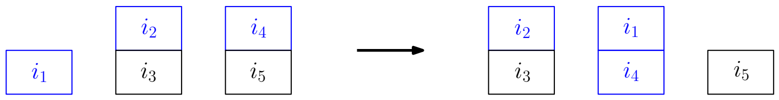


Figure 2.4: Relocate two I^1-items in a partly filled I^1-stack and a fully filled stack.

Example 2.7. *(Action 4) Consider a stacking configuration c of $I^{fix} \to I^1$ having a fully filled stack and three partly filled ones. The fully filled stack consists of item $i_2 \in I^1$ stored at the ground level and item $i_1 \in I^1$ stored at the second level. Each of the three partly filled stacks consists of an item in $\{i_3, i_4, i_5\}$, where these three items are in $I^{fix} \cup I^1$, and $i_1 \preccurlyeq i_2 \prec i_3 \prec i_4 \prec i_5$. In this case the stack q containing i_1 and i_2 is the first fully filled I^1-stack. The order of the stacks in c is illustrated in the left side of Figure 2.5, where for any two stacks, the one on the left is before the one on the right. If we relocate i_1 and i_2 to the second level of two stacks after q, then the stacking constraints s_{ij} are still satisfied. However, one fully filled stack and two partly filled stacks in c are replaced by one empty stack and two fully filled stacks. Therefore, by Lemma 2.3 (iii), the new stacking configuration obtained in that way is more flexible than c. Since $i_3 \prec i_4 \prec i_5$, following Lemma 2.3 (ii), the new stacking configuration that is most flexible is obtained by moving items i_1 and i_2 to the top of i_3 and i_4, i.e., to the second level of the first two partly filled stacks after q.*

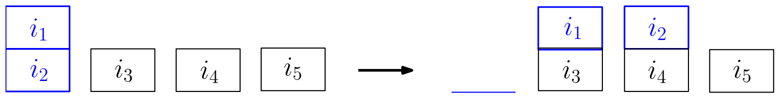


Figure 2.5: Relocate the items in the first fully filled I^1-stack.

It is also important to have a structural initial stacking configuration for **Step 2**. We construct such a stacking configuration as follows. For an I^{fix}-item i^f, we denote by $[i^f]$ the set of I^{fix}-items that are equal to i^f (with respect to the comparator $\triangleq$). The set I^{fix} can be partitioned into disjoint subsets $[i^f_1], \dots, [i^f_k]$ (for some $k \in \mathbb{N}$) where $i^f_1 \prec i^f_2 \prec \dots \prec i^f_k$. For convenience, we shall denote $[i^f_0] := \emptyset$. We partition the set I^1 into subsets $A_0, A_1, \dots, A_k$ as follows:

$$A_0 = \{i \in I^1 \mid i \prec i^f_1\}, \tag{2.1}$$

$$A_u = \{i \in I^1 \mid i^f_u \preccurlyeq i \prec i^f_{u+1}\} \qquad (u = 1, \dots, k-1), \tag{2.2}$$

$$A_k = \{i \in I^1 \mid i^f_k \preccurlyeq i\}. \tag{2.3}$$

In other words, the set A_0 consists of I^1-items that are strictly smaller than the smallest I^{fix}-item, the set A_k contains I^1-items that are larger than the largest I^{fix}-item, while A_u includes all I^1-items that are larger than the u-th smallest I^{fix}-items but strictly smaller than the $(u+1)$-th smallest I^{fix}-items. Let $\bar{c}$ be the stacking configuration that is the union over all $u = 0, \dots, k$ of $g([i^f_u] \to A_u)$. The advantage of using $\bar{c}$ as initial stacking configuration is twofold. Firstly, it can be constructed in $\mathcal{O}(n)$ time once all items are sorted non-decreasingly with respect to $\mathrel{\mathring{=}}$. Secondly, by the construction, $\bar{c}$ has the following property:

($*$) each I^1-item i cannot be relocated to any empty second-level location in any stack before the one containing i.

This property is useful for constructing new stacking configurations with better flexibility in **Step 2**.

Following the above discussion, we come up with Algorithm 1 to construct a stacking configuration of $I^{fix} \to I^1$ that is most flexible for stacking I^2-items. Starting with the initial stacking configuration $\bar{c}$ constructed as above, the algorithm applies Actions 1-4 in an appropriate way to step-by-step construct new stacking configurations with better flexibility, and ends up with a desired stacking configuration.

Algorithm 1 Store I^1-items keeping the largest flexibility for storing I^2-items.

```
1: Partition $I^{fix}$ into subsets $[i^f_1], \dots, [i^f_k]$, where $[i^f]$ is the set of $I^{fix}$-items that are equal
   to $i^f$ and $i^f_1 \prec i^f_2 \prec \dots \prec i^f_k$. Let $[i^f_0] := \emptyset$.
2: Partition $I^1$ into subsets $A_0, \dots, A_k$ according to (2.1)-(2.3).
3: Execute greedy procedures $G([i^f_u] \to A_u)$ for $u = 0, \dots, k$.
4: while there exists a partly filled $I^1$-stack do
5:     Let $q$ be the first partly filled $I^1$-stack.
6:     Let $\mathcal{Q}_1^{\succ q}$ be the set of partly filled stacks after $q$.
7:     if $\mathcal{Q}_1^{\succ q} \neq \emptyset$ then
8:         Relocate the item in stack $q$ to the second level of the first stack in $\mathcal{Q}_1^{\succ q}$.
9:     else
10:        Let $\mathcal{Q}_2^{\succ q}$ be the set of fully filled stacks after $q$.
11:        if $\mathcal{Q}_2^{\succ q} \neq \emptyset$ then
12:            Let $i$ be the item in the second level of the last stack in $\mathcal{Q}_2^{\succ q}$.
13:            Relocate $i$ and the item in stack $q$ into one stack.
14:        else
15:            Break.
16:        end if
17:    end if
18: end while
19: while there exists a fully filled $I^1$-stack do
20:    Let $q_1$ be the first fully filled $I^1$-stack.
21:    Let $\mathcal{Q}^{\succ q_1}$ be the set of partly filled stacks after $q_1$.
22:    if $|\mathcal{Q}^{\succ q_1}| \geq 2$ then
23:        Relocate the items in $q_1$ to the second levels of the first two stacks in $\mathcal{Q}^{\succ q_1}$.
24:    else
25:        Break.
26:    end if
27: end while
28: return most flexible stacking configuration of $I^{fix} \to I^1$.
```

Theorem 2.8. *Algorithm 1 gives a stacking configuration of $I^{fix} \to I^1$ with the largest flexibility in $\mathcal{O}(n)$ time.*

Proof. At first we note that the initial stacking configuration $\bar{c}$ is constructed by Steps 1-3 of Algorithm 1, and Steps 8, 13, 23 of the algorithm respectively correspond to Actions 2, 3, 4 mentioned above. By the choices of I^1-items to be relocated and new locations for these items in each execution of Steps 8, 13, 23, property $(*)$ still holds for the stacking result obtained after each iteration. It follows from the discussion about Actions 2-4 that if a new stacking configuration is generated in some iteration, then it has better flexibility than the previous one.

Starting with the initial stacking configuration $\bar{c}$, each time Step 8 is executed, instead of having two partly filled stacks, we have one more fully filled stack and one more empty stack in the new stacking configuration. Step 8 is repeatly executed until one of the following two cases occurs.

- Case 1: no partly filled I^1-stack exists. In this case, the while loop 4-18 ends.
- Case 2: there is only one partly filled I^1-stack q but $\mathcal{Q}_1^{\succ q} = \emptyset$. In this case, q is not only the unique partly filled I^1-stack but also the last partly filled stack in the current stacking configuration. Therefore, if $\mathcal{Q}_2^{\succ q} = \emptyset$, then q is also the last stack which consists of the largest I^1-item. Otherwise, $\mathcal{Q}_2^{\succ q} \neq \emptyset$, and there are two following possibilities.
  - If the last stack in $\mathcal{Q}_2^{\succ q}$ is a fully filled I^1-stack, then after executing Step 13, this stack becomes the unique partly filled I^1-stack, which consists of the largest item in $I^{fix} \cup I^1$. This item cannot be relocated to any stack to have a more flexible stacking configuration.
  - If the last stack in $\mathcal{Q}_2^{\succ q}$ is an I^{fix}-stack, then after executing Step 13, this stack consists of only one item which is the largest I^{fix}-item, and there is no partly filled I^1-stack in the stacking result.

  Therefore, Step 13 can be executed at most once. Note that the action in Step 13 does not change the number of fully filled stacks and also the number of partly filled stacks, but results in a stacking configuration that is more flexible than the previous one.

In any case, after the while loop 4-18, the stacking result, say $\hat{c}$, has at most one partly filled I^1-stack. If this stack exists, then it must be the last stack in $\hat{c}$. We denote this property of $\hat{c}$ by $(**)$.

The while loop 19-27 is not executed if the stacking configuration $\hat{c}$ has no fully filled I^1-stack. In this situation, except for the largest I^1-item stored in the last partly filled I^1-stack in $\hat{c}$ (if this stack exists due to property $(**)$), all I^1-items are stored at the second levels of some I^{fix}-stacks. This means that $\hat{c}$ already has as many fully filled stacks of storing $I^{fix} \to I^1$ as possible. Keeping this fact in mind, we distinguish two following cases of $\hat{c}$.

- If $\hat{c}$ has no partly filled I^{fix}-stack, then all non-empty stacks in $\hat{c}$ are fully filled, except for the last stack that might be partly filled by the largest item in $I^{fix} \cup I^1$. In this case, it is obvious that $\hat{c}$ is the most flexible stacking configuration of $I^{fix} \to I^1$.
- Otherwise, $\hat{c}$ has p partly filled I^{fix}-stacks (for some $p \in \mathbb{N}$). Let $i_1, \dots, i_p$ be the I^{fix}-items stored in these stacks. Due to property $(*)$, no I^1-item in any stack after the one containing i_k can be relocated to the position on top of i_k (for $k = 1, \dots, p$).

Furthermore, from the construction of $\hat{c}$, any relocation of I^1-items stored in any stack before the one containing i_k (for any $k = 1, \ldots, p$) does not result in a stacking configuration with better flexibility than $\hat{c}$. Thus, the stacking configuration $\hat{c}$ has the largest flexibility of storing $I^{fix} \to I^1$.

If $\hat{c}$ has some fully filled I^1-stacks, then the while loop 19-27 starts with $\hat{c}$. After each execution of Step 23, two partly filled stacks are replaced by one fully filled stack and one empty stack, while no new partly filled I^1-stack arises, i.e., property $(**)$ still holds true for the new stacking configuration. Let c^* be the stacking configuration obtained after all possible executions of Step 23. There are two following cases of c^*.

- In the first case, c^* has no fully filled I^1-stack. By property $(**)$, c^* has at most one partly filled I^1-stack, which (if exists) is the last stack in this stacking configuration. Therefore, except for the largest I^1-item stored in that stack, all I^1-items are now located at the second level of some I^{fix}-stacks. This case of c^* is similar to the situation of $\hat{c}$ discussed above, where $\hat{c}$ has no fully filled I^1-stack. By the same arguments, c^* is the most flexible stacking configuration of $I^{fix} \to I^1$.

- In the second case, c^* has at most one partly filled stack after its first fully filled I^1-stack q_1. This stacking configuration can be partitioned into two parts. The first part consists of stacks before q_1. The second part consists of stack q_1 and the ones after q_1. By property $(**)$, there is no partly filled I^1-stack before q_1, i.e., all I^1-items in the first part of c^* are stored at second levels. Taking into account property $(*)$, all I^1-items in the first part of c^* are optimally stored. In the second part of c^*, there is at most one partly filled stack. If this stack is an I^1-stack, then it must be the last stack in c^* due to property $(**)$, and therefore the I^1-item in this stack cannot be relocated to have better flexibility. If this stack is an I^{fix}-stack, then by property $(*)$ no relocation of I^1-items is possible to have better flexibility. Since all other stacks in the second part of c^* are fully filled, all I^1-items in the second part of c^* are optimally stored. Therefore, c^* is an optimal stacking configuration of $I^{fix} \to I^1$ with the largest flexibility.

We have shown that the stacking result obtained after the last while loop is one of the most flexible stacking configurations of storing $I^{fix} \to I^1$. Constructing the initial stacking configuration $\bar{c}$ by Steps 1-3 can be done in $\mathcal{O}(n)$ time (once all items are sorted non-increasingly with respect to the comparator $\mathrel{\mathring{=}}$). Each of while loops 4-18 and 19-27 requires $\mathcal{O}(m)$ time. Therefore, the algorithm can be implemented in $\mathcal{O}(n)$ time. $\square$

Algorithm 1 describes precisely how **Step 2** is processed, and it can be executed in $\mathcal{O}(n)$. Recall that **Step 1** can be done in $\mathcal{O}(n \log n)$ and **Step 3** can be implemented in $\mathcal{O}(n)$ time. To the end, the problem (P_1) can be solved in $\mathcal{O}(n \log n)$ using our method consisting of **Steps 1-3**.

We illustrate our method of solving (P_1) in the following example.

Example 2.9. *Consider an instance of* (P_1) *where*

$$\begin{aligned} I^{fix} &= \{i_1^f, i_2^f, i_3^f, i_4^f\}, \\ I^1 &= \{i_1^1, i_2^1, i_3^1, i_4^1, i_5^1, i_6^1, i_7^1\}, \\ I^2 &= \{i_1^2, i_2^2, i_3^2\}. \end{aligned}$$

The items are sorted as follows:

$$i_1^1 \prec i_1^f \approx i_2^1 \prec i_3^1 \prec i_2^f \prec i_4^1 \prec i_5^1 \approx i_1^2 \prec i_3^f \prec i_6^1 \prec i_7^1 \approx i_4^f \prec i_2^2 \prec i_3^2.$$

Figure 2.6 illustrates the result obtained after executing Steps 1-3 of Algorithm 1. After executing the while loop 4-18 of the algorithm, we obtain the stacking result illustrated in Figure 2.7. Afterwards the while loop 19-27 of the algorithm results in the stacking configuration illustrated in Figure 2.8. Then, an optimal stacking configuration of $I^{fix} \to I^1 \to I^2$ is obtained by greedily storing I^2-items, as illustrated in Figure 2.9. In the figures, the arrows show the relocations of the items in the next steps.

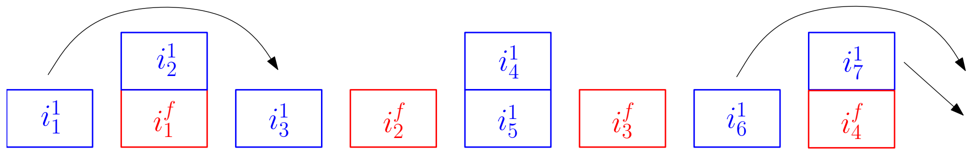


Figure 2.6: The stacking result after executing Steps 1-3 of Algorithm 1.

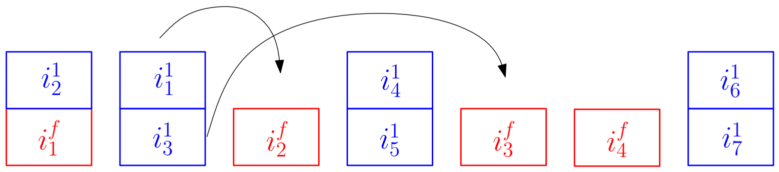


Figure 2.7: The stacking result after executing Steps 4-18 of Algorithm 1.

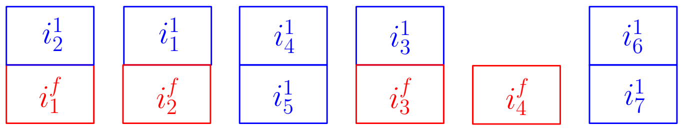


Figure 2.8: The stacking result after executing Steps 19-27 of Algorithm 1.

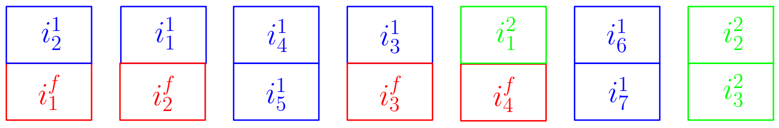


Figure 2.9: An optimal stacking configuration.

2.1.2 NP-complete cases

In this subsection we prove that the deterministic feasibility storage loading problem

$$(P_k) \qquad L, b = k \mid I^1, s_{ij} \textit{ transitive} \mid -,$$

is (strongly) NP-complete for each fixed $k \geq 3$, even for transitive stacking constraints s_{ij}. This is a generalization of a result in [31] which considered only the case $k = 3$. To prove this generalized result, we need to prove the (strong) NP-completeness of the following auxiliary problem:

EXACT COVER BY k-SETS (XkC), $k \geq 3$:

Instance: A finite set X with $|X| = kd$ for some positive integer d, together with a collection C of k-element subsets of X.

Question: Does C contain an exact cover for X, i.e., a subcollection $C' \subset C$ such that every element of X occurs in exactly one member of C'?

Proposition 2.10. *The problem* XkC *is strongly NP-complete for each fixed* $k \geq 3$.

Proof. It is well-known that the problem X3C is strongly NP-complete (see [43]). Thus, the proposition can be proven by induction on k, that is, given the fact that XkC is strongly NP-complete, we need to show that X$(k+1)$C is also strongly NP-complete.

Indeed, consider an arbitrary instance of XkC, in which X is a set with $|X| = kd$ and $C = \{c_1, \ldots, c_p\}$ is a collection of k-element subsets of X, i.e., $c_i = \{x_1^i, \ldots, x_k^i\} \in X^k$ for each $i = 1, \ldots, p$. We construct the following instance of X$(k+1)$C: let $A = \{a_1, \ldots, a_d\}$ be a set consisting of d elements such that $X \cap A = \emptyset$, and let

$$\widetilde{X} = X \cup A,$$
$$\widetilde{C} = \{c_i \cup \{a_j\} \mid i = 1, \ldots, p, j = 1, \ldots, d\}.$$

If C' is an exact cover by k-element subsets for X, say $C' = \{c_1, \ldots, c_d\}$, then the corresponding collection $\widetilde{C}' = \{c_1 \cup \{a_1\}, \ldots, c_d \cup \{a_d\}\}$ is an exact cover by $(k+1)$-element subsets for $\widetilde{X}$. Conversely, if there exists an exact cover by $(k+1)$-element subsets for $\widetilde{X}$, say $\widetilde{C}' = \{\tilde{c_1}, \ldots, \tilde{c_d}\}$, then each member $\tilde{c_i}$ of $\widetilde{C}'$ must contain exactly one element, say a_{j_i}, of A, and $\{j_1, \ldots, j_d\}$ is some permutation of $\{1, \ldots, d\}$. Clearly, the collection $C = \{c_1, \ldots, c_d\}$, in which $c_i = \tilde{c_i} \backslash \{a_{j_i}\}$ for $i = 1, \ldots, d$, is an exact cover by k-element subsets for X.

We have shown that C contains an exact cover by k-element subsets for X if and only if $\widetilde{C}$ contains an exact cover by $(k+1)$-element subsets for $\widetilde{X}$. Moreover, it is obvious that the instance $(\widetilde{X}, \widetilde{C})$ of X$(k+1)$C can be constructed in polynomial time from the instance (X, C) of XkC. Therefore the (strong) NP-completeness of X$(k+1)$C follows immediately from the (strong) NP-completeness of XkC. □

Theorem 2.11. *The feasibility problem* $L, b = k \mid I^1, s_{ij} \mid -$ *is strongly NP-complete for each fixed* $k \geq 3$*, even for transitive stacking constraints* s_{ij}.

Proof. We transform XkC to this feasibility problem. Let the set X with $|X| = kd$ and the collection $C = \{c_1, \ldots, c_p\}$ of k-element subsets of X, in which $c_i = \{x_1^i, \ldots, x_k^i\} \in X^k$ for each $i = 1, \ldots, p$, be an arbitrary instance of XkC. Corresponding to this instance of XkC, we introduce an instance of the feasibility problem with $m = d + kp$ stacks, the common height limit of these stacks is $b = k$, the number of items in I^1 is $n = km = k(d + kp)$, and the stacking constraints s_{ij} are represented by the directed graph $G = (V, A)$ which is constructed as follows.

- The nodes $V = V_1 \cup V_2$ correspond to the items in I^1. The set V_1 consists of the main nodes in which each one represents one element of X. The set V_2 consists of pk^2 auxiliary nodes $\{a_{j,l}^i \mid j = 1, \ldots, k, l = 1, \ldots, k, i = 1, \ldots, p\}$.

- Each member $c_i = \{x_1^i, \ldots, x_k^i\}$ of C defines a *substitution graph* which is a subgraph of G consisting of k main nodes $\{x_1^i, \ldots, x_k^i\}$, k^2 auxiliary nodes $\{a_{j,l}^i \mid j, l \in \{1, \ldots, k\}\}$, and collection A_i of $k^2 + k - 1$ directed arcs

$$\begin{aligned} &\{(a_{j,l}^i, a_{j,l+1}^i) \mid l \in \{1, \ldots, k-1\}, j \in \{1, \ldots, k\}\} \\ \cup\ &\{(a_{j,k-1}^i, x_j^i) \mid j \in \{1, \ldots, k\}\} \\ \cup\ &\{(a_{j,k}^i, a_{j+1,k}^i) \mid j \in \{1, \ldots, k-1\}\}. \end{aligned}$$

  The set of arcs A is the union of the collections A_i of arcs in all substitution graphs, that is

$$A = \bigcup_{i=1}^{p} A_i.$$

Figure 2.10 illustrates a substitution graph in the case $k = 4$. Note that there might be some substitution graphs having some mutual main nodes, but there is no arc between auxiliary nodes of different substitution graphs.

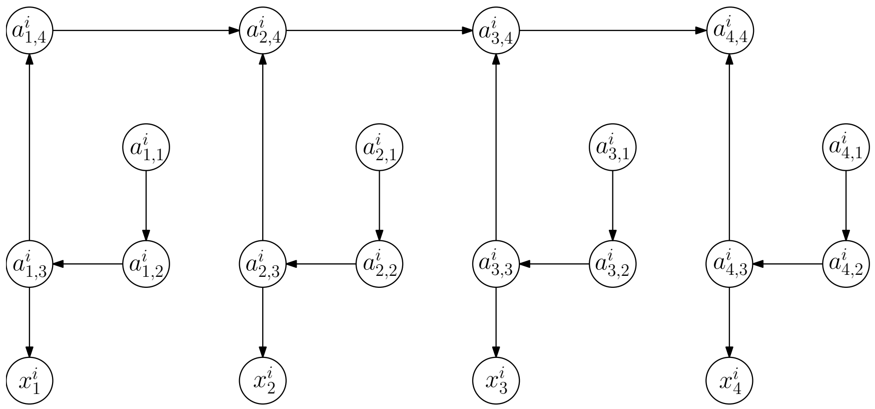


Figure 2.10: Substitution graph in the case $k = 4$.

Obviously, this instance of the feasibility problem can be constructed in polynomial time from the instance of XkC. We will show that C contains an exact cover by k-element subsets for X if and only if the stacking problem has a feasible solution. The strong NP-completeness of the feasibility problem then follows from the strong NP-completeness of the XkC problem, which is proved in Proposition 2.10.

We first show the necessity. Assume that the instance of XkC has an exact cover $C' \subset C$. The corresponding stacking problem has a feasible solution constructed as follows. Whenever $c_i = \{x_1^i, \ldots, x_k^i\}$ is in the exact cover, we stack the items represented by the nodes of the corresponding substitution graph into $k + 1$ stacks:

$$(a_{j,1}^i, \ldots, a_{j,k-1}^i, x_j^i), j = 1, \ldots, k, \text{and } (a_{1,k}^i, \ldots, a_{k,k}^i). \tag{2.4}$$

Note that the items corresponding to the main nodes of c_i are stacked in these stacks and cannot be used in any other stacks. Figure 2.11 illustrates these stacks in the case $k = 4$, in which each red path represents the way of stacking items in each stack. On the other hand, if $c_i = \{x_1^i, \ldots, x_k^i\}$ is not in the exact cover, we stack the items in the corresponding substitution graph into k stacks:

$$(a_{j,1}^i, \ldots, a_{j,k}^i), j = 1, \ldots, k. \tag{2.5}$$

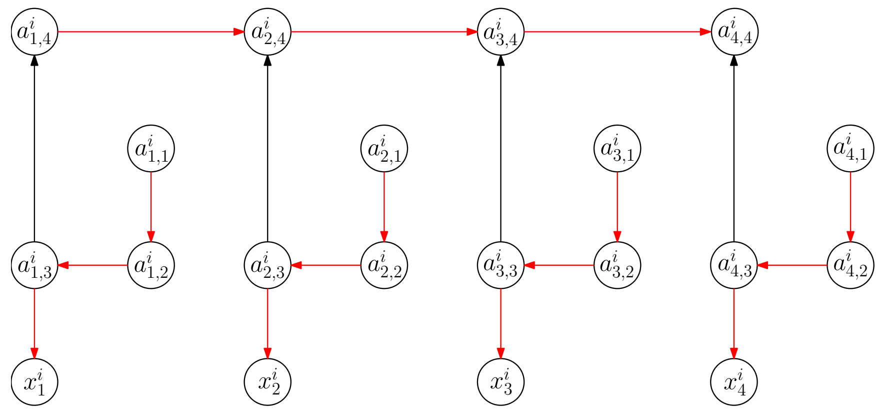


Figure 2.11: Stacks if c_i is a member of the exact cover in the case $k = 4$.

In this case the items corresponding to the main nodes of c_i are not in these stacks while the items corresponding to all auxiliary nodes are stacked. These stacks are illustrated by the red paths in Figure 2.12 for the case $k = 4$.

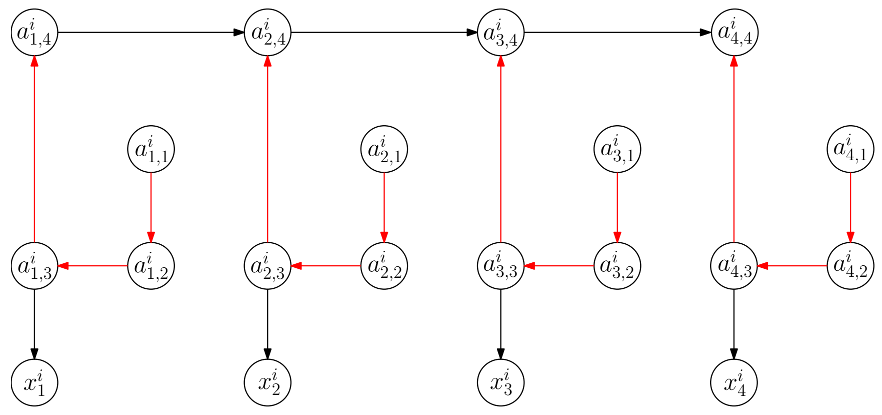


Figure 2.12: Stacks if c_i is not a member of the exact cover in the case $k = 4$.

We now show the sufficiency. Assume that there exists a feasible solution to the stacking problem. Note that by the construction of the substitution graphs, in such a feasible solution to the stacking problem, the items corresponding to the main nodes can only be stacked at the ground level. If the item x^i_1 is stacked at the ground level of some stack, say q_{i_1}, and the item $a^i_{1,k-1}$ is stacked on top of x^i_1, then it follows from the construction of the substitution graph containing $a^i_{1,k-1}$, say G_i, that the stack q_{i_1} must be $(a^i_1, \ldots, a^i_{k-1}, x^i_1)$. Then the only way to stack $a^i_{1,k}$ is to put it into the stack $(a^i_{1,k}, \ldots, a^i_{k,k})$, and the other items corresponding to the nodes of G_i must be stacked into the stacks as described in (2.4). On the other hand, if the item $a^i_{1,k-1}$ is not stacked on top of x^i_1, then it must be stacked into the stack $(a^i_{1,1}, \ldots, a^i_{1,k-1}, a^i_{1,k})$, and it is not hard to see that the items corresponding to the other auxiliary nodes of G_i must be stacked into the stacks as described in (2.5). Therefore we can construct the corresponding exact cover by k-element subsets for X by collecting $c_i \in C$ into the exact cover C' whenever the items represented by the nodes in the corresponding substitution graph are stacked into the stacks of form (2.4).

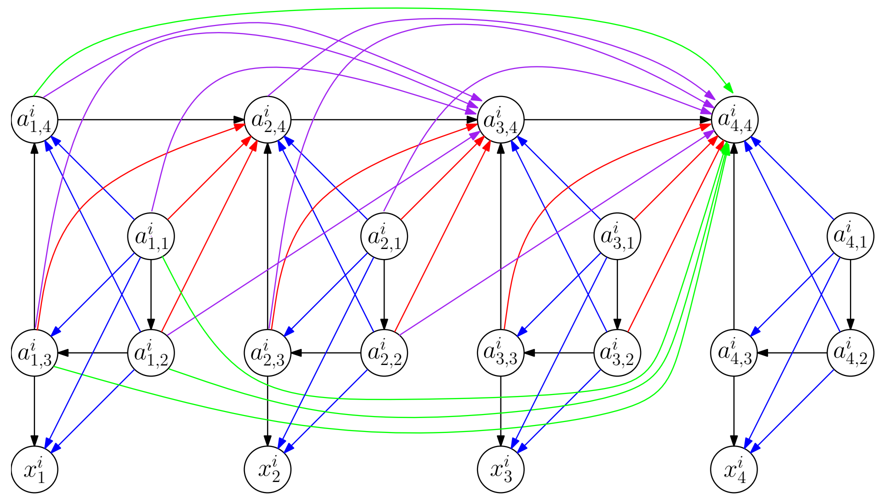


Figure 2.13: Transitive substitution graph in the case $k = 4$.

The strong NP-completeness of the feasibility problem in the case of transitive stacking constraints s_{ij} can be proven by similar arguments as above but on *transitive substitution graphs* instead of the substitution ones. A transitive substitution graph is constructed by adding to the substitution graph defined above the arcs (u, v) whenever there exists a directed path from the node u to the node v (here u, v are the nodes in the substitution graph.) Figure 2.13 illustrates a transitive substitution graph in the case $k = 4$. $\square$

2.2 Some robust storage loading problems

As introduced in the beginning of this chapter, in this section we follow the idea that actual data of items in $I^{fix} \cup I^1$ are already known, while there is some uncertainty on data of items in I^2. More precisely, we consider the following two uncertain storage loading problems. The first one, denoted by

$$(P_{\mathcal{D}}) \qquad L \mid I^{fix} \to I^1 \to \tilde{I}^2, \tilde{s}_{ij}, \mathcal{S}_{\mathcal{D}} \mid \#St,$$

corresponds to the case of discrete uncertainty set $\mathcal{S}_{\mathcal{D}} := \{(s^1_{ij}), \dots, (s^N_{ij})\}$ of possible outcomes of stacking matrix. Adapting the fact that here-and-now items $I^{fix} \cup I^1$ have certain actual data, these realizations of stacking matrices must have a common deterministic part, i.e., $s^1_{ij} = \dots = s^N_{ij}$ for $i, j \in I^{fix} \cup I^1$. The second uncertain storage loading problem, denoted by

$$(P_{\mathcal{I}}) \qquad L \mid I^{fix} \to I^1 \to \tilde{I}^2, s_{ij}(\tilde{a}), \mathcal{S}_{\mathcal{I}} \mid \#St,$$

corresponds to the case of interval uncertainty. That is, in this problem, each item i has an associated parameter a_i, in which each here-and-now item $i \in I^{fix} \cup I^1$ has deterministic value a_i, while for each wait-and-see item $i \in I^2$ its a_i-value may vary in an interval $[a^{min}_i, a^{max}_i]$. Here $\mathcal{S}_{\mathcal{I}}$ denotes the set of all possible values of vector $a = (a_1, \dots, a_n)$.

Furthermore, stacking constraints $s_{ij}(a)$ defines a total order based on values a_i's by imposing that item i is stackable on top of item j if and only if $a_i \leq a_j$.

We study the complexity of (strictly and adjustable) robust counterparts of the uncertain storage loading problems $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$. The approach of strict robustness (cf. Section 1.2.1) requires that a complete stacking solution for all items $I^1 \cup I^2$ has to be determined before the actually realized scenario of I^2 becomes known. Such an approach is required if the storage plan has to be announced before the actual data of the items in I^2 are known and the plan cannot be changed later on. In the approach of adjustable robustness (cf. Section 1.2.2), however, not all decisions have to be fixed in advance, but some can be made after the realized scenario becomes known. In the context of the stacking problems we are considering, this means that only all items in I^1 have to be assigned to locations in the storage area and that the items in I^2 can be assigned later when their actual data are known. Therefore, the locations of items in $I^{fix} \cup I^1$ are "here-and-now" variables, while the positions of items in I^2 are "wait-and-see" variables. An adjustable robust stacking solution is a stacking configuration of $I^{fix} \cup I^1$ such that for every scenario of the I^2-items we can find a feasible assignment of all I^2-items to locations in the storage area. The adjustable robust counterparts of $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$ aim to find an adjustable robust stacking solution that minimizes the number of used stacks in the worst case over all scenarios.

2.2.1 Strictly robust counterparts

In this part we aim to find strictly robust solutions to uncertain storage loading problems $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$ with $b = 2$. The key idea to solve these problems is representing the robust ability of stacking an item on other ones as an undirected graph and then solving a matching problem on the constructed graph. This is a generalization of the solution method proposed in [31] which is applied to solve the deterministic version of these problems.

Theorem 2.12. *The problem of finding a strictly robust solution to $(P_{\mathcal{D}})$ or $(P_{\mathcal{I}})$ in case $b = 2$ can be solved in polynomial time.*

Proof. We introduce a "strict stacking matrix" (s^*_{ij}) where $s^*_{ij} = 1$ if and only if item i can be stacked on top of item j in any scenario. For the uncertain problem $(P_{\mathcal{D}})$ where the data uncertainty is given in form of a finite set $\mathcal{S}_{\mathcal{D}} = \{(s^1_{ij}), \ldots, (s^N_{ij})\}$ of N stacking matrices, we can easily compute s^*_{ij} by

$$s^*_{ij} = \begin{cases} 1 & \text{if } s^k_{ij} = 1 \text{ for all } k = 1, \ldots, N, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

For the uncertain problem $(P_{\mathcal{I}})$ with stacking constraints $s_{ij}(a)$ defining a total order based on values a_i, interval uncertainties $[a^{min}_i, a^{max}_i]$ for all $i \in I^2$ and deterministic values a_i for all $i \in I^{fix} \cup I^1$, we can compute the strict stacking matrix by

$$s^*_{ij} = \begin{cases} 1 & \text{if } i, j \in I^{fix} \text{ and } i \text{ is stacked on top of } j, \\ 1 & \text{if } i \in I^1, j \in I^{fix} \cup I^1, \text{ and } a_i \leq a_j, \\ 1 & \text{if } i \in I^2, j \in I^{fix} \cup I^1, \text{ and } a^{max}_i \leq a_j, \\ 1 & \text{if } i, j \in I^2 \text{ and } a^{max}_i \leq a^{min}_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

We now construct the undirected graph $G = (V, E)$ where the nodes $V = \{1, \dots, n\}$ correspond to the items in the set I, and the edges E correspond to the pairs of items in which one item is certainly stackable on top of the other, i.e. the unordered pairs $\{i, j\}$ satisfying at least one of the following two conditions:

$$(i, j) \in (I^1 \times (I^{fix} \cup I^1)) \cup (I^2 \times I) \text{ and } s^*_{ij} = 1,$$

or

$$(j, i) \in (I^1 \times (I^{fix} \cup I^1)) \cup (I^2 \times I) \text{ and } s^*_{ji} = 1.$$

A matching in the graph G corresponds to a strictly robust stacking solution in the following sense. Each pair of matched items $\{i, j\}$ corresponding to a chosen edge in the matching is stored in a stack in a way that regards the sequence $I^{fix} \to I^1 \to \tilde{I}^2$ and i is stacked on top of j if $s^*_{ij} = 1$. Additionally, the items that are not matched in the matching have to be stored at the ground level. Thus, a matching of maximum cardinality in the graph G corresponds to a strictly robust stacking solution with the largest number of stacks containing two items, and consequently, the total number of used stacks is minimized. The set of edges E can be determined in $\mathcal{O}(Nn^2)$ in case of $(P_{\mathcal{D}})$ and $\mathcal{O}(n^2)$ in case of $(P_{\mathcal{I}})$. The maximum cardinality matching can be computed in $\mathcal{O}(n^{2.5})$ time (cf. [39]). Thus, a strictly robust solution to $(P_{\mathcal{D}})$ can be found in $\mathcal{O}(Nn^2) + \mathcal{O}(n^{2.5})$, and a strictly robust solution to $(P_{\mathcal{I}})$ can be found in $\mathcal{O}(n^{2.5})$. □

2.2.2 Adjustable robust counterparts

In this subsection, we consider the adjustable robust counterpart of $(P_{\mathcal{I}})$ where the stacking constraints define a total order based on values a_i and we have interval uncertainties $[a_i^{min}, a_i^{max}]$ for all $i \in I^2$. For convenience we denote by $(arP_{\mathcal{I}})$ the adjustable robust counterpart of $(P_{\mathcal{I}})$. We will show that an optimal solution to $(arP_{\mathcal{I}})$ can be found in the context of a so-called dominant scenario, where each item $i \in I^2$ has its maximum associated value a_i^{max}. Therefore we can derive complexity results for solving some particular cases of $(arP_{\mathcal{I}})$ from the ones for their corresponding deterministic versions.

Recall from the beginning of this section that we denote by $a = (a_1, \dots, a_n)$ the vector of a_i-values for all items $i \in I$, and $\mathcal{S}_{\mathcal{I}}$ the set of all possible scenarios of a. Given a stacking configuration x of here-and-now items $I^{fix} \cup I^1$ and a scenario a of values for all items, we denote by $\mathcal{F}(x, a)$ all feasible solutions y of stacking wait-and-see items I^2 into the storage area, and by $f_{St}(x, y)$ the number of used stacks in the combination of stacking configurations x and y. Let F_{fea} be the set of all stacking configurations x of here-and-now items $I^{fix} \cup I^1$ such that for each scenario a there exists a feasible solution y of stacking wait-and-see items I^2. Then $(arP_{\mathcal{I}})$ can be stated as follows.

$$\begin{aligned}(arP_{\mathcal{I}}) \qquad \min \quad & f^{ar}_{St}(x) := \max_{a \in \mathcal{S}_{\mathcal{I}}} \min_{y \in \mathcal{F}(x,a)} f_{St}(x, y) \\ \text{subject to} \quad & x \in F_{fea}.\end{aligned}$$

We call a scenario $a^* \in \mathcal{S}_{\mathcal{I}}$ a *dominant scenario* for $(arP_{\mathcal{I}})$ if for any stacking configuration x of here-and-now items $I^{fix} \cup I^1$ with $\mathcal{F}(x, a^*) \neq \emptyset$ we have $\mathcal{F}(x, a) \neq \emptyset$ for all $a \in \mathcal{S}_{\mathcal{I}}$. Roughly speaking, the existence of a feasible solution in the setting of the dominant scenario a^* guarantees the existence of a feasible solution in the setting of an arbitrary scenario $a \in \mathcal{S}_{\mathcal{I}}$.

In [50] a related concept of so-called worst-case scenarios was introduced as follows. A scenario $a^{wc} \in \mathcal{S}_{\mathcal{I}}$ is called a *worst-case scenario* if

$$\mathcal{F}(x, a^{wc}) \subseteq \mathcal{F}(x, a) \quad \forall a \in \mathcal{S}_{\mathcal{I}}, x \in F_{fea}.$$

In other words, a solution for the wait-and-see variables in the worst-case scenario is also a solution for the wait-and-see variables in an arbitrary scenario $a \in \mathcal{S}_{\mathcal{I}}$. It follows from the definition that a worst-case scenario is a dominant scenario, but generally the reverse does not hold.

Proposition 2.13. *Let $a^{max} \in \mathcal{S}_{\mathcal{I}}$ be the scenario in which each item $i \in I^2$ has the value a_i^{max}. Then a^{max} is a dominant scenario for $(arP_{\mathcal{I}})$.*

Proof. Let x be a feasible solution of here-and-now items $I^{fix} \cup I^1$ such that $\mathcal{F}(x, a^{max}) \neq \emptyset$, i.e., in scenario a^{max} there exists a feasible solution $y(a^{max}) \in \mathcal{F}(x, a^{max})$ of stacking wait-and-see items I^2 later on. Let Q^{max} be the set of stacks in $(x, y(a^{max}))$ containing at least one I^2-item. For each stack $q \in Q^{max}$, let I_q^2 be the set of I^2-items and i_q the topmost here-and-now item (if it exists) stored in this stack. Since $(x, y(a^{max}))$ is a feasible solution of all items, the largest a-value of I^2-items in stack q is not larger than a_{i_q}.

Now we consider an arbitrary scenario $a \in \mathcal{S}_{\mathcal{I}}$. We construct a solution $(x, y(a))$ for $I^{fix} \to I^1 \to I^2$ in the context of scenario a as follows.

- $(x, y(a))$ has the same stacking configuration x of here-and-now items $I^{fix} \cup I^1$ as in $(x, y(a^{max}))$.
- For each stack $q \in Q^{max}$, re-arrange the items in I_q^2 in non-increasing order of their a_i-values in the setting of scenario a, and store these items consecutively onto this stack (from bottom to top) ordered by non-increasing a_i-values.

By definition of a^{max}, within stack $q \in Q^{max}$, the largest a_i-value of an item $i \in I^2$ in scenario a is not larger than the value in scenario a^{max}. Therefore, the stacking configuration $(x, y(a))$ takes into account the stacking constraints based on the a_i-values. Moreover, by the above construction, $(x, y(a))$ has the same number of used stacks as $(x, y(a^{max}))$, which does not exceed the given number m of stacks in the storage area. Therefore, $(x, y(a))$ is a feasible solution for stacking all items in the context of scenario a, i.e., $y(a) \in \mathcal{F}(x, a)$. In other words, we have proven that $\mathcal{F}(x, a) \neq \emptyset$ given the assumption that $\mathcal{F}(x, a^{max}) \neq \emptyset$. Thus, a^{max} is a dominant scenario for $(arP_{\mathcal{I}})$. □

The following example illustrates the construction of $(x, y(a))$ based on $(x, y(a^{max}))$.

Example 2.14. *Consider a storage area with $m = 2$ stacks of height $b = 3$, and $n = 6$ items $I^{fix} = \{1, 2\}$, $I^1 = \{3\}$, $I^2 = \{4, 5, 6\}$. We assume $a_1 = 110, a_2 = 90, a_3 = 100$, the a_i-intervals for items $i = 4, 5, 6$ are $[80, 100], [80, 90], [75, 85]$. In the dominant scenario a^{max}, the a_i-values of the I^2-items are $a_4^{max} = 100, a_5^{max} = 90, a_6^{max} = 85$. For this scenario, a feasible stacking solution is given in Figure 2.14 (a). Figure 2.14 (b) presents a stacking solution for scenario a where the a_i-values of the I^2-items are $a_4 = 90, a_5 = 81, a_6 = 84$. The locations of the items in $I^{fix} \cup I^1$ are the same in both solutions. Furthermore, both solutions have item 4 in the first stack and items 5, 6 in the second stack. However, in the second solution, the positions of the I^2-items are changed in comparison with the former solution to adapt the order of these items in the setting of scenario a.*

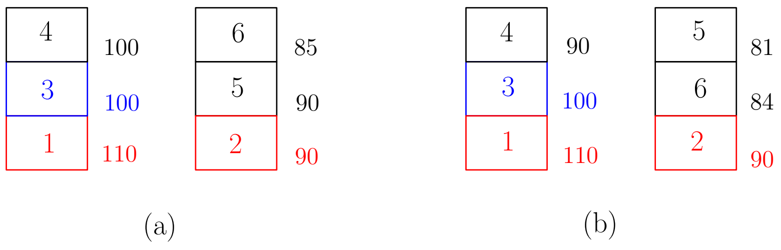


Figure 2.14: Two stacking solutions for Example 2.14.

Now we consider the optimization problem $(arP_{\mathcal{I}})$.

Theorem 2.15. *Let x^* be the stacking configuration of here-and-now items $I^{fix} \cup I^1$ in an optimal solution (x^*, y^*) in the context of the dominant scenario a^{max}. Then x^* is also an optimal solution to $(arP_{\mathcal{I}})$. Moreover, the optimal value of $(P_{\mathcal{I}})$ for the dominant scenario a^{max} equals the optimal value of $(arP_{\mathcal{I}})$.*

Proof. Since (x^*, y^*) is an optimal solution to $(P_{\mathcal{I}})$ in the context of the dominant scenario a^{max} and thanks to Proposition 2.13, for any scenario $a \in \mathcal{S}_{\mathcal{I}}$ we can construct a feasible stacking configuration $y(a)$ of all wait-and-see items I^2 such that $(x^*, y(a))$ is a feasible stacking solution of all items. Thus, $x^* \in F_{fea}$, or in other words, x^* is an adjustable robust solution to $(arP_{\mathcal{I}})$.

Furthermore, for an arbitrary $x \in F_{fea}$, it follows from the construction of $(x, y(a))$ in the proof of Proposition 2.13 that

$$\min_{y \in \mathcal{F}(x,a)} f_{St}(x,y) \leq \min_{y \in \mathcal{F}(x,a^{max})} f_{St}(x,y) \qquad \forall a \in \mathcal{S}_{\mathcal{I}}.$$

Since $a^{max} \in \mathcal{S}_{\mathcal{I}}$, we have

$$\max_{a \in \mathcal{S}_{\mathcal{I}}} \min_{y \in \mathcal{F}(x,a)} f_{St}(x,y) = \min_{y \in \mathcal{F}(x,a^{max})} f_{St}(x,y).$$

This equality holds for arbitrary $x \in F_{fea}$, therefore we obtain

$$\max_{a \in \mathcal{S}_{\mathcal{I}}} \min_{y \in \mathcal{F}(x^*,a)} f_{St}(x^*,y) = \min_{y \in \mathcal{F}(x^*,a^{max})} f_{St}(x^*,y), \tag{2.8}$$

and

$$\min_{x \in F_{fea}} \max_{a \in \mathcal{S}_{\mathcal{I}}} \min_{y \in \mathcal{F}(x,a)} f_{St}(x,y) = \min_{x \in F_{fea}} \min_{y \in \mathcal{F}(x,a^{max})} f_{St}(x,y). \tag{2.9}$$

On one hand, the left hand side of equality (2.9) is the optimal value of $(arP_{\mathcal{I}})$. On the other hand, since (x^*, y^*) is an optimal solution to $(P_{\mathcal{I}})$ in the context of the dominant scenario a^{max}, the right hand sides of equalities (2.8) and (2.9) are both equal to $f_{St}(x^*, y^*)$, which is the optimal value of $(P_{\mathcal{I}})$ for the dominant scenario a^{max}. Hence, equality (2.9) implies that the optimal value of $(P_{\mathcal{I}})$ for the dominant scenario a^{max} equals the optimal value of $(arP_{\mathcal{I}})$. Moreover, we have shown that

$$\min_{x \in F_{fea}} \max_{a \in \mathcal{S}_{\mathcal{I}}} \min_{y \in \mathcal{F}(x,a)} f_{St}(x,y) = \max_{a \in \mathcal{S}_{\mathcal{I}}} \min_{y \in \mathcal{F}(x^*,a)} f_{St}(x^*,y),$$

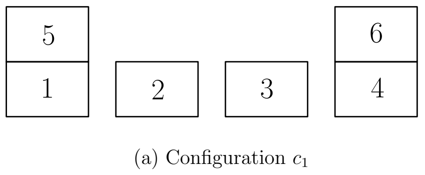
i.e., x^* is indeed an optimal solution to $(arP_{\mathcal{I}})$. □

In the following, we show that it is important for the uncertainties of the I^2-items to be given by intervals. In the case of an arbitrary total order and a finite number of stacking matrix scenarios, it may happen that no dominant scenario exists, which is shown by the following example.

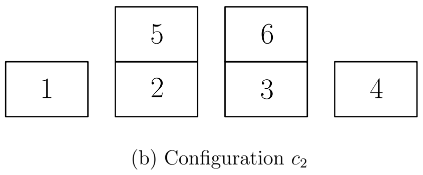
Example 2.16. *Consider a storage area with $m = 4$ stacks of height $b = 2$, and $n = 8$ items $I^1 = \{1, 2, 3, 4, 5, 6\}$, $I^2 = \{7, 8\}$. We assume that there are two possible scenarios to stack the I^2-items such that the stacking constraints define a total order on all items:*

$$\textit{Scenario } \tau_1\textit{:} \quad 6 \to 5 \to 4 \to 8 \to 3 \to 7 \to 2 \to 1.$$
$$\textit{Scenario } \tau_2\textit{:} \quad 6 \to 5 \to 8 \to 4 \to 3 \to 2 \to 7 \to 1.$$

Note that both scenarios τ_1 and τ_2 have the same order on the I^1-items. Let c_1 be the configuration of I^1-items in which items $1, 2, 3, 4$ are stored (separately) at the ground level of the four stacks, item 5 is on top of item 1, and item 6 is on top of item 4 (see Figure 2.15(a)). Let c_2 be the configuration of I^1-items in which items $1, 2, 3, 4$ are stored at the same locations as in c_1, while item 5 is on top of item 2 and item 6 is on top of item 3 (see Figure 2.15(b)).



(a) Configuration c_1



(b) Configuration c_2

Figure 2.15: Configurations for Example 2.16.

For configuration c_1 there is a feasible solution for scenario τ_1 (by putting item 7 on top of item 2 and putting item 8 on top of item 3), but no feasible solution for scenario τ_2. Similarly, for configuration c_2 there is a feasible solution for scenario τ_2 (by putting item 7 on top of item 1 and putting item 8 on top of item 4), but no feasible solution for scenario τ_1. Therefore, there is no dominant scenario in this case.

It is also important that the idea of pointing out dominant scenarios can only be applied to the case in which each item has only one associated parameter. In the following we show a counter-example when two parameters are associated with each item.

Example 2.17. *Consider a storage area with $m = 2$ stacks of height $b = 2$, and $n = 4$ items $I^{fix} = \{1\}, I^1 = \{2\}, I^2 = \{3, 4\}$. Each item i has two associated parameters denoted by a_i and d_i. Item i is stackable on top of item j if $a_i \leq a_j$ and $d_i \leq d_j$. Consider the case in which the values of the associated parameters of the items are given as follows.*

$$\begin{aligned} a_1 &= 10, & d_1 &= 4, \\ a_2 &= 9, & d_2 &= 3, \\ a_3 &\in [7, 12], & d_3 &\in [1, 6], \\ a_4 &\in [8, 11], & d_4 &\in [2, 5]. \end{aligned}$$

Note that there are exactly two possible stacking configurations of here-and-now items $I^{fix} \cup I^1$. In the first configuration, say c_1, item 2 is stacked on top of item 1. In the second configuration, say c_2, items 1 and 2 are put separately on the ground level of the two stacks.

Let τ_1 be the scenario in which $(a_3, a_4, d_3, d_4) = (7, 8, 6, 5)$, and τ_2 be the one with $(a_3, a_4, d_3, d_4) = (12, 11, 1, 2)$. Let τ^* be an arbitrary scenario of data (a_3, a_4, d_3, d_4) of wait-and-see items. Assume that $\mathcal{F}(c_1, \tau^*) \neq \emptyset$, i.e., there exists a feasible solution of stacking items 3 and 4 into the storage area where items 1 and 2 are stored according to configuration c_1. Then in this case items 3 and 4 must be stored in the same stack (since there are $m = 2$ stacks, and in configuration c_1 two here-and-now items 1 and 2 occupy one full stack). However, in scenario τ_1, the wait-and-see items cannot be stored in the same stack due to the given stacking constraints ($a_3 < a_4$ but $d_3 > d_4$). Therefore, $\mathcal{F}(c_1, \tau_1) = \emptyset$. Similarly, if $\mathcal{F}(c_2, \tau^*) \neq \emptyset$, then items 3 and 4 must be stored at the second level of the two stacks (since there are $m = 2$ stacks, and in stacking configuration c_2 each here-and-now item is stored at the ground level of one stack). However, due to the given stacking constraints, in scenario τ_2 the wait-and-see items cannot be stored at the second level of the two stacks, i.e., $\mathcal{F}(c_2, \tau_2) = \emptyset$.

We have shown that $\mathcal{F}(c_i, \tau^*) \neq \emptyset$ does not imply $\mathcal{F}(c_i, \tau) \neq \emptyset$ for all possible scenarios τ ($i = 1, 2$). Thus, τ^* cannot be a dominant scenario. Since τ^* is chosen arbitrarily, no dominant scenario exists.

We derive from Theorem 2.15 and the discussion in Section 2.1 the complexity results for the following particular cases of $(arP_{\mathcal{I}})$.

Corollary 2.18. *The adjustable robust counterpart of*

$$L \mid I^1 \to \tilde{I}^2, s_{ij}(\tilde{a}), \mathcal{S}_{\mathcal{I}} \mid \#St$$

can be solved in $\mathcal{O}(n \log n)$.

Proof. This problem is the particular case of $(arP_{\mathcal{I}})$ where $I^{fix} = \emptyset$. Thanks to Theorem 2.15, the complexity of solving this problem is determined by the complexity of solving its deterministic case where all items in I^2 are assigned with their maximum associated value a_i^{max}. Note that in the context of this dominant scenario, the stacking constraints still define a total order on the set of all items. This means the problem we are considering has the same complexity as the deterministic problem

$$L \mid I^1 \to I^2, s_{ij} \textit{ total order} \mid \#St,$$

which is solvable in $\mathcal{O}(n \log n)$ (cf. Theorem 8 [31]). □

Corollary 2.19. *The adjustable robust counterpart of*

$$L, b = 2 \mid I^{fix} \to I^1 \to \tilde{I}^2, s_{ij}(\tilde{a}), \mathcal{S}_{\mathcal{I}} \mid \#St$$

can be solved in $\mathcal{O}(n \log n)$.

Proof. This problem is the particular case of $(arP_{\mathcal{I}})$ where $b = 2$. By similar arguments as in the proof of Corollary 2.18, the problem we are considering has the same complexity as the deterministic problem

$$L, b = 2 \mid I^{fix} \to I^1 \to I^2, s_{ij} \textit{ total order} \mid \#St,$$

which is solvable in $\mathcal{O}(n \log n)$, as shown in Theorem 2.8. □

We have discussed some particular cases of strictly and adjustable robust counterparts of the uncertain storage loading problems $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$ that are polynomially solvable. The following corollary, on the other hand, shows two NP-hard cases.

Corollary 2.20. *For every fixed $b = k \geq 3$, the strictly and adjustable robust counterparts of $(P_{\mathcal{D}})$ are strongly NP-hard.*

Proof. The problems of solving strictly and adjustable robust counterparts of $(P_{\mathcal{D}})$ for fixed $b = k \geq 3$ include the deterministic problem

$$L, b = k \mid I^1, s_{ij} \mid \#St$$

as a special case ($N = 1, I^{fix} = I^2 = \emptyset$). As shown in Theorem 2.11, the feasibility version of this deterministic problem is strongly NP-complete. Therefore, the result follows straightforwardly from Theorem 2.11. □

2.3 Conclusions

In this chapter we considered some variants of the storage loading problem

$$(P) \qquad L \mid I^{fix} \to I^1 \to I^2, s_{ij} \mid \#St.$$

This problem is motivated by practical settings in which one has to store a set I^1 of items into a storage area partly filled by items in a set I^{fix}, taking into account that a set I^2 of items will arrive later. The objective is to minimize the number of used stacks regarding hard stacking constraints (s_{ij}).

In Section 2.1 we studied the complexity of some particular cases of the deterministic version of (P). We showed in Theorem 2.8 that the problem can be efficiently solved by an algorithm of complexity $\mathcal{O}(n \log n)$ if $b = 2$ and the stacking constraints s_{ij} define a total order on the list of all items. On the other hand, Theorem 2.11 establishes the boundary to NP-hardness by showing that for each fixed $b \geq 3$ the problem is NP-hard even with only one set of items I^1 and transitive stacking constraints s_{ij}.

In Section 2.2 we considered some robust versions of (P), in which the actual data of here-and-now items $I^{fix} \cup I^1$ are known exactly, while the actual data of wait-and-see items I^2 are uncertain. Discrete and interval-based uncertainty sets were concerned. We studied complexity results for strictly and adjustable robust counterparts of the uncertain problems, that are summarized in Table 2.2. The complexity results for the problems reported in Table 2.2 are derived from the ones for their deterministic versions. Open problems are marked by question marks.

Robustness	$(P_{\mathcal{D}})$		$(P_{\mathcal{I}})$		
	$b = 2$	$b \geq 3$	$b = 2$	$b \geq 3, I^{fix} = \emptyset$	$b \geq 3, I^{fix} \neq \emptyset$
Strict	$\mathcal{O}(Nn^2) + \mathcal{O}(n^{2.5})$ (Theorem 2.12)	strongly NP-hard even with $N = 1$ (Corollary 2.20)	$\mathcal{O}(n^{2.5})$ (Theorem 2.12)	?	?
Adjustable	?	strongly NP-hard even with $N = 1$ (Corollary 2.20)	$\mathcal{O}(n \log n)$ (Corollary 2.19)	$\mathcal{O}(n \log n)$ (Corollary 2.18)	?

Table 2.2: Complexity results for some robust storage loading problems.

Solving the strictly robust counterparts of the uncertain problems is equivalent to solving their deterministic versions with the "strict stacking constraints" defined by the certainly stackable relations between the items. In contrast, for solving the adjustable ones, we need to consider all scenarios of uncertain items to find out the worst-case scenario.

Therefore, at first glance, it seems that the adjustable robust counterparts are "harder" to solve than the strict ones (with the same input setting of stacking height and data uncertainty).

The above argument may be particularly true for the uncertain problem with discrete uncertainty $(P_{\mathcal{D}})$ and stacking height $b = 2$. The strictly robust counterpart of this uncertain problem can be solved in polynomial time. The key ideas are to transform the robust counterpart to an undirected graph representing strictly stackable relations between the items, then find a maximum cardinality matching of this graph. The complexity of solving the adjustable robust counterpart of this uncertain problem, however, is still an open problem. It is nothing but finding a matching of all items for each scenario of stacking matrix $(s_{ij}) \in \mathcal{S}_{\mathcal{D}}$ with the following additional restrictions. First, restricted on the here-and-now items $I^{fix} \cup I^1$, these matchings must be the same. Second, the worst-case number of stacks corresponding to such a matching must be less than a given predicted value. These additional restrictions make the adjustable case far from the simplicity of the strict case.

It is very interesting that the obtained complexity results for the robust storage loading problems with interval uncertainty $(P_{\mathcal{I}})$ disprove the argument mentioned above. In general, the "strict stacking constraints" (for solving the strictly robust counterpart of this uncertain problem) satisfy the transitivity property. Solving the adjustable robust counterpart of this problem, however, has the same complexity as solving the problem for the dominant scenario, in which all wait-and-see items have maximum associated values. The dominant scenario provides a total order on the set of all items, that offers more possibilities to stack items onto each other than transitive stacking constraints. Because of this reason, the adjustable robust counterpart of $(\mathcal{P}_{\mathcal{I}})$ can be solved more efficiently than the strict one. It is clearly true in the case $b = 2$, where the adjustable robust counterpart can be solved in $\mathcal{O}(n \log n)$ by a greedy algorithm, while the strict one can be solved in $\mathcal{O}(n^{2.5})$ by using a matching method. Another counter-example may be the case of $(\mathcal{P}_{\mathcal{I}})$ with $b \geq 3$ and no I^{fix}-item. In this case, the adjustable robust counterpart can be solved very efficiently by a greedy algorithm of complexity $\mathcal{O}(n \log n)$. Solving the strictly robust counterpart, however, is more involved. This robust counterpart can be formulated as a directed graph, in which the vertices correspond to the items, and a directed arc from i to j means that item i is certainly stackable on top of item j (taking into account the stacking sequence and the intervals constituting the uncertainty set $\mathcal{S}_{\mathcal{I}}$). Solving the problem we are considering is more or less partitioning the specific directed graph into vertex-disjoint paths of length at most b. Its complexity is still an open problem.

Chapter 3

Storage loading with stacking constraints

In Chapter 2 we considered the deterministic version and two non-deterministic versions of the storage loading problem

$$(P) \qquad L \mid I^{fix} \to I^1 \to I^2, s_{ij} \mid \#St,$$

which occurs in some practical contexts where one has to store a set of items into a partly filled storage area, taking into account that some items will arrive later. Again, I^{fix} is the set of items that are already stored in the storage area, I^1 denotes the set of items to be stored now, and I^2 stands for the set of items coming later. The objective is to minimize the number of used stacks to have more flexibility for the remaining stacks to store items arriving later on, while taking into account hard stacking constraints s_{ij} during loading process.

For non-deterministic versions of (P), we follow the idea that the practitioner knows exactly the actual data of here-and-now items $I^{fix} \cup I^1$, while the actual data of wait-and-see items I^2 are uncertain. This leads to the uncertain storage loading problem with discrete uncertainty

$$(P_{\mathcal{D}}) \qquad L \mid I^{fix} \to I^1 \to \tilde{I}^2, \tilde{s}_{ij}, \mathcal{S}_{\mathcal{D}} \mid \#St,$$

and the one with interval uncertainty

$$(P_{\mathcal{I}}) \qquad L \mid I^{fix} \to I^1 \to \tilde{I}^2, s_{ij}(\tilde{a}), \mathcal{S}_{\mathcal{I}} \mid \#St,$$

as introduced in Section 2.2 of Chapter 2.

Complexity results for some particular cases of the deterministic problem (P), as well as strictly and adjustable robust counterparts of uncertain problems $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$, were studied in Chapter 2. This chapter, which is mainly based on our work in [62], presents different MIP formulations for the deterministic problem (Section 3.1) and for the robust counterparts of the uncertain problems (Sections 3.2 and 3.3). In Section 3.4 we present computational results for randomly generated instances with up to 480 items. The results show that instances of this size can be solved in reasonable time and that including robustness improves solutions where uncertainty is not taken into account. Section 3.5 closes this chapter with some conclusions.

3.1 MIP formulations for the deterministic problem

In this section, we propose three different (mixed-) integer linear programming formulations for the deterministic problem (P). While the first two formulations can deal with arbitrary stacking constraints s_{ij}, the third one is only valid for transitive constraints. Recall from Section 1.1.1 that we denote the set of all items by $I = \{1, \dots, n\}$, the set of stacks by $Q = \{1, \dots, m\}$, and the set of levels in one stack by $L = \{1, \dots, b\}$.

3.1.1 Three-index formulation

In this subsection, we describe the first IP formulation which is based on three-indexed binary variables $x_{i,q,l}$ for $i \in I, q \in Q, l \in L$, where

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

Let

$$F = \{(i,q,l) \in I^{fix} \times Q \times L \mid \begin{array}{l} \text{item } i \text{ is stored in stack } q \text{ at level } l \\ \text{in the given storage configuration}\} \end{array}$$

be the set of triples corresponding to locations of fixed items.

Then problem (P) can be formulated as the following IP:

$$(Ind-P) \qquad \min \sum_{(i,q) \in I \times Q} x_{i,q,1} \tag{3.1}$$

$$\text{s.t.} \qquad x_{i,q,l} = 1 \qquad \forall (i,q,l) \in F \tag{3.2}$$

$$\sum_{(q,l) \in Q \times L} x_{i,q,l} = 1 \qquad \forall i \in I^1 \cup I^2 \tag{3.3}$$

$$\sum_{i \in I} x_{i,q,l} \le 1 \qquad \forall (q,l) \in Q \times L \tag{3.4}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x_{j,q,l-1} \ge x_{i,q,l} \qquad \forall (i,q,l) \in I \times Q \times (L \setminus \{1\}) \tag{3.5}$$

$$x_{i,q,l} \in \{0,1\} \qquad \forall (i,q,l) \in I \times Q \times L \tag{3.6}$$

The objective function (3.1) minimizes the number of items stored at level 1 (the ground level), which equals the number of used stacks. Constraints (3.2) ensure that each item in the set I^{fix} is fixed at its given location in the storage area, while constraints (3.3) guarantee that all items in $I^1 \cup I^2$ are stored. Constraints (3.4) ensure that at most one item is stored at each level of each stack. Constraints (3.5) ensure that the stacking constraints s_{ij} (including the sequence $I^{fix} \to I^1 \to I^2$) are taken into account and that no item is put at a location where no item is stored below. Finally, the domains of the variables are defined in (3.6). This formulation contains $\mathcal{O}(nmb)$ variables and $\mathcal{O}(nmb)$ constraints.

3.1.2 Network flow formulation

In this subsection, we show that (P) can also be modeled as a network flow problem with some additional constraints, and propose a MIP formulation for it.

Let $\underline{I}^{fix}$ be the set of items in I^{fix} stored at the ground level, and $\overline{I}^{fix}$ the set of the topmost I^{fix}-items in the stacks. We start with a directed graph $G = (V, A)$ representing the stacking constraints s_{ij}. The nodes in V correspond to the items in I, while an arc (i, j) in A connects item i with item j if i is stackable on top of j (i.e., $s_{ij} = 1$). Since we assume that the stacking sequence $I^{fix} \to I^1 \to I^2$ is integrated into the stacking matrix (s_{ij}), the condition $s_{ij} = 1$ also implies that item j does not arrive later than i. More precisely, $A = A_0 \cup A_1$ with

$$A_0 = \{(i, j) \in I^{fix} \times I^{fix} \mid \text{item } i \text{ is stacked on top of item } j\},$$
$$A_1 = \{(i, j) \in (I^1 \times (I^{fix} \cup I^1)) \cup (I^2 \times I) \mid s_{ij} = 1\}.$$

By the construction of G, each feasible solution to (P) corresponds to a partition of G into at most m pairwise node-disjoint paths such that each path contains at most b nodes. The paths in such a partition of G can be viewed as pairwise node-disjoint flows, each flow sending one unit through one path. This motivates embedding $G = (V, A)$ in a network $G' = (V', A')$ which is constructed as follows.

- $V' = V \cup \{u_0, u_1, \ldots, u_m\} \cup \{v_0, v_1, \ldots, v_m\}$, in which u_0 is the source node, v_0 is the sink node, and u_k, v_k for $k = 1, \ldots, m$ are auxiliary nodes corresponding to the top and the bottom of the m stacks.
- $A' = A \cup A_2 \cup A_3 \cup A_4$, where

  $$A_2 = \{(u_0, u_k) \mid k = 1, \ldots, m\} \cup \{(v_k, v_0) \mid k = 1, \ldots, m\} \cup \{(u_k, v_k) \mid k = 1, \ldots, m\}$$

  contains m directed arcs from the source node u_0 to the auxiliary nodes $u_1, \ldots, u_m$ as well as m directed arcs from the auxiliary nodes $v_1, \ldots, v_m$ to the sink node v_0, and m directed arcs $(u_1, v_1), \ldots, (u_m, v_m)$. Furthermore,

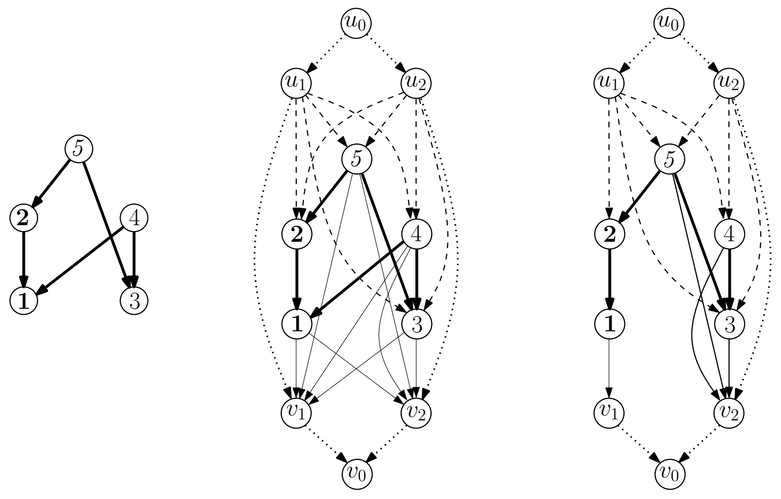
  $$A_3 = \{(u_k, i) \mid k = 1, \ldots, m, i \in \overline{I}^{fix} \cup I^1 \cup I^2\}$$

  contains directed arcs from the auxiliary nodes $u_1, \ldots, u_m$ to the nodes corresponding to items in $I^1 \cup I^2$ and the topmost I^{fix}-items in the stacks, and

  $$A_4 = \{(i, v_k) \mid k = 1, \ldots, m, i \in \underline{I}^{fix} \cup I^1 \cup I^2\}$$

  contains directed arcs from the nodes corresponding to items in $I^1 \cup I^2$ and the I^{fix}-items stored at the ground level of the stacks to the auxiliary nodes $v_1, \ldots, v_m$.

Example 3.1. *Consider a small storage area with $m = 2$ stacks of height $b = 3$ and $n = 5$ items in the sets $I^{fix} = \{1, 2\}, I^1 = \{3, 4\}, I^2 = \{5\}$ where item 2 is stacked on top of item 1 in the first stack. Furthermore, we assume that item 4 can be stacked on top of items 1 and 3, and item 5 is stackable onto items 2 and 3, i.e., $s_{41} = s_{43} = s_{52} = s_{53} = 1$. On the other hand, we assume that item 3 cannot be stacked on items 1, 2, 4, item 4 cannot be stacked on item 2, and item 5 cannot be stacked on items 1 and 4, i.e., $s_{31} = s_{32} = s_{34} = s_{42} = s_{51} = s_{54} = 0$. Due to the stacking sequence $I^{fix} \to I^1 \to I^2$, we also have $s_{13} = s_{14} = s_{15} = s_{23} = s_{24} = s_{25} = s_{35} = s_{45} = 0$. The corresponding graph G is shown in Figure 3.1(a) and the network G' constructed from G in (b). The arcs of graph G are represented with thick lines, while the arcs in A_2, A_3, and A_4 are represented with dotted lines, dashed lines, and continuous thin lines, respectively. The path partition $\{5 \to 2 \to 1, 4 \to 3\}$ of G leads to the solution where items 5, 1, 2 are stacked in this order*



(a) Graph G (b) Network G' (c) Reduced network

Figure 3.1: Networks G and G' in Example 3.1.

in the first stack and item 4 is on top of item 3 in the second stack. This configuration corresponds to the flows $u_0 \to u_1 \to 5 \to 2 \to 1 \to v_1 \to v_0, u_0 \to u_2 \to 4 \to 3 \to v_2 \to v_0$ in G'.

Since each I^{fix}-item belongs to exactly one stack, we can remove some unnecessary arcs in the construction of the network as follows. Let Q' be the set of stacks containing at least one I^{fix}-item. For each stack $q \in Q'$, let $\underline{i}^q$ be the I^{fix}-item stored at the ground level of the stack. Then we only introduce the directed arc $(\underline{i}^q, v_q)$ and remove all other arcs going into v_q and going out of $\underline{i}^q$. Similarly, let $\bar{i}^q$ be the topmost I^{fix}-item in $q \in Q'$. Then, we introduce the directed arc $(u_q, \bar{i}^q)$ and remove all other arcs of the form $(u_q, \bar{i}^{q'})$ or $(u_{q'}, \bar{i}^q)$ with $q' \in Q \backslash \{q\}$. Furthermore, if i and j are two I^{fix}-items where i is stacked on top of j, then we only keep the arc (i, j) and remove all other arcs going into j and going out of i. Figure 3.1(c) shows the reduced network for Example 3.1.

To model the network flow problem as a MIP, we introduce binary variables x_{ij} corresponding to arcs $(i, j) \in A'$, where $x_{ij} = 1$ if the arc (i, j) carries some flow (i.e., i is stacked on top of j), otherwise $x_{ij} = 0$. Note that if an arc (u_k, v_k) carries flow, then the corresponding stack remains empty. To ensure that each stack contains at most b items and to avoid cycles in the flow, we introduce auxiliary variables c_i for all nodes $i \in V' \backslash \{u_0, v_0\}$, where c_i represents the number of arcs on the path from u_0 to node i carrying the flow through i. Then, the problem can be formulated as follows.

$$(Flow - P) \qquad \min \sum_{(i,j)\in A_4} x_{ij} \tag{3.7}$$

$$\text{s.t.} \qquad x_{ij} = 1 \qquad \forall\ (i,j) \in A_0 \tag{3.8}$$

$$\sum_{j:(i,j)\in A'} x_{ij} = 1 \qquad \forall i \in V' \backslash \{u_0, v_0\} \tag{3.9}$$

$$\sum_{j:(j,i)\in A'} x_{ji} = 1 \qquad \forall i \in V' \backslash \{u_0, v_0\} \tag{3.10}$$

$$c_i = 1 \qquad \forall i \in \{u_1, \ldots, u_m\} \tag{3.11}$$

$$c_j - c_i + b(1 - x_{ij}) \geq 1 \qquad \forall (i,j) \in A', j \neq v_0 \qquad (3.12)$$
$$c_i \leq b + 2 \qquad \forall i \in \{v_1, \ldots, v_m\} \qquad (3.13)$$
$$c_i \geq 0 \qquad \forall i \in V' \backslash \{u_0, v_0\} \qquad (3.14)$$
$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A' \qquad (3.15)$$

According to (3.7) the total flow on the arcs from A_4, which corresponds to the number of used stacks is minimized. Constraints (3.8) ensure that the given stacking configuration of the I^{fix}-items is respected by stating that the arcs in A_0 must always carry some flow. Constraints (3.9) and (3.10) model that each node (except the source node u_0 and the sink node v_0) transfers exactly one unit of flow (i.e., each item belongs to exactly one stack). Due to constraints (3.11) the values of c_i for the top elements $u_1, \ldots, u_m$ are set to one. In order to compute the c_i-values, we have to ensure that $c_j \geq c_i + 1$ if $x_{ij} = 1$, which is modeled by (3.12). These constraints also forbid cycles in the flow. Note that due to constraints (3.12) c_i may be chosen larger than necessary, i.e., these values are only upper bounds for the actual number of arcs on the path from u_0 to i carrying the flow through i. However, as long as (3.13) (guaranteeing that each stack contains at most b items) is satisfied, this does not lead to any problems. Note that no integrality constraints are required for the variables c_i, i.e., they may be introduced as continuous variables in (3.14). This formulation contains $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^2)$ constraints.

3.1.3 Bin packing formulation

In this subsection, we consider an IP formulation for problem (P) where contrary to the two previous formulations it is necessary that the stacking constraints s_{ij} are transitive. As discussed in Section 1.1.3, the stacking constraints are often transitive in practice, especially, if they are based on several $\leq$-relations. We use a formulation similar to bin packing where each item is only assigned to a stack (similar to a bin) but not to a specific level in this stack. With some appropriate constraints and due to the transitivity of the stacking constraints, we can ensure that if the IP has a feasible solution, then a feasible assignment of all items is also always possible.

We call a subset $J \subseteq I$ a S-total set if the stacking matrix S defines a total relation on the items in J (i.e., for all $i, j \in J$ we have $s_{ij} = 1$ or $s_{ji} = 1$). If J is a S-total set and the stacking constraints s_{ij} are transitive, then the stacking matrix S defines a total order on the items of J. Consequently, if $|J| \leq b$, we can feasibly store all items from J in a stack where the position (level) of each item can be easily determined by sorting the items according to the total order on J. This implies that if there is a partition of I into at most m disjoint S-total subsets of cardinality at most b, then this partition corresponds to a feasible solution of the storage problem (each subset corresponds to a set of items placed into the same stack). Conversely, it is obvious that each feasible solution of the storage problem corresponds to a partition of I into at most m disjoint S-total subsets of cardinality at most b. Therefore, finding a feasible solution of the storage loading problem is equivalent to finding such a partition of I.

Based on this idea, we introduce binary variables x_{iq} for $i \in I, q \in Q$ with

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

To calculate the number of used stacks, we introduce additional binary variables z_q for

$q \in Q$ with

$$z_q = \begin{cases} 1 & \text{if at least one item is stored in stack } q, \\ 0 & \text{otherwise.} \end{cases}$$

Let

$$F_{bin} = \{(i,q) \in I^{fix} \times Q \mid \begin{array}{l} \text{item } i \in I^{fix} \text{ is stored in stack } q \\ \text{in the given storage configuration}\} \end{array}$$

and

$$F_{bin}^{top} = \{(i,q) \in I^{fix} \times Q \mid \text{item } i \text{ is the topmost } I^{fix}\text{-item stored in stack } q\}.$$

Furthermore, we again denote by

$$Q' = \{q \in Q \mid \exists\, i \in I^{fix} : (i,q) \in F_{bin}\}$$

the set of stacks that contain at least one I^{fix}-item. For each $q \in Q'$ let b_q be the number of I^{fix}-items in stack q. As discussed above, the set of items in each stack of a feasible stacking solution must be S-total, i.e., for any two items $i, j \in I$ we must have $s_{ij} = 1$ or $s_{ji} = 1$. This is equivalent to the condition that two items $i, j \in I$ cannot be stored together in the same stack if and only if $s_{ij} = 0$ and $s_{ji} = 0$. Since the stacking sequence $I^{fix} \to I^1 \to I^2$ is already taken into account in the s_{ij}-matrix, and due to the transitivity of the stacking constraints s_{ij}, if an item $i \in I^1 \cup I^2$ is stackable on top of the topmost I^{fix}-item of a stack, then i is also stackable on top of the other I^{fix}-items in the same stack. However, if an item $i \in I^1 \cup I^2$ cannot be stored together with the topmost I^{fix}-item of a stack, then i cannot be stored in that stack. Therefore, any feasible solution x must satisfy the inequalities

$$\sum_{\substack{j \in I^1 \cup I^2 \setminus \{i\} \\ s_{ij} = s_{ji} = 0}} x_{jq} \leq b(1 - x_{iq}) \qquad \forall (i,q) \in (I^1 \cup I^2) \times (Q \setminus Q'),$$

$$\sum_{\substack{j \in I^1 \cup I^2 \setminus \{i\} \\ s_{ij} = s_{ji} = 0}} x_{jq} \leq (b - b_q)(1 - x_{iq}) \qquad \forall (i,q) \in F_{bin}^{top} \cup (I^1 \cup I^2) \times Q'.$$

Based on the above considerations we obtain the following IP formulation:

$$(Bin - P) \qquad \min \sum_{q \in Q} z_q \tag{3.16}$$

$$\text{s.t.} \quad x_{iq} = 1 \qquad \forall (i,q) \in F_{bin} \tag{3.17}$$

$$\sum_{i \in I} x_{iq} \leq b z_q \qquad \forall q \in Q \tag{3.18}$$

$$\sum_{q \in Q} x_{iq} = 1 \qquad \forall i \in I \tag{3.19}$$

$$\sum_{\substack{j \in I^1 \cup I^2 \setminus \{i\} \\ s_{ij} = s_{ji} = 0}} x_{jq} \leq b(1 - x_{iq}) \qquad \forall (i,q) \in (I^1 \cup I^2) \times (Q \setminus Q') \tag{3.20}$$

$$\sum_{\substack{j \in I^1 \cup I^2 \setminus \{i\} \\ s_{ij} = s_{ji} = 0}} x_{jq} \leq (b - b_q)(1 - x_{iq}) \quad \forall (i,q) \in F_{bin}^{top} \cup (I^1 \cup I^2) \times Q' \tag{3.21}$$

$$x_{iq} \in \{0,1\} \qquad \forall i \in I, q \in Q \tag{3.22}$$

$$z_q \in \{0,1\} \qquad \forall q \in Q \tag{3.23}$$

According to (3.16) the number of used stacks is minimized. Constraints (3.17) ensure that the items of I^{fix} are stored in the correct stacks according to the given configuration. Constraints (3.18) guarantee that at most b items are assigned to each stack. Additionally, they enforce that the variable z_q is set to one if at least one item is assigned to stack q. Due to equalities (3.19) each item is stored in exactly one stack. As discussed above, constraints (3.20) and (3.21) take into account that only compatible items are assigned to the same stack. This formulation contains $\mathcal{O}(nm)$ variables and $\mathcal{O}(nm)$ constraints.

Furthermore, we can add the following nm redundant constraints to $(Bin - P)$

$$x_{iq} \leq z_q \qquad \forall\ i \in I, q \in Q \tag{3.24}$$

to represent the requirement that the variable z_q is set to one if some item is assigned to stack q. Then we obtain the extended variant

$$(BinE - P) \qquad \left\{ \min \sum_{q \in Q} z_q : (3.17) - (3.24) \right\}.$$

The IP formulation $(BinE - P)$ uses the same number of variables as $(Bin - P)$ and also uses $\mathcal{O}(nm)$ constraints. However, it is well known that $(BinE - P)$ in general has a better LP relaxation than $(Bin - P)$, since the feasible set of the LP relaxation of $(BinE - P)$ is a subset of the one corresponding to $(Bin - P)$.

3.2 MIP formulations for the strictly robust problems

In this section, we consider the strictly robust counterparts of the non-deterministic problems $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$. In the setting of these problems, a complete stacking solution for all items $I^1 \cup I^2$ has to be determined before the actually realized scenario of I^2 becomes known. Such an approach is required if the storage plan has to be announced before the actual data of the items in I^2 are known and the plan cannot be changed later on. Following the approach of strict robustness introduced in Section 1.2.1, we focus on strictly robust solutions where the worst case over all scenarios is optimized.

In this context, a *strictly robust stacking solution* is a stacking configuration of all items which satisfies the stacking constraints in all realizations of the uncertain data. The strictly robust counterparts of $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$ aim to find a strictly robust solution that minimizes the number of used stacks in the worst case over all scenarios.

The main idea when dealing with the strictly robust counterparts of these uncertain problems is to introduce a "strict stacking matrix" (s^*_{ij}) where $s^*_{ij} = 1$ if and only if item i can be stacked on top of item j in any scenario.

For the uncertain problem $(P_{\mathcal{D}})$ where the data uncertainty is given in the form of a finite set $\mathcal{S}_{\mathcal{D}} = \{(s^1_{ij}), \ldots, (s^N_{ij})\}$ of N stacking matrices, we can easily compute s^*_{ij} by (2.6), i.e.,

$$s^*_{ij} = \begin{cases} 1 & \text{if } s^k_{ij} = 1 \text{ for all } k = 1, \ldots, N, \\ 0 & \text{otherwise.} \end{cases}$$

We will denote by $(srP_{\mathcal{D}})$ the strictly robust counterpart of $(P_{\mathcal{D}})$. Note that if all scenarios $s^1_{ij}, \ldots, s^N_{ij}$ are transitive, then s^*_{ij} also is, and we can apply the bin packing formulation in this case. Replacing the stacking matrix (s_{ij}) in formulations $(Ind-P), (Flow-P), (Bin-P)$, $(BinE-P)$ by the strict stacking matrix (s^*_{ij}) from (2.6), we obtain MIP formulations for $(srP_{\mathcal{D}})$ denoted by $(Ind-srP_{\mathcal{D}}), (Flow-srP_{\mathcal{D}}), (Bin-srP_{\mathcal{D}})$, $(BinE-srP_{\mathcal{D}})$, respectively. In all these formulations no additional variables or constraints are needed.

For the uncertain problem $(P_{\mathcal{I}})$ with stacking constraints s_{ij} which define a total order based on values a_i, interval uncertainties $[a_i^{min}, a_i^{max}]$ for all $i \in I^2$ and deterministic values a_i for all $i \in I^{fix} \cup I^1$, we can compute the strict stacking matrix by (2.7), i.e.,

$$s^*_{ij} = \begin{cases} 1 & \text{if } i,j \in I^{fix} \text{ and } i \text{ is stacked on top of } j, \\ 1 & \text{if } i \in I^1, j \in I^{fix} \cup I^1, \text{ and } a_i \leq a_j, \\ 1 & \text{if } i \in I^2, j \in I^{fix} \cup I^1, \text{ and } a_i^{max} \leq a_j, \\ 1 & \text{if } i,j \in I^2 \text{ and } a_i^{max} \leq a_j^{min}, \\ 0 & \text{otherwise.} \end{cases}$$

We will denote the strictly robust counterpart of the problem for interval uncertainties by $(srP_{\mathcal{I}})$. Replacing the stacking matrix (s_{ij}) in formulations $(Ind-P), (Flow-P), (Bin-P), (BinE-P)$ by the strict stacking matrix (s^*_{ij}) from (2.7), we obtain MIP formulations for $(srP_{\mathcal{I}})$ denoted by $(Ind-srP_{\mathcal{I}}), (Flow-srP_{\mathcal{I}}), (Bin-srP_{\mathcal{I}}), (BinE-srP_{\mathcal{I}})$, respectively.

3.3 MIP formulations for the adjustable robust problems

In this section, we consider the adjustable robust counterparts of the non-deterministic problems $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$. Following the ideas for adjustable robustness introduced in Section 1.2.2, not all decisions have to be fixed in advance, but some can be made after the realized scenario becomes known. In the context of our stacking problems, this means that only the items in I^1 have to be assigned to locations in the storage area and that the items in I^2 can be assigned later when their actual data are known. Therefore, the locations of items in $I^{fix} \cup I^1$ are "here-and-now" variables, while the positions of items in I^2 are "wait-and-see" variables. An *adjustable robust stacking solution* is a stacking configuration of $I^{fix} \cup I^1$ such that for every scenario of the I^2-items we can find a feasible assignment of all I^2-items to locations in the storage area. The adjustable robust counterparts of $(P_{\mathcal{D}})$ and $(P_{\mathcal{I}})$ aim to find an adjustable robust stacking solution that minimizes the number of used stacks in the worst case over all scenarios.

3.3.1 Discrete uncertainty

In this subsection, we assume that we have a finite set of scenarios described by a set $\mathcal{S}_{\mathcal{D}} = \{(s^1_{ij}), \ldots, (s^N_{ij})\}$ of possible outcomes of stacking matrix. Let $\mathcal{N} = \{1, \ldots, N\}$ and $(arP_{\mathcal{D}})$ be the adjustable robust counterpart of $(P_{\mathcal{D}})$.

Three-index formulation

The IP formulation $(Ind-P)$ proposed in Section 3.1.1 uses binary variables $x_{i,q,l}$ $(i \in I, q \in Q, l \in L)$ to represent an assignment of all items in I to feasible locations in the storage area. For adjustable robustness, we have to distinguish between here-and-now decisions for items from $I^{fix} \cup I^1$ and wait-and-see decisions for items from I^2 in all scenarios. Hence, we split the variables into two sets. The first set consists of binary here-and-now variables $x_{i,q,l}$ with $(i,q,l) \in (I^{fix} \cup I^1) \times Q \times L$. These variables determine the assignments of items in $I^{fix} \cup I^1$ to locations where

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

The second set consists of binary wait-and-see variables $y^k_{i,q,l}$ for $(i,q,l) \in I^2 \times Q \times L$ and all $k \in \mathcal{N}$. These variables determine the positions of I^2-items in a solution for scenario k where

$$y^k_{i,q,l} = \begin{cases} 1 & \text{if item } i \in I^2 \text{ is stored in stack } q \text{ at level } l \text{ in the solution for scenario } k, \\ 0 & \text{otherwise.} \end{cases}$$

In order to calculate the worst-case number of used stacks over all scenarios we introduce an additional auxiliary variable $v \geq 0$. Let $X = (I^{fix} \cup I^1) \times Q \times L$, $X' = I^1 \times Q \times (L \backslash \{1\})$, $Y = I^2 \times Q \times L$, $Y' = I^2 \times Q \times (L \backslash \{1\})$. From $(Ind - P)$ we then derive the following MIP formulation for $(arP_{\mathcal{D}})$.

$$
\begin{aligned}
&(Ind - arP_{\mathcal{D}}) && \min \quad v && && (3.25)\\
&\text{s.t.} && \sum_{(i,q,1) \in X} x_{i,q,1} + \sum_{(j,q,1) \in Y} y^k_{j,q,1} \leq v && \forall k \in \mathcal{N} && (3.26)\\
& && x_{i,q,l} = 1 && \forall (i,q,l) \in F && (3.27)\\
& && \sum_{(q,l) \in Q \times L} x_{i,q,l} = 1 && \forall i \in I^1 && (3.28)\\
& && \sum_{(q,l) \in Q \times L} y^k_{i,q,l} = 1 && \forall i \in I^2, k \in \mathcal{N} && (3.29)\\
& && \sum_{i \in I^{fix} \cup I^1} x_{i,q,l} + \sum_{j \in I^2} y^k_{j,q,l} \leq 1 && \forall (q,l) \in Q \times L, k \in \mathcal{N} && (3.30)\\
& && \sum_{j \in I^{fix} \cup I^1} s_{ij} x_{j,q,l-1} \geq x_{i,q,l} && \forall (i,q,l) \in X' && (3.31)\\
& && \sum_{j \in I^{fix} \cup I^1} s^k_{ij} x_{j,q,l-1} + \sum_{j \in I^2} s^k_{ij} y^k_{j,q,l-1} \geq y^k_{i,q,l} && \forall (i,q,l) \in Y', k \in \mathcal{N} && (3.32)\\
& && x_{i,q,l} \in \{0,1\} && \forall (i,q,l) \in X && (3.33)\\
& && y^k_{i,q,l} \in \{0,1\} && \forall (i,q,l) \in Y, k \in \mathcal{N} && (3.34)\\
& && v \geq 0 && && (3.35)
\end{aligned}
$$

The left hand side of (3.26) is equal to the number of used stacks in the solution for scenario $k \in \mathcal{N}$, therefore by (3.25) the worst-case number of used stacks over all scenarios in $\mathcal{N}$ is minimized. Constraints (3.27) ensure that each item in the set I^{fix} is stored at its given position in the storage area. Constraints (3.28) guarantee that all items from I^1 are stored, while constraints (3.29) make sure that in each scenario $k \in \mathcal{N}$ all items from I^2 are stored. Constraints (3.30) mean that for each scenario $k \in \mathcal{N}$ at most one item can be stored at each level of each stack. Constraints (3.31) and (3.32) together guarantee that in each scenario $k \in \mathcal{N}$ the corresponding stacking constraints s^k_{ij} (including the stacking sequence $I^{fix} \to I^1 \to I^2$) are taken into account and that no item is put at a location where no item is stored below. This formulation contains $\mathcal{O}(nmbN)$ variables and $\mathcal{O}(nmbN)$ constraints.

Network flow formulation

In the following, we adapt the network flow formulation proposed in Section 3.1.2 to model the adjustable robust counterpart $(arP_{\mathcal{D}})$. Based on the idea of constructing the reduced network $G' = (V', A')$, we construct a network $G'(s) = (V'(s), A'(s))$ corresponding to scenario $s \in \mathcal{S}_{\mathcal{D}}$ by setting

- $V'(s) = V'$;
- $A'(s) = A_0 \cup A_1^h \cup A_1^w(s) \cup A_2^h \cup A_2^w \cup A_3 \cup A_4^h \cup A_4^w$, where

$$\begin{aligned} A_1^h &= \{(i,j) \in I^1 \times (I^{fix} \cup I^1) \mid s_{ij} = 1\}, \\ A_1^w(s) &= \{(i,j) \in I^2 \times I \mid s_{ij} = 1\}, \\ A_2^h &= \{(u_0, u_k) \mid k = 1, \dots, m\} \cup \{(v_k, v_0) \mid k = 1, \dots, m\}, \\ A_2^w &= \{(u_k, v_k) \in A_2 \mid k = 1, \dots, m\}, \\ A_4^h &= \{(i, v_k) \in A_4 \mid i \in \underline{I}^{fix} \cup I^1, k = 1, \dots, m\}, \\ A_4^w &= \{(i, v_k) \in A_4 \mid i \in I^2, k = 1, \dots, m\}, \end{aligned}$$

and A_0, A_2, A_3, A_4 are defined as in the construction of G' in Section 3.1.2.

Note that

$$A_1 = A_1^h \cup A_1^w(s),\ A_2 = A_2^h \cup A_2^w,\ A_4 = A_4^h \cup A_4^w.$$

The sets A_0, A_1^h, A_2^h, A_4^h correspond to the here-and-now decisions, while the sets $A_1^w(s)$, A_2^w, A_4^w correspond to the wait-and-see decisions. By construction of the network in Section 3.1.2, if an arc $(u_k, i) \in A_3$ has a positive flow, then i is the topmost item of stack k. However, in the stacking solutions corresponding to different data scenarios, the topmost items may change. Therefore, all elements in A_3 correspond to wait-and-see decisions.

For modeling $(arP_{\mathcal{D}})$ we use the following sets of variables. Let $A^h = A_0 \cup A_1^h \cup A_2^h \cup A_4^h$ and define binary here-and-now variables x_{ij} for $(i,j) \in A^h$ with

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ carries to some flow,} \\ 0 & \text{otherwise.} \end{cases}$$

Let $A^w(s^k) = A_1^w(s^k) \cup A_2^w \cup A_3 \cup A_4^w$ and define binary wait-and-see variables y_{ij}^k for $(i,j) \in A^w(s^k), k \in \mathcal{N}$ with

$$y_{ij}^k = \begin{cases} 1 & \text{if arc } (i,j) \text{ carries to some flow in the solution for scenario } k \in \mathcal{N}, \\ 0 & \text{otherwise.} \end{cases}$$

In addition, we use wait-and-see variables $c_i^k \geq 0$ for all $i \in V' \backslash \{u_0, v_0\}, k \in \mathcal{N}$. These variables determine the number of arcs on the path from u_0 to i carrying the flow through i in the solution for scenario k. Again, we use an auxiliary variable $v \geq 0$ to calculate the worst-case number of used stacks over all scenarios. We obtain the following MIP formulation for $(arP_{\mathcal{D}})$:

$$
\begin{aligned}
&(Flow - arP_{\mathcal{D}}) \qquad \min \quad v && && (3.36)\\
&\text{s.t.} \quad \sum_{(i,j)\in A_4^h} x_{ij} + \sum_{(i,j)\in A_4^w} y_{ij}^k \le v && \forall k \in \mathcal{N} && (3.37)\\
&x_{ij} = 1 && \forall\ (i,j) \in A_0 && (3.38)\\
&\sum_{j:(i,j)\in A^h} x_{ij} + \sum_{j:(i,j)\in A^w(s^k)} y_{ij}^k = 1 && \forall i \in V'\backslash\{u_0, v_0\}, k \in \mathcal{N} && (3.39)\\
&\sum_{j:(j,i)\in A^h} x_{ji} + \sum_{j:(j,i)\in A^w(s^k)} y_{ji}^k = 1 && \forall i \in V'\backslash\{u_0, v_0\}, k \in \mathcal{N} && (3.40)\\
&c_i^k = 1 && \forall i \in \{u_1, \dots, u_m\}, k \in \mathcal{N} && (3.41)\\
&c_j^k - c_i^k + b(1 - x_{ij}) \ge 1 && \forall (i,j) \in A^h, j \neq v_0, k \in \mathcal{N} && (3.42)\\
&c_j^k - c_i^k + b(1 - y_{ij}^k) \ge 1 && \forall (i,j) \in A^w(s^k), k \in \mathcal{N} && (3.43)\\
&c_i^k \le b + 2 && \forall i \in \{v_1, \dots, v_m\}, k \in \mathcal{N} && (3.44)\\
&c_i^k \ge 0 && \forall i \in V'\backslash\{u_0, v_0\}, k \in \mathcal{N} && (3.45)\\
&x_{ij} \in \{0,1\} && \forall (i,j) \in A^h && (3.46)\\
&y_{ij}^k \in \{0,1\} && \forall (i,j) \in A^w(s^k), k \in \mathcal{N} && (3.47)\\
&v \ge 0 && && (3.48)
\end{aligned}
$$

The number of used stacks in the solution for scenario $k \in \mathcal{N}$ is computed by the left hand side of (3.37), hence the objective function in (3.36) minimizes the worst-case number of used stacks over all scenarios in $\mathcal{N}$. Through constraints (3.38), the arcs in A_0 must always carry some flow, so the given stacking configuration of I^{fix}-items is respected. Constraints (3.39) and (3.40) ensure that each node (except the source u_0 and the sink v_0) belongs to exactly one path in each scenario $k \in \mathcal{N}$. Similar to (3.11)-(3.13), constraints (3.41)-(3.44) guarantee that no cycles occur in the flow and that each stack contains at most b items in each scenario $k \in \mathcal{N}$. This formulation contains $\mathcal{O}(n^2N)$ variables and $\mathcal{O}(n^2N)$ constraints.

Bin packing formulation

In the following we adapt the two bin packing formulations proposed in Section 3.1.3 to model the adjustable robust counterpart $(arP_{\mathcal{D}})$ in the situation of transitive stacking constraints.

Motivated by the IP formulation $(Bin-P)$, we distinguish two sets of binary variables: here-and-now variables x_{iq} for $(i,q) \in (I^{fix} \cup I^1) \times Q$ and wait-and-see variables y_{iq}^k for $(i,q) \in I^2 \times Q, k \in \mathcal{N}$ where

$$x_{iq} = \begin{cases} 1 & \text{if item } i \in I^{fix} \cup I^1 \text{ is stored in stack } q,\\ 0 & \text{otherwise,}\end{cases}$$

$$y_{iq}^k = \begin{cases} 1 & \text{if item } i \in I^2 \text{ is stored in stack } q \text{ in the solution for scenario } k \in \mathcal{N},\\ 0 & \text{otherwise.}\end{cases}$$

To compute the number of used stacks in the solution for scenario $k \in \mathcal{N}$, we introduce binary variables z_q^k for $q \in Q, k \in \mathcal{N}$ with

$$z_q^k = \begin{cases} 1 & \text{if stack } q \text{ contains at least one item in the solution for scenario } k \in \mathcal{N}, \\ 0 & \text{otherwise.} \end{cases}$$

Again, we use an auxiliary variable $v \geq 0$ to calculate the worst-case number of used stacks over all scenarios. We obtain the following MIP formulation for $(arP_{\mathcal{D}})$:

$$(Bin - arP_{\mathcal{D}}) \qquad \min \quad v \tag{3.49}$$

$$\text{s.t.} \quad \sum_{q \in Q} z_q^k \leq v \qquad \forall k \in \mathcal{N} \tag{3.50}$$

$$x_{iq} = 1 \qquad \forall (i,q) \in F_{bin} \tag{3.51}$$

$$\sum_{i \in I^{fix} \cup I^1} x_{iq} + \sum_{j \in I^2} y_{jq}^k \leq b z_q^k \qquad \forall q \in Q, k \in \mathcal{N} \tag{3.52}$$

$$\sum_{q \in Q} x_{iq} = 1 \qquad \forall i \in I^1 \tag{3.53}$$

$$\sum_{q \in Q} y_{iq}^k = 1 \qquad \forall i \in I^2, k \in \mathcal{N} \tag{3.54}$$

$$\sum_{\substack{j \in I^1 \setminus \{i\} \\ s_{ij}=s_{ji}=0}} x_{jq} + \sum_{\substack{j \in I^2 \\ s_{ji}^k=0}} y_{jq}^k \leq (b - b_q)(1 - x_{iq}) \qquad \forall (i,q) \in F_{bin}^{top} \cup I^1 \times Q', k \in \mathcal{N} \tag{3.55}$$

$$\sum_{\substack{j \in I^1 \\ s_{ij}^k=0}} x_{jq} + \sum_{\substack{j \in I^2 \setminus \{i\} \\ s_{ij}^k=s_{ji}^k=0}} y_{jq}^k \leq (b - b_q)(1 - y_{iq}^k) \qquad \forall (i,q) \in I^2 \times Q', k \in \mathcal{N} \tag{3.56}$$

$$\sum_{\substack{j \in I^1 \setminus \{i\} \\ s_{ij}=s_{ji}=0}} x_{jq} + \sum_{\substack{j \in I^2 \\ s_{ji}^k=0}} y_{jq}^k \leq b(1 - x_{iq}) \qquad \forall (i,q) \in I^1 \times (Q \setminus Q'), k \in \mathcal{N} \tag{3.57}$$

$$\sum_{\substack{j \in I^1 \\ s_{ij}^k=0}} x_{jq} + \sum_{\substack{j \in I^2 \setminus \{i\} \\ s_{ij}^k=s_{ji}^k=0}} y_{jq}^k \leq b(1 - y_{iq}^k) \qquad \forall (i,q) \in I^2 \times (Q \setminus Q'), k \in \mathcal{N}, \tag{3.58}$$

$$x_{iq} \in \{0,1\} \qquad \forall i \in I^{fix} \cup I^1, q \in Q \tag{3.59}$$

$$y_{jq}^k \in \{0,1\} \qquad \forall j \in I^2, q \in Q, k \in \mathcal{N} \tag{3.60}$$

$$z_q^k \in \{0,1\} \qquad \forall q \in Q, k \in \mathcal{N} \tag{3.61}$$

$$v \geq 0 \tag{3.62}$$

The left hand side of (3.50) is equal to the number of used stacks in the solution for scenario $k \in \mathcal{N}$, hence the objective function in (3.49) minimizes the worst-case number of used stacks over all scenarios in $\mathcal{N}$. Constraints (3.51) ensure that all I^{fix}-items are stored in their corresponding stacks. Constraints (3.52) guarantee that at most b items are assigned to each stack in each scenario and enforce variable z_q^k to one if at least one item is assigned to stack q in scenario $k \in \mathcal{N}$. Equalities (3.53)-(3.54) guarantee that each item is stored in exactly one stack in each scenario. Finally, (3.55)-(3.58) ensure that only compatible items are assigned to the same stack in each scenario. This formulation contains $\mathcal{O}(nmN)$ variables and $\mathcal{O}(nmN)$ constraints.

In the same way that the redundant constraints (3.24) were added to $(Bin - P)$ (cf. Section 3.1.3), we can add the following constraints to $(Bin - arP_{\mathcal{D}})$:

$$x_{iq} \leq z_q^k \qquad \forall i \in I^{fix} \cup I^1, q \in Q, k \in \mathcal{N}, \tag{3.63}$$

$$y_{jq}^k \leq z_q^k \qquad \forall j \in I^2, q \in Q, k \in \mathcal{N}. \tag{3.64}$$

Then we obtain the extended MIP formulation

$$(BinE - arP_{\mathcal{D}}) \qquad \{\min v : (3.50) - (3.64)\}.$$

In comparison with $(Bin-arP_{\mathcal{D}})$, this MIP formulation uses the same number of variables, also contains $\mathcal{O}(nmN)$ constraints, but may have a better LP relaxation.

3.3.2 Interval uncertainty

In this subsection, we consider the adjustable robust counterpart of $(P_{\mathcal{I}})$ where the stacking constraints define a total order based on associated values a_i of items and we have interval uncertainties $[a_i^{min}, a_i^{max}]$ for all $i \in I^2$.

We have shown in Theorem 2.15 that for finding an adjustable robust solution to $(P_{\mathcal{I}})$ it is sufficient to only consider the dominant scenario a^{max} in which all wait-and-see items have maximum values of their associated parameters. Let (s_{ij}^{max}) be the stacking matrix for the dominant scenario a^{max}. Thanks to Theorem 2.15, replacing the stacking matrix (s_{ij}) in formulations $(Ind-P)$, $(Flow-P)$, $(Bin-P)$, $(BinE-P)$ from Section 3.1 by the stacking matrix (s_{ij}^{max}), we obtain MIP formulations for $(arP_{\mathcal{I}})$ denoted by $(Ind - arP_{\mathcal{I}})$, $(Flow - arP_{\mathcal{I}})$, $(Bin - arP_{\mathcal{I}})$, $(BinE - arP_{\mathcal{I}})$, respectively. In all these formulations no additional variables or constraints are needed. Note that the matrix (s_{ij}^{max}) differs from the strict stacking matrix (s_{ij}^*) only with respect to the fourth line in (2.7). More precisely, for $i, j \in I^2$ we have $s_{ij}^{max} = 1$ if $a_i^{max} \leq a_j^{max}$, while $s_{ij}^* = 1$ if $a_i^{max} \leq a_j^{min}$. It means that, in comparison with (s_{ij}^*), some more 1-entries may exist in (s_{ij}^{max}), which allows more flexibility in the adjustable case than requiring strict robustness.

3.4 Computational experiments

In this section, we compare the different MIP formulations from Sections 3.2 and 3.3, as summarized in Table 3.1. Recall that the bin packing formulations can only be applied to instances with transitive stacking constraints.

Uncertainty	Robustness	Type of formulation			
		Three-index	Network flow	Bin packing	
Finite	Strict	$(Ind - srP_{\mathcal{D}})$	$(Flow - srP_{\mathcal{D}})$	$(Bin - srP_{\mathcal{D}})$	$(BinE - srP_{\mathcal{D}})$
	Adjustable	$(Ind - arP_{\mathcal{D}})$	$(Flow - arP_{\mathcal{D}})$	$(Bin - arP_{\mathcal{D}})$	$(BinE - arP_{\mathcal{D}})$
Interval	Strict	$(Ind - srP_{\mathcal{I}})$	$(Flow - srP_{\mathcal{I}})$	$(Bin - srP_{\mathcal{I}})$	$(BinE - srP_{\mathcal{I}})$
	Adjustable	$(Ind - arP_{\mathcal{I}})$	$(Flow - arP_{\mathcal{I}})$	$(Bin - arP_{\mathcal{I}})$	$(BinE - arP_{\mathcal{I}})$

Table 3.1: Summary of the MIP formulations from Sections 3.2 and 3.3.

Our main goal is to determine the best formulations for different problem instances, different types of data uncertainty and different structures of stacking constraints. We also compute the gain of using solutions that include adjustable decisions on items that arrive later in comparison with strictly robust solutions.

3.4.1 Setup

For evaluating the performance of the MIP formulations summarized in Table 3.1, we implemented them using ZIMPL 3.3.2 (cf. [60]) and GUROBI 6.5.1 as MIP solver. All experiments were conducted on a computer with a Core 2 Duo 2 * 2.1 GHz processor and 2 GB of RAM. For each run, a time limit of 30 minutes was imposed to the MIP solver.

In practice, the storage area in a rail-road container terminal contains up to 60 stacks with at most $b = 3$ containers in each stack. When considering maritime container terminals, the yard contains up to 180 stacks with at most $b = 6$ containers in each stack. In our experiments, we generated instances with up to $n = 480$ items and $b = 6$ levels.

A problem instance is specified by the cardinalities of the item sets (i.e., the triple $(|I^{fix}|, |I^1|, |I^2|)$), together with the common height b of stacks, and the stacking configuration of the I^{fix}-items. The set $\mathcal{S}_{\mathcal{D}}$ of stacking matrix scenarios (in case of finite uncertainty) and the a-values for the items (in case of interval uncertainty) have to to be given as well.

To generate problem instances, we first specified the common height b of the stacks and the number $n = |I|$ of all items. For each combination of (b, n) we randomly generated the triple $(|I^{fix}|, |I^1|, |I^2|)$, in which $|I^{fix}|$ and $|I^1|$ range from $\lfloor \frac{n}{4} \rfloor$ to $\lfloor \frac{n}{2} \rfloor$, and $|I^2| = n - |I^{fix}| - |I^1|$. To generate a stacking configuration of the I^{fix}-items, we let the first $m_f = \lceil \frac{n}{6} \rceil$ stacks of the corresponding storage area be the I^{fix}-area for each setting of $\mathcal{T} = (b, n, |I^{fix}|, |I^1|, |I^2|)$. The average number of I^{fix}-items stored in each stack in the I^{fix}-area is therefore $b_f = \frac{|I^{fix}|}{m_f}$. According to the assumption that no stack is fully filled by I^{fix}-items, the number of I^{fix}-items stored in each stack must be smaller than b. Therefore if $\lceil b_f \rceil = b$ we reset m_f to $\lceil \frac{n}{3} \rceil$. Then, within the I^{fix}-area we randomly generated two instances for the stacking configuration of the I^{fix}-items: one instance is generated in *spread style*, the other in *non-spread style*. With "spread style", we mean that the I^{fix}-items are distributed "uniformly" in the I^{fix}-area, or more precisely, each stack in the I^{fix}-area contains at most $\lceil b_f \rceil$ items from I^{fix}. In the "non-spread style", some stacks in the I^{fix}-area can contain more than $\lceil b_f \rceil$ (but less than b) items from I^{fix}.

We organized our experiments as follows.

- Experiment 1: test the two MIPs based on the three-index and network flow formulations on instances of $(P_{\mathcal{D}})$ with finite uncertainty and arbitrary stacking constraints.
- Experiment 2: test the MIPs based on the three-index, network flow, and bin packing formulations on instances of $(P_{\mathcal{D}})$ with finite uncertainty and transitive stacking constraints.
- Experiment 3: test the MIPs based on the three-index, network flow, and bin packing formulations on instances of $(P_{\mathcal{I}})$ with interval uncertainty and total order stacking constraints.

We describe the detailed setup for the experiments in Section 3.4.3. In each experiment, we first compared the performance of all MIP formulations on problem instances of moderate size (up to $n = 90$ and $b = 5$). Then we performed experiments with the best formulations to know the largest instance size which can be solved to optimality within the given time limit.

The number m of available stacks in the storage area was chosen in such a way that the feasibility of storing all items into the storage area is guaranteed. Note that by definition, the concept of strictly robust stacking solutions is more conservative than that of adjustable ones. Thus, strictly robust stacking solutions in general use more stacks than the corresponding adjustable ones. Therefore, in the strict models we use a larger value for m than in the adjustable models. When setting m, we additionally took into account that total order stacking constraints usually offer more possibilities to stack items onto each other than partial orders, and transitive stacking constraints offer more possibilities in comparison with arbitrary ones. Table 3.2 shows the precise setup for the values of m in our experiments. We first solved the generated instances with these settings. Then for any infeasible instance we reset $m = n$ (so that all items are always feasibly stored) and solved again these instances. This situation occured rarely and did not have any significant impact on the computational times since the presolver of the MIP solver eliminates superfluous variables very fast.

Experiment	Strict robustness	Adjustable robustness
Experiment 1	$m = \lceil 0.8n \rceil$	$m = \lceil 0.75n \rceil$
Experiment 2	$m = \lceil 0.75n \rceil$	$m = \lceil 0.6n \rceil$
Experiment 3	$m = \lceil 0.6n \rceil$	$m = \lceil 0.5n \rceil$

Table 3.2: Number of stacks m in the generated instances.

3.4.2 Reduction of solution space

Since the stacks that do not contain any I^{fix}-item are interchangeable, a lot of symmetric solutions exist (e.g., any permutation of empty and used stacks leads to an equivalent solution using the same number of stacks). For this reason, we tried to restrict the solution space by imposing some symmetry breaking restrictions. To do this, at first we numbered the stacks in the storage area from 1 to m. Each stacking configuration of the I^{fix}-items was generated in such a way that the stacks containing these items have consecutive numbers (counting from stack 1). Then we added some symmetry breaking constraints to the MIP formulations based on one of the following ideas.

- Used stacks have consecutive indices, i.e., stack $q-1$ must be used if stack q is used.
- Each non-I^{fix} stack has at least as many items as the next one, i.e., in the set of stacks containing no I^{fix}-item, stack q has at most as many items as stack $q-1$.

Let q^* be the largest-indexed stack containing I^{fix}-items, and $Q^* = Q \setminus \{1, \dots, q^*+1\}$. We first describe more precisely how the former idea can be implemented. For $(Ind - srP_{\mathcal{D}})$, $(Ind - srP_{\mathcal{I}})$, and $(Ind - arP_{\mathcal{I}})$ we add the following constraints:

$$\sum_{i \in I} x_{i,q,1} \leq \sum_{i \in I} x_{i,q-1,1} \qquad \forall\, q \in Q^*, \tag{3.65}$$

while for $(Ind - arP_{\mathcal{D}})$ the following constraints are added:

$$\sum_{i \in I^{fix} \cup I^1} x_{i,q,1} + \sum_{j \in I^2} y^k_{j,q,1} \leq \sum_{i \in I^{fix} \cup I^1} x_{i,q-1,1} + \sum_{j \in I^2} y^k_{j,q-1,1} \qquad \forall\, q \in Q^*, k \in \mathcal{N}. \tag{3.66}$$

With the same purpose, we add the following constraints to $(Flow - srP_{\mathcal{D}})$, $(Flow - srP_{\mathcal{I}})$, and $(Flow - arP_{\mathcal{I}})$:

$$\sum_{i \in I:(i,v_q) \in A_4} x_{iv_q} \leq \sum_{i \in I:(i,v_{q-1}) \in A_4} x_{iv_{q-1}} \qquad \forall\, q \in Q^*, \tag{3.67}$$

and integrate the following constraints into $(Flow - arP_{\mathcal{D}})$:

$$\sum_{\substack{i \in I^{fix} \cup I^1 \\ (i,v_q) \in A_4^h}} x_{iv_q} + \sum_{\substack{j \in I^2 \\ (j,v_q) \in A_4^w}} y_{jv_q}^k \leq \sum_{\substack{i \in I^{fix} \cup I^1 \\ (i,v_{q-1}) \in A_4^h}} x_{iv_{q-1}} + \sum_{\substack{j \in I^2 \\ (j,v_{q-1}) \in A_4^w}} y_{jv_{q-1}}^k \quad \forall\, q \in Q^*, k \in \mathcal{N}. \tag{3.68}$$

For the bin packing formulations of the adjustable models with finite uncertainty (i.e., $(Bin - arP_{\mathcal{D}})$, $(BinE - arP_{\mathcal{D}})$), we add the constraints

$$z_q^k \leq z_{q-1}^k \qquad \forall\, q \in Q^*, k \in \mathcal{N}, \tag{3.69}$$

while for the other bin packing formulations we add

$$z_q \leq z_{q-1} \qquad \forall\, q \in Q^*. \tag{3.70}$$

The symmetry breaking constraints in the spirit of the latter idea are as follows. We add the following constraints to $(Ind - srP_{\mathcal{D}})$, $(Ind - srP_{\mathcal{I}})$, and $(Ind - arP_{\mathcal{I}})$:

$$\sum_{(i,l) \in (I^1 \cup I^2) \times L} x_{i,q,l} \leq \sum_{(i,l) \in (I^1 \cup I^2) \times L} x_{i,q-1,l} \qquad \forall\, q \in Q^*, \tag{3.71}$$

while for $(Ind - arP_{\mathcal{D}})$ we add

$$\sum_{\substack{i \in I^1 \\ l \in L}} x_{i,q,l} + \sum_{\substack{j \in I^2 \\ l \in L}} y_{j,q,l}^k \leq \sum_{\substack{i \in I^1 \\ l \in L}} x_{i,q-1,l} + \sum_{\substack{j \in I^2 \\ l \in L}} y_{j,q-1,l}^k \qquad \forall\, q \in Q^*, k \in \mathcal{N}. \tag{3.72}$$

For $(Flow - srP_{\mathcal{D}})$, $(Flow - srP_{\mathcal{I}})$, and $(Flow - arP_{\mathcal{I}})$, the symmetry breaking constraints are

$$c_{v_q} \leq c_{v_{q-1}} \qquad \forall\, q \in Q^*, \tag{3.73}$$

while for $(Flow - arP_{\mathcal{D}})$ the symmetry breaking constraints are

$$c_{v_q}^k \leq c_{v_{q-1}}^k \quad \forall\, q \in Q^*, k \in \mathcal{N}. \tag{3.74}$$

For the bin packing formulations of the adjustable models with discrete uncertainty (i.e., $(Bin - arP_{\mathcal{D}})$, $(BinE - arP_{\mathcal{D}})$), we add the constraints

$$\sum_{i \in I^1} x_{iq} + \sum_{j \in I^2} y_{jq}^k \leq \sum_{i \in I^1} x_{i,q-1} + \sum_{j \in I^2} y_{j,q-1}^k \qquad \forall\, q \in Q^*, k \in \mathcal{N}, \tag{3.75}$$

while for the other bin packing formulations we integrate the following constraints:

$$\sum_{i \in I^1 \cup I^2} x_{iq} \leq \sum_{i \in I^1 \cup I^2} x_{i,q-1} \qquad \forall\, q \in Q^*. \tag{3.76}$$

The optimal values of the LP relaxations of the original MIP formulations do not change when the corresponding symmetry breaking constraints are added. Indeed, given any solution to such an original MIP formulation, we can rearrange the non-I^{fix} stacks in the solution to satisfy the corresponding additional symmetry breaking constraints and still obtain the same number of used stacks. However, different symmetry breaking constraints may have different impacts on performance of the proposed MIP formulations. A similar phenomenon has been reported for some symmetry breaking constraints in other applications such as scheduling of a doubles tennis training tournament [47], set partitioning [48], lot-sizing problems on parallel identical machines [55], job grouping problem [56]. In our case study of modeling the storage loading problems, we did some preliminary computational experiments and compared the original versions of the proposed MIP formulations to the versions equipped with the symmetry breaking constraints. We found out that the bin packing MIP formulations perform better when being combined with (3.69)-(3.70), while the network flow formulations have better performance when being equipped by (3.73)-(3.74). The three-index formulations equipped by symmetry breaking constraints, however, have worse performance than their original version. A possible explanation for this phenomenon is that some symmetry breaking constraints are compatible with branch-and-bound algorithms implemented in the black-box of the MIP solver GUROBI, while some are not (cf. the concept of *symmetry compatible formulations* introduced in [47]). Therefore, in the following subsections we report numerical results related to the original MIPs for the three-index formulations, the network flow formulations equipped by (3.73)-(3.74), and the bin packing formulations equipped by (3.69)-(3.70).

3.4.3 Computational results

First, we note that in all our experiments, for all instances and all MIP formulations, feasible solutions could be found by the MIP solver within the given time limit of 30 minutes. We call an instance "verified" if the obtained solution could be proven to be optimal within the time limit (i.e., the MIP solver terminated the search using the standard setting of 0.01 % as optimality gap).

In order to have a baseline for comparing the performance of the proposed MIP formulations, we apply the concept of a *performance profile* introduced in [38]. More precisely, let $\mathcal{P}$ be the set of tested instances and $\mathcal{M}$ the subset of the MIP formulations applied to solve the instances in $\mathcal{P}$. For each instance $p \in \mathcal{P}$ and each MIP formulation $f \in \mathcal{M}$, we report $t_{p,f}$ as the computation time (in minutes) to solve p using formulation f. We reset $t_{p,f}$ to 1 if the solver terminates in less than one minute and to 30 if the solver reaches the time limit of 30 minutes. We then compute the *performance ratio*

$$r_{p,f} = \frac{t_{p,f}}{\min\{t_{p,f} : f \in \mathcal{M}\}}$$

which compares the performance of formulation f with the best formulation. It follows from our setting that $r_{p,f} \leq 30$, and $r_{p,f} = 30$ if and only if the solver does not prove optimality for instance p by using formulation f within the time limit. As shown in [38], this way of setting an upper bound for $r_{p,f}$ does not affect the performance evaluation. To have an overall evaluation of the performance of formulation f, for $\tau \geq 1$ we compute

$$\rho_f(\tau) = \frac{|\{p \in \mathcal{P} : r_{p,f} \leq \tau\}|}{|\mathcal{P}|}.$$

In other words, $\rho_f(\tau)$ is the probability that formulation $f \in \mathcal{M}$ is not worse than a factor τ when compared with the best performance formulation. The function ρ_f is called *performance profile* of the formulation f, which encodes all major performance characteristics of f (cf. [38]). In particular, the value of $\rho_f(1)$ is the probability that the formulation f has the best performance (in terms of computation time) among all formulations in $\mathcal{M}$.

Experiment 1: discrete uncertainty with arbitrary stacking constraints

As detailed setup for Experiment 1, for each combination of $\mathcal{T} = (b, n, |I^{fix}|, |I^1|, |I^2|)$ and stacking configurations of I^{fix}-items, we generated five instances with different lists $\mathcal{S}_{\mathcal{D}}$ of stacking matrix scenarios. Each instance of $\mathcal{S}_{\mathcal{D}}$ contains up to ten different, randomly generated stacking matrices. The density of each matrix (before integrating the stacking sequence $I^{fix} \to I^1 \to I^2$) was randomly chosen in the interval $[0.25, 0.75]$ in advance and the first scenario in each instance of $\mathcal{S}_{\mathcal{D}}$ had a density of approximately 0.5. Here the density of an $n \times n$-matrix with binary entries is defined by the ratio $\alpha = \frac{p}{n(n-1)}$, where p is the number of 1-entries in the matrix (the diagonal entries are always assumed to be equal to 0). Given the fact that items in $I^{fix} \cup I^1$ are here-and-now, while items in I^2 are wait-and-see, all stacking matrices in each instance of $\mathcal{S}_{\mathcal{D}}$ have the same deterministic part $(s_{ij})_{i,j \in I^{fix} \cup I^1}$.

Since the generated stacking matrices have an arbitrary structure, we can only apply the three-index and network flow MIP formulations. We tested the formulations on 120 problem instances with $b \in \{3, 4\}$ and $n \in \{30, 60\}$. It turned out that the optimality of only very small instances was verified within the time limit for the adjustable robust problem. More precisely, when using the network flow formulation, the time limit was reached for 41 instances out of 60 with only $n = 30$ items, while for $n = 60$ items, the optimality of only 10 instances out of 60 was verified. Using the three-index formulation, the optimality of 70 instances out of 120 was verified.

	$(srP_{\mathcal{D}})$		$(arP_{\mathcal{D}})$	
	(Ind)	$(Flow)$	(Ind)	$(Flow)$
Number of verified instances	116	95	70	29
Average running time (seconds)	30.3	42.7	272.6	766.7
Average algorithm gap (%)	20.4	23.7	31.4	43.4

Table 3.3: Summarized results for Experiment 1 (120 instances).

A summary of the numerical results is reported in Table 3.3, where the average running time is taken over all verified instances, and the average algorithm gap is computed for the instances where both approaches resulted in no verified solutions. In solving both the strictly and the adjustable robust problems, the three-index formulation performs better than the network flow approach (more solutions can be proven to be optimal). For the tested instances where both approaches resulted in no verified solutions, the three-index approach also performs better than the network flow approach (having a smaller average algorithm gap).

Figure 3.2 shows the performance profiles of the three-index and network flow formulations on the strictly and adjustable robust problems. From this figure it can be seen that the three-index formulation has the highest probability of being the best formulation in solving both the strictly and adjustable robust problems. In particular, the probability that the three-index formulation is the best in solving the adjustable robust problems is 0.94, while the corresponding value for the network flow formulation is just 0.59. The three-index formulation also has a better performance than the network flow formulation for every value of factor τ.

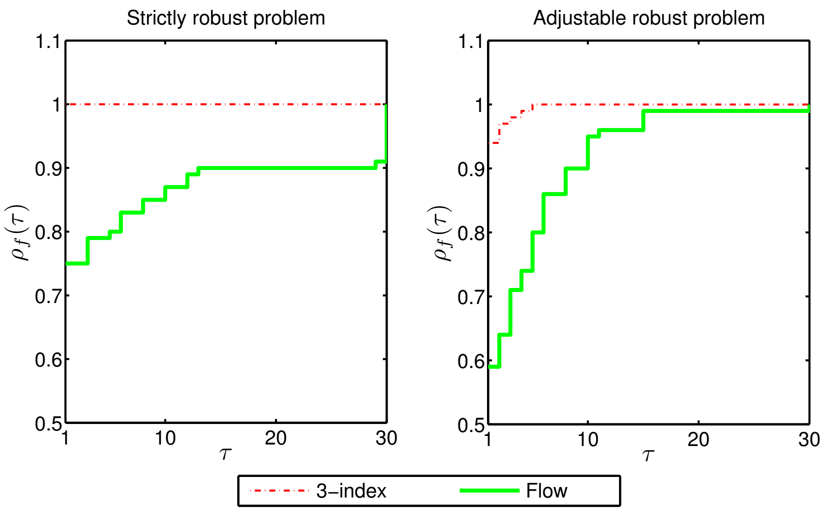


Figure 3.2: Performance profile of formulations on tested instances in Experiment 1.

Experiment 2: discrete uncertainty with transitive stacking constraints

The detailed setup for Experiment 2 is as follows. For each choice of the tuple $\mathcal{T} = (b, n, |I^{fix}|, |I^1|, |I^2|)$ and each stacking configuration of I^{fix}, we generated five instances with different lists $\mathcal{S}_\mathcal{D}$ of stacking matrix scenarios. Motivated by practical constraints, the instances of $\mathcal{S}_\mathcal{D}$ were generated in the following way. We assigned to each item $i \in I$ integral values w_i for its weight (in tons) and d_i for its departure time (in hours, counting from the loading start time of I^1 to the time that item i is retrieved). The stacking constraints in this context were then defined by setting

$$s_{ij} = \begin{cases} 1 & \text{if } w_i \le w_j \text{ and } d_i \le d_j, \\ 0 & \text{otherwise,} \end{cases} \tag{3.77}$$

which means that only an item of lighter weight and earlier departure time is allowed to be put on top of an item of heavier weight and later departure time. Clearly, the stacking constraints defined by (3.77) are transitive. We randomly generated a scenario for $\{(w_i, d_i)|i \in I^{fix} \cup I^1\}$ and different scenarios for $\{(w_i, d_i)|i \in I^2\}$ in such a way that firstly, $(w_i, d_i) \in [5, 50] \times [10, 100]$ for all $i \in I$, and secondly, the generated values for $\{(w_i, d_i)|i \in I^{fix}\}$ satisfy the given stacking configuration of I^{fix}-items as well as the stacking constraints defined by (3.77). With the generated scenario for $\{(w_i, d_i)|i \in I^{fix} \cup I^1\}$, each scenario for $\{(w_i, d_i)|i \in I^2\}$ corresponds to a stacking matrix which is computed according to (3.77). We generated up to 10 different stacking matrices for each instance of $\mathcal{S}_\mathcal{D}$.

We tested the corresponding formulations on 90 instances with $b \in \{3, 4, 5\}$ and $n \in \{30, 60, 90\}$. Table 3.4 summarizes some results from this experiment. Again, for each formulation, the average running time is taken over all verified instances, and the average algorithm gap is computed for the instances that resulted in no verified solutions. When solving both the strictly and the adjustable robust problems, the network flow formulation performs better than the others over all mentioned criteria: more solutions are proved to be optimal, while smaller average running times and smaller algorithm gaps are observed.

Figure 3.3 illustrates the performance profiles of these formulations on the strictly and adjustable robust counterparts. It is easy to see that the network flow approach

	$(srP_{\mathcal{D}})$				$(arP_{\mathcal{D}})$			
	(Ind)	$(Flow)$	(Bin)	$(BinE)$	(Ind)	$(Flow)$	(Bin)	$(BinE)$
Number of verified instances	42	90	79	88	15	82	53	63
Average running time (seconds)	107.7	0.9	99.9	63.8	454.3	20.7	492.8	540.2
Average algorithm gap (%)	9.7	0.0	2.3	2.1	26.0	11.1	11.2	22.8

Table 3.4: Summarized results for Experiment 2 (90 instances).

performs best for both $(srP_{\mathcal{D}})$ and $(arP_{\mathcal{D}})$. It has the highest probability to be the best formulation (100% for the former and 93% for the latter), and it is the best formulation for every value of factor τ. Therefore, the network flow approach is definitely the "winner" in this experiment.

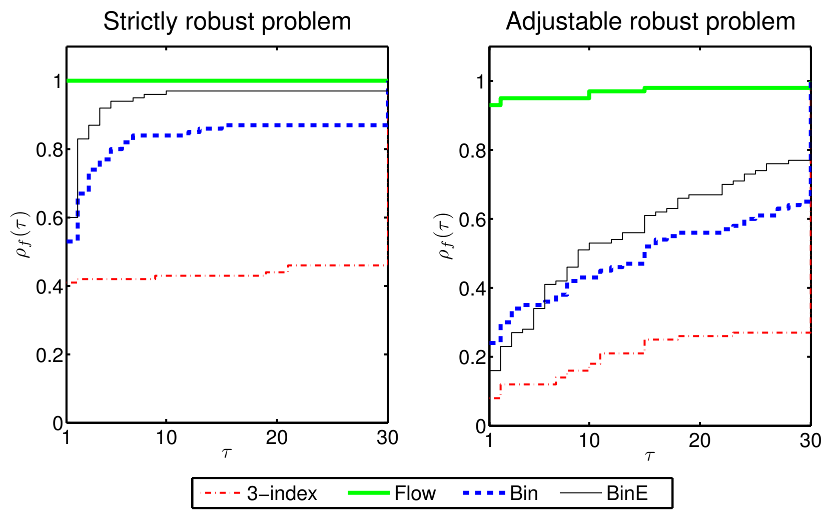


Figure 3.3: Performance profile of formulations on tested instances in Experiment 2.

Additionally, it can be seen from Figure 3.3 that $(BinE - srP_{\mathcal{D}})$ has a better performance profile than $(Bin - srP_{\mathcal{D}})$. Moreover, when $\tau \geq 10$, formulation $(BinE - srP_{\mathcal{D}})$ is a very good choice for solving $(srP_{\mathcal{D}})$. For the adjustable robust problem, $(Bin - arP_{\mathcal{D}})$ has a slightly higher probability to be the best formulation than $(BinE - arP_{\mathcal{D}})$. However, if we increase the factor τ, the latter formulation performs slightly better than the former. The (Ind)-approach has the worst performance in solving both $(srP_{\mathcal{D}})$ and $(arP_{\mathcal{D}})$.

The efficiency of the network flow approach in this experiment can be explained by the fact that the densities of the stacking matrices in the generated instances are quite small (in the tested instances they range from 0.2 to 0.35). This results in less choices for stacking items together into stacks. Consequently, it has a positive impact on the network flow models, since in these models one needs to determine not only which item is stored in which stack but also the location of each item in its stack.

To see the limits of the best performing network flow approach, we tested $(Flow - arP_{\mathcal{D}})$ on six problem instances of larger size. Table 3.5 reports the running times of the tests, and gives an idea of the size of the largest instances that can be proven to be optimal within the time limit of 30 minutes. For $|\mathcal{S}_{\mathcal{D}}| = 5$, $(Flow - arP_{\mathcal{D}})$ can prove the optimality of instances with up to $n = 270$ and $b = 6$. For $|\mathcal{S}_{\mathcal{D}}| = 10$ the limit is about $n = 180$ and $b = 6$, while for $|\mathcal{S}_{\mathcal{D}}| = 20$ at most 150 items can be handled.

We also evaluated the quality (feasibility) of the strictly robust solutions when more

n	b	$\lvert\mathcal{S}_\mathcal{D}\rvert$	Time	Algorithm gap
120	5	20	295 seconds	0
150	5	20	Time limit reached	9.3%
180	6	10	166 seconds	0
270	6	5	334 seconds	0
300	6	5	Time limit reached	4.41%

Table 3.5: Additional tests of $(Flow - arP_\mathcal{D})$ on larger instances in Experiment 2.

stacking matrix scenarios are taken into account. Our evaluation framework is as follows. At first we compute a strictly robust solution to each problem instance with $|\mathcal{S}_\mathcal{D}| = 5$ stacking matrix scenarios. Then we generate 5 more stacking matrix scenarios. For each additional scenario, we count how many pairs of stacked items in the strictly robust solution that do not satisfy the stacking constraints defined by the stacking matrix. For the tested instances, in the best case the strictly robust solution is still feasible in all additional scenarios. On average, 10.55% pairs of stacked items in the computed strictly robust solutions do not satisfy the new scenarios of stacking constraints, while the value in the worst case is 13.89%. Thus, the obtained strictly robust solutions still have a high possibility to be feasible if we consider more stacking matrix scenarios. However, in terms of the optimal objective value, strictly robust solutions may use at least 35.7% and at most 110.7% more stacks than the optimal solutions to the additional scenarios.

Experiment 3: interval uncertainty with total order stacking constraints

In Experiment 3, for each choice of $\mathcal{T} = (b, n, |I^{fix}|, |I^1|, |I^2|)$ and each stacking configuration of I^{fix}-items, we generated randomly five instances for $\{a_i \mid i \in I^{fix} \cup I^1\} \cup \{a_j^{min}, a_j^{max} \mid j \in I^2\}$, in which the former set contains parameters a_i of here-and-now items $i \in I^{fix} \cup I^1$ and the latter set contains the endpoints of the intervals $[a_j^{min}, a_j^{max}]$ for the parameters a_j of wait-and-see items $j \in I^2$. The generated values satisfy $a_i \in [5, 100]$ for all $i \in I^{fix} \cup I^1$ and $5 \leq a_j^{min} \leq a_j^{max} \leq 100$ for all $j \in I^2$. The generated values $\{a_i | i \in I^{fix}\}$ respect the stacking constraints defined by $s_{ij} = 1 \Leftrightarrow a_i \leq a_j$ and the given stacking configuration of the I^{fix}-items.

We performed tests on 90 problem instances with $b \in \{3, 4, 5\}$ and $n \in \{30, 60, 90\}$. An overview of the computational results for this experiment is given in Table 3.6, where again for each formulation the average running time is taken over all verified instances and the average algorithm gap is computed for the instances that resulted in no verified solutions. When solving both the strictly and the adjustable robust problems, the bin packing formulations perform better than the three-index and network flow formulations (with more verified instances, shorter average solver running time, and smaller average algorithm gap). In particular, the bin packing models (Bin) and $(BinE)$ are both very efficient for solving adjustable robust problem $(arP_\mathcal{I})$. That is, all tested instances are quickly solved to optimality.

In Figure 3.4 we show the performance profiles of the formulations in Experiment 3. The left part of this figure shows that both bin packing formulations have high probability of being the best formulation in solving strictly robust problem $(srP_\mathcal{I})$, where $(BinE - srP_\mathcal{I})$ has a slightly better performance profile than $(Bin - srP_\mathcal{I})$. With a probability of more than 80% to be the best, the network flow formulation $(Flow - srP_\mathcal{I})$ is also a good choice. The right part of Figure 3.4 shows that, for solving $(arP_\mathcal{I})$ with $b \leq 5$ and $n \leq 90$,

	$(srP_{\mathcal{I}})$				$(arP_{\mathcal{I}})$			
	(Ind)	$(Flow)$	(Bin)	$(BinE)$	(Ind)	$(Flow)$	(Bin)	$(BinE)$
Number of verified instances	37	73	84	85	70	20	90	90
Average running time (seconds)	129.9	3.3	15.1	51.2	248.6	107.9	0.4	0.9
Average algorithm gap (%)	26.9	7.9	3.8	2.9	26.6	14.5	0.0	0.0

Table 3.6: Summarized results for Experiment 3 (90 instances).

all bin packing formulations have the same performance and they are the best choices with 100% of being the best performance formulations. However, unlike the case of solving the strictly robust counterparts $(srP_{\mathcal{I}})$, for solving the adjustable robust counterparts $(arP_{\mathcal{I}})$, the network flow approach $(Flow - arP_{\mathcal{I}})$ has the worst performance.

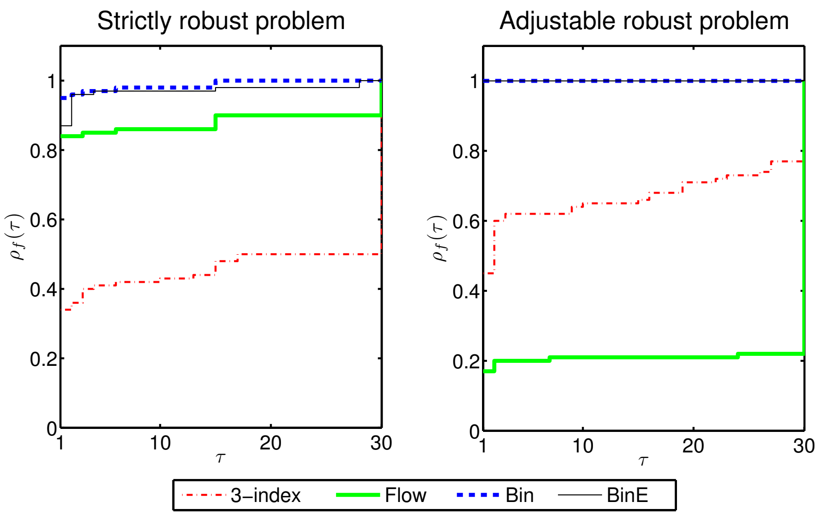


Figure 3.4: Performance profile of formulations on tested instances in Experiment 3.

A reason for the efficiency of the bin packing formulations to solve instances of $(arP_{\mathcal{I}})$ may be that the stacking matrices have a higher density in the tested instances, when compared to the previous experiment (ranging from 0.4 to 0.6), which offers more choices for stacking items together into the same stack. Since the bin packing models only need to decide about the combinations of items that are stored together in one stack, this can be done in a small amount of time.

Having a look at the detailed computational results, we see that for the same problem instance, the bin packing approaches solve $(arP_{\mathcal{I}})$ faster than $(srP_{\mathcal{I}})$, while it is the opposite for the network flow approach. This is due to the strict stacking matrix (s^*) which has a smaller density than the one in the dominant scenario (s^{max}). Note that, unlike in Experiment 2, the network flow approach is no longer efficient in Experiment 3 due to the larger densities of the stacking matrices.

To see the limit and clarify the performance of the best performing bin packing formulations, we tested them on problem instances of $(arP_{\mathcal{I}})$ of larger size $(n \geq 300)$. Computational results show that instances of $(arP_{\mathcal{I}})$ with up to $b = 6$ and $n = 480$ can be optimally solved within 10 minutes. On the other hand, on our computer with 2 GB of RAM, instances with $n = 510$ are already out of memory. For 80% of the larger instances, the extended formulation $(BinE - arP_{\mathcal{I}})$ has better running time than the others, while the original formulation $(Bin - arP_{\mathcal{I}})$ wins on the remaining instances.

Comparing strictly and adjustable robust solutions

To compare the strict and adjustable robustness approaches, we calculated for each instance how much we gain when we allow "wait-and-see" stacking decisions for I^2-items (i.e., items arriving later) in comparison with using strictly robust decisions. This gain is measured by

$$\text{gain} = \frac{f_{sr} - f_{ar}}{f_{sr}} \times 100\%,$$

where f_{sr} (resp., f_{ar}) is the best objective value computed by the MIP formulations for strictly (resp., adjustable) robustness. Note that if at least one of the corresponding solutions to the strictly (resp., adjustable) robust problem in the test is proved to be optimal, then the computed value of f_{sr} (resp., f_{ar}) is actually optimal.

Experiment	I^{fix}-configuration	Maximum gain	Average gain
Experiment 1	Non-spread	53.12%	34.94%
	Spread	54.54%	34.16%
Experiment 2	Non-spread	53.33%	41.78%
	Spread	52.54%	40.45%
Experiment 3	Non-spread	38.23%	19.33%
	Spread	33.33%	17.33%

Table 3.7: Gain of using adjustable robustness in comparison with strict robustness.

Table 3.7 reports the gain of using adjustable robustness in comparison with strict robustness (in terms of the optimal objective value) on our test instances. It shows that the use of adjustable robust stacking solutions achieves a considerable gain in comparison with strictly robust ones. In other words, later decisions on the "wait-and-see" items enable a much smaller number of used stacks when compared with strict robustness. This is because the strictly robust stacking solutions are conservative (in the sense that they have to satisfy the stacking constraints for all possible realizations of uncertain data). Thus, the number of possibilities for stacking an item on top of another is reduced quite a lot in comparison with adjustable robustness (where the practitioner can react for each realization of the uncertain data).

It can be also seen from Table 3.7 that in each experiment there is only a small difference between the gains (maximum and average) corresponding to the non-spread and spread stacking configurations of I^{fix}-items. Thus, the stacking configuration of the already stored items does not affect the gain much. However, the gain computed from Experiment 3 (with interval uncertaity and total order stacking constraints) is much smaller than the one from Experiments 1-2 (with finite uncertainty and arbitrary/transitive stacking constraints). This is because total order stacking constraints provide more possibilities for stacking items together in a stack than arbitrary/transitive stacking constraints. We therefore conclude that the type of uncertainty and the structure of the stacking constraints have a large impact on the gain.

We were also interested in evaluating the influence of the stacking height b on the gain. To this end, the problem instances with $b = 3$ were reassigned to $b = 5$. For the tests with finite uncertainty and transitive stacking constraints (Experiment 2), the average gain increases from 34.39% to 38.24% when stacking height b increases from 3 to 5. For the instances with interval uncertainty and total order stacking constraints

(Experiment 3), the average gain increases from 15.85% to 32.38%. Thus, increasing the stacking height b offers more possibilities for stacking wait-and-see items together with the other items in the stacks, which results in an increased gain for adjustable robust stacking solutions. Moreover, the average gain with total order stacking constraints is larger than with transitive stacking constraints because the former constraints provide more possibilities for stacking items together in stacks than the latter.

3.5 Conclusions

In this chapter we studied storage loading problems where a set of items must be loaded into a partly filled storage area, taking into account stacking constraints and the fact that another set of items arrives later (with uncertain data). To deal with data uncertainty, we applied two types of robustness: strict robustness, in which the stacking solution must be fixed in advance, and adjustable robustness, that allows the practitioner to react when the data of later arriving items become known.

The existence of adjustable robust stacking solutions to our considered problem requires feasibility over all scenarios for the uncertain items. This implies that in the case of interval data uncertainty, the adjustable robust counterpart involves an infinite number of scenarios. However, we proved that it is sufficient to focus on a single scenario of uncertain items, provided that the stacking constraints define a total order on all items. Thanks to this result, the effort to find adjustable robust stacking solutions can be significantly reduced.

We proposed different approaches to model the deterministic version of the storage loading problem as MIP formulations: one is based on three-index variables, one is based on network flows, while the last views each stack as a bin and the actual location of each item inside the bin is decided by exploiting transitivity of the stacking constraints. Then we derived different MIP formulations for the strictly and the adjustable robust problems under discrete and interval uncertainties with arbitrary, transitive, or total order stacking constraints.

Our experiments on randomly generated instances show that the performance of each formulation depends on the type of data uncertainty and the structure of the stacking constraints. On instances with discrete uncertainty and arbitrary stacking constraints the three-index approach has the best performance. On instances with discrete uncertainty and transitive stacking constraints (where the densities of the stacking matrices are small), the network flow approach performs best. The bin packing approach is the best choice for instances with interval uncertainty and total order stacking constraints (where the stacking matrices have a large density).

Furthermore, the experimental results show that relatively large gains can be achieved by adjustable robust stacking solutions in comparison with strictly robust ones.

Chapter 4

Security level of robust stacking solutions

In Section 2.2 we introduced the storage loading problem with interval uncertainty

$$(P_{\mathcal{I}}) \qquad L \mid I^{fix} \to I^1 \to \tilde{I}^2, s_{ij}(\tilde{a}), \mathcal{S}_{\mathcal{I}} \mid \#St.$$

This section also showed complexity results for some particular cases of the strictly and adjustable robust counterparts of $(P_{\mathcal{I}})$. Different MIP formulations for finding strictly and adjustable robust solutions to $(P_{\mathcal{I}})$ were proposed in Chapter 3. In the detailed computational study in the chapter, we compared the run-time of the formulations on randomly generated instances and provided guidelines which formulation to use in which setting. However, the influence of building different uncertainty sets $\mathcal{S}_{\mathcal{I}}$ on the outcomes of the robust stacking solutions to $(P_{\mathcal{I}})$ had not been investigated. This chapter, which is mainly based on our work in [32], tries to close this gap.

We propose a concept, so-called *security level*, as a key tool for our investigation. In Section 4.1 we explain how this concept is motivated from computing robust stacking solutions to $(P_{\mathcal{I}})$. In Section 4.2 we discuss different ways of deriving an interval uncertainty set $\mathcal{S}_{\mathcal{I}}$ from stochastic data of uncertain items. Through some numerical experiments in Section 4.3, we study the impact of different scenario sets $\mathcal{S}_{\mathcal{I}}$ on the trade-off between the optimal objective values and the security levels of the robust stacking solutions. Section 4.4 closes this chapter with some conclusions.

4.1 Security level of stacking solutions

To have a complete view about the motivation for the concept *security level*, we first recall from Section 2.2 the detail setup of $(P_{\mathcal{I}})$. For this problem, we are given a storage area arranged in m stacks $Q := \{1, \dots, m\}$ with fixed positions in a two-dimensional area, each stack contains b levels $L := \{1, \dots, b\}$ (i.e., at most b items can be stored in each stack). There are n items $I := \{1, \dots, n\}$ partitioned into three disjoint subsets: the set I^{fix} contains items that are already stored in the stacks, the set I^1 contains items that have to be stored in the storage area now, and the set I^2 contains items that will arrive later. We assume that relocations of I^{fix}-items are not allowed, and hence exclude from consideration all stacks which are already completely filled with such items (i.e., the corresponding items are removed from the sets I^{fix} and I). Due to the stacking sequence $I^{fix} \to I^1 \to I^2$, the loading of all I^1-items must be finished before storing any I^2-item.

That is, no item from $I^1 \cup I^2$ can be stored below an I^{fix}-item, and no I^2-item can be stacked below an I^1-item. Each item i has an associated value a_i which may refer to its weight, length, departure time, etc. Without loss of generality, we assume that a_i represents the weight (for example, in tons) of item i. The loading process of the items regards hard stacking constraints on weights $s_{ij}(a)$, i.e., item i is stackable directly on top of item j iff $a_i \leq a_j$. The uncertainty set $\mathcal{S_I}$ consists of all possible values of vector $a = (a_1, \ldots, a_n)$, in which the actual weights a_i of here-and-now items $i \in I^{fix} \cup I^1$ are known exactly, while the actual weight a_j of each wait-and-see item $j \in I^2$ is uncertain but varies in an interval $[a_j^{min}, a_j^{max}]$. It is assumed that the actual weights of I^{fix}-items adapt the given stacking configuration of these items.

An important issue is how to specify the interval uncertainty set $\mathcal{S_I}$. A natural way to define the intervals forming the uncertainty set is to derive them from stochastic data of the uncertain items. More precisely, throughout this chapter we assume that the weights a_i of wait-and-see items $i \in I^2$ are of statistical nature and mutually independent. Furthermore, for each item $i \in I^2$ the value a_i of its weight is assumed to follow some distribution function F_i. Having known the distribution of a_i, the decision maker can derive an interval $[a_i^{min}, a_i^{max}]$ so that the realization of a_i will belong to the interval with high probability. The intervals corresponding to the uncertain items $i \in I^2$ then constitute an instance of the uncertainty set $\mathcal{S_I}$.

Example 4.1. *Let X be a random variable having normal distribution with mean μ and standard deviation $\sigma > 0$. According to the $68 - 95 - 99.7$ rule in statistics (cf. [52]), we have*

$$
\begin{aligned}
P(X \in [\mu - \sigma,\ \mu + \sigma]) &\approx 0.6827,\\
P(X \in [\mu - 2\sigma,\ \mu + 2\sigma]) &\approx 0.9545,\\
P(X \in [\mu - 3\sigma,\ \mu + 3\sigma]) &\approx 0.9973.
\end{aligned}
$$

In other words, 68.27% observations of X lie within a band around the mean μ with a width of the standard deviation σ. If we extend the width of the band to two (three, resp.) times the standard deviation, then the obsevation of X will be in the band with a probability of 95.45% (99.73%, resp.).

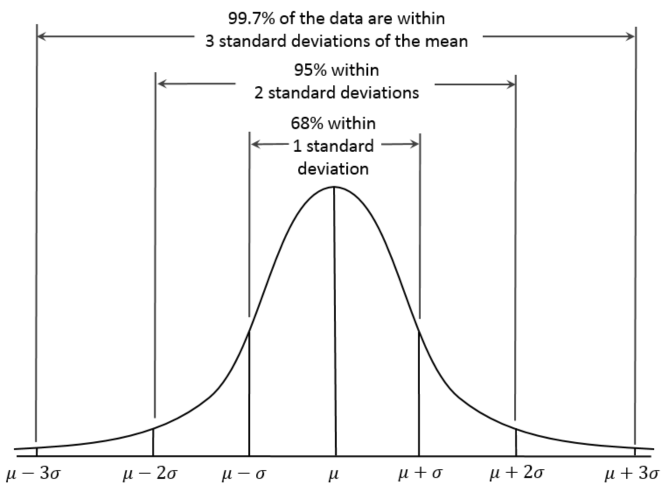


Figure 4.1: The $68 - 95 - 99.7$ rule for a normal distribution.
Source: https://en.wikipedia.org/wiki/68-95-99.7_rule.

Assuming that $\mu > 3\sigma$*, we have* $P(X \leq 0) < 0.135\%$*. We now assume that the weight* a_i *of an item* $i \in I^2$ *is the truncated version of the random variable* X *on* $[0, +\infty)$*, i.e.,*

$$a_i = \max\{0, X\}.$$

Then the interval $[\mu-\sigma, \mu+\sigma]$ *accounts for about* 68% *realizations of* a_i*, while the intervals* $[\mu - 2\sigma, \mu + 2\sigma]$ *and* $[\mu - 3\sigma, \mu + 3\sigma]$ *account for about* 95% *and* 99%*, respectively.*

Once $\mathcal{S_I}$ is specified, we can compute a robust stacking solution to $(P_\mathcal{I})$ with respect to the indicated uncertainty set. However, since we exclude from our consideration some unlikely realizations of the uncertain items (i.e., the values of $a_i (i \in I^2)$ that are not in the specified intervals $[a_i^{min}, a_i^{max}]$), the obtained robust stacking solution may be infeasible in some realizations of the items' weights. Therefore, the stacking result has a certain probability to be feasible when the wait-and-see items are realized. We call the probability value *security level* of the stacking solution (with respect to the specified uncertainty set $\mathcal{S_I}$).

To be precise, the security level $sl(x)$ of a stacking solution x to $(P_\mathcal{I})$ is defined as follows. Let $\overline{a}_i (i \in I^{fix} \cup I^1)$ be the actual weights of here-and-now items. For convenience, we denote $n_0 := |I^{fix}|$, $n_1 := |I^1|$, and $n_2 := |I^2|$. Let $\overline{w} \in \mathbb{R}^{n_0+n_1}$ be the vector whose components are actual weights of here-and-now items, i.e., $\overline{w}_i = \overline{a}_i$ for all $i \in I^{fix} \cup I^1$. Let $v \in \mathbb{R}^{n_2}$ be the weight vector of wait-and-see items, and $\mathcal{A}_2$ the set of all possible realizations of v. Then $\mathcal{A} := \{\overline{w}\} \times \mathcal{A}_2$ is the set of all possible values of weight vector a of all items in I. For the given stacking configuration x, let $\mathcal{A}(x)$ be the set of realizations $a \in \mathcal{A}$ such that x is a feasible solution (with respect to the stacking constraints on weights $s_{ij}(a)$). The security level $sl(x)$ is defined by

$$sl(x) = P(a \in \mathcal{A}(x) \mid a \in \mathcal{A}), \tag{4.1}$$

where $P(A \mid B)$ is the conditional probability of event A given the fact that event B already occurs (cf. [54], Chapter 3). In other words, $sl(x)$ is the probability that a weight realization a is part of $\mathcal{A}(x)$, or equivalently, the probability that x is a feasible stacking configuration (regarding the hard stacking constraints on weight $s_{ij}(a)$) when the uncertain items are realized. Note that by definition we always have $0 \leq sl(x) \leq 1$.

4.2 Deriving interval uncertainty sets

As discussed in the previous section, before calculating a robust solution to $(P_\mathcal{I})$, there are two key factors needed being determined. The first factor is the weight distribution of each wait-and-see item. The other is how to derive an appropriate uncertainty set $\mathcal{S_I}$ once we know how the items' weights are distributed. In this section we discuss these factors in detail.

4.2.1 Weight distributions

We first briefly review some probability distribution functions that are most suitable to represent a real-valued random variable as the weight of an item. The (truncated) normal, continuous uniform, and lognormal distributions are the most well-known and widely used statistical distributions. We refer the reader to [54] for a comprehensive study about distribution functions.

Normal and truncated normal distributions

The normal distribution has two parameters: the mean μ and the standard deviation $\sigma > 0$. If a random variable X has normal distribution, we write $X \sim \mathcal{N}(\mu, \sigma^2)$. Its probability density function is

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}.$$

Its mean and variance respectively are

$$E\{X\} = \mu, \tag{4.2}$$

$$\sigma_X^2 = \sigma^2. \tag{4.3}$$

Let $[\alpha, \beta] \subseteq \mathbb{R}_+$ be an interval. Then X conditional on $\alpha \le X \le \beta$ has a truncated normal distribution where the probability density function is given by

$$f_{[\alpha,\beta]}(t) = \begin{cases} \frac{\frac{1}{\sigma}\phi(\xi)}{\Phi(B)-\Phi(A)} & \text{if } \alpha \le t \le \beta, \\ 0 & \text{otherwise,} \end{cases}$$

with

$$\xi := \frac{t-\mu}{\sigma},\ A := \frac{\alpha-\mu}{\sigma},\ B := \frac{\beta-\mu}{\sigma},$$

$$\phi(\xi) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\xi^2},$$

$$\Phi(\xi) = \int_{-\infty}^{\xi} \phi(u) \mathrm{d}u.$$

The mean and the variance of the truncated normal distribution are computed by

$$E\{X \mid \alpha \le X \le \beta\} = \mu + \sigma \frac{\phi(A) - \phi(B)}{\Phi(B) - \Phi(A)},$$

$$\sigma^2_{\alpha \le X \le \beta} = \sigma^2 \left[1 + \frac{A\phi(A) - B\phi(B)}{\Phi(B) - \Phi(A)} - \left(\frac{\phi(A) - \phi(B)}{\Phi(B) - \Phi(A)} \right)^2 \right].$$

To model the weight of an item, the truncated distribution is more appropriate than the original normal distribution (since it excludes negative weight values). However, as shown in the above formulas, it is complicated to compute exactly the mean and variance of the truncated distribution. Therefore, when applying the truncated normal distribution to model the weight of an uncertain item, we impose $\mu > 3\sigma$ and take $[0, +\infty)$ as the truncated interval. By this setting, the truncated interval contains realizations of the item's weight with a probability of more than 99.7% (due to the $68 - 95 - 99.7$ rule, cf. Example 4.1). Then, for the sake of simplicity, we use (4.2)-(4.3) to estimate the mean and variance of the truncated distribution.

Continuous uniform distribution

This distribution has two parameters: the minimum value α and the maximum value $\beta > \alpha$. If X is a random variable with continuous uniform distribution, then we write $X \sim \mathcal{U}(\alpha, \beta)$. The probability density function of X is

$$f(t) = \begin{cases} \frac{1}{\beta-\alpha} & \text{for } t \in [\alpha, \beta], \\ 0 & \text{for } t \notin [\alpha, \beta]. \end{cases}$$

The mean and variance of X respectively are

$$E\{X\} = \frac{\alpha+\beta}{2},$$
$$\sigma_X^2 = \frac{1}{12}(\beta-\alpha)^2.$$

Lognormal distribution

The lognormal distribution is closely related to the normal distribution. The logarithm of a random variable with a lognormal distribution has a normal distribution. In the rest of this chapter, we will refer to this transformed distribution as the associated normal distribution.

The lognormal distribution has two parameters, that are the mean μ and the standard deviation $\sigma > 0$ of the associated normal distribution. If a random variable X has lognormal distribution, then we write $X \sim \log\mathcal{N}(\mu, \sigma)$. Its probability density function is

$$f(t) = \frac{1}{t\sigma\sqrt{2\pi}} e^{-\frac{(\log t-\mu)^2}{2\sigma^2}} \qquad \text{for } t > 0.$$

We can also write $X = e^{\mu+\sigma Z}$, where Z is a standard normal random variable (i.e., $Z \sim \mathcal{N}(0,1)$). The mean and variance of X respectively are

$$E\{X\} = e^{\mu+\frac{1}{2}\sigma^2},$$
$$\sigma_X^2 = e^{2\mu+\sigma^2}(e^{\sigma^2}-1).$$

4.2.2 Rules for deriving interval uncertainty sets

We now discuss some ways to derive an interval uncertainty set $\mathcal{S}_{\mathcal{I}}$ from a weight distribution of the uncertain items. By assumption, we already know the actual weights of the here-and-now items $I^{fix} \cup I^1$. For each wait-and-see item $i \in I^2$, the distribution of its weight a_i may be given by a probability density function f_i, or equivalently, the corresponding cumulative distribution function F_i defined by

$$F_i(t) = P(a_i \leq t) = \int_{-\infty}^{t} f_i(u)\mathrm{d}u.$$

We take into account this information in order to define the input parameters a_i^{min} and a_i^{max} for the interval uncertainty set $\mathcal{S}_{\mathcal{I}}$. Moreover, we construct different instances of $\mathcal{S}_{\mathcal{I}}$ to see how they affect the outcomes of robust stacking solutions to $(P_{\mathcal{I}})$. For the purpose of building different interval scenarios, we introduce the so-called *interval-control parameter* $\delta > 0$, and consider the following rules.

- Rule 1: for each item $i \in I^2$ we set $a_i^{min} = \overline{a}_i - \delta$ and $a_i^{max} = \overline{a}_i + \delta$, where $\overline{a}_i$ is the expected value of the weight of item i.
- Rule 2: for each item $i \in I^2$ we set $a_i^{min} = \overline{a}_i - \delta\sigma_i$ and $a_i^{max} = \overline{a}_i + \delta\sigma_i$, where $\overline{a}_i$ is the expected value and σ_i is the standard deviation of the weight of item i.
- Rule 3: for each item $i \in I^2$ we set a_i^{min} and a_i^{max} such that

$$P(a_i \leq a_i^{min}) = F_i(a_i^{min}) = \delta,$$

$$P(a_i \geq a_i^{max}) = 1 - F_i(a_i^{max}) = \delta,$$

where $F_i(t)$ is the cumulative distribution function of the weight of item i.

Each rule, together with a value of δ, determines an uncertainty set $\mathcal{S}_{\mathcal{I}}$. Note that for Rule 3, the interval-control parameter δ must be in $]0, 0.5[$ in order to have $a_i^{min} < a_i^{max}$. In all above rules, if $a_i^{min} \leq 0$ for some item i, then we reset a_i^{min} to 10^{-3}. This is because in reality the weight of an item must be always positive. Then, for some item i the interval $[a_i^{min}, a_i^{max}]$ may be not symmetric with respect to the expected weight $\overline{a}_i$.

4.3 Computational experiments

A stacking solution to $(P_{\mathcal{I}})$ can be evaluated by its cost (i.e., the number of used stacks) and its security level. In this section, we will figure out the impact of the parameters determining interval uncertainty set $\mathcal{S}_{\mathcal{I}}$ on the two outcome measures of strictly robust stacking solutions to $(P_{\mathcal{I}})$.

4.3.1 Computational setup

To set up problem instances for a computational study, we first specified values for the common height of stacks b and the cardinality of each set I^{fix}, I^1, and I^2. Then we randomly generated stacking configurations for I^{fix}-items by using the same method as proposed in Section 3.4.1. Afterward, we set values for actual weights of here-and-now items $I^{fix} \cup I^1$, and for parameters determining the weight distribution of wait-and-see items I^2. All uncertain items are assumed to have the same type of weight distribution. As discussed in Section 4.2.1, we consider three types of distributions.

Case of truncated normal distribution: We randomly generated values $\overline{a}_i > 0$ for actual weights of here-and-now items $i \in I^{fix} \cup I^1$ and expected weights $\overline{a}_i > 0$ of wait-and-see items $i \in I^2$. Then we randomly generated the standard deviations $\sigma_i > 0$ of the uncertain items $i \in I^2$. The actual weight a_i of each wait-and-see item $i \in I^2$ follows the truncated normal distribution $\mathcal{N}(\overline{a}_i, \sigma_i)$ on $[0, +\infty)$.

Case of continuous uniform distribution: Similar to the previous case, we randomly generated values $\overline{a}_i > 0$ for actual weights of here-and-now items and expected weights $\overline{a}_i > 0$ of wait-and-see items. For each uncertain item $i \in I^2$, its actual weight a_i is uniformly distributed on $[\overline{a}_i - \Delta_i, \overline{a}_i + \Delta_i]$, where Δ_i was randomly generated satisfying $0 < \Delta_i \leq \overline{a}_i$.

Case of lognormal distribution: For each item $i \in I^2$ we randomly generated values $\hat{a}_i > 0$ for the mean and $\hat{\sigma}_i > 0$ for the standard deviation of the associated normal distribution. Its actual weight a_i follows the lognormal distribution $\log\mathcal{N}(\hat{a}_i, \hat{\sigma}_i)$. Note that by this construction, each item $i \in I^2$ has the expected weight $\overline{a}_i = e^{\hat{a}_i + \frac{1}{2}\hat{\sigma}_i^2}$. Afterward, we randomly generated the actual weights $\overline{a}_i$ for here-and-now items $i \in I^{fix} \cup I^1$.

In all cases, the generated actual weights of I^{fix}-items adapt the generated stacking configuration of these items and the stacking constraints $s_{ij}(a)$. Moreover, we distinguish different setups for input parameters of the items. To do this, we apply the concept *coefficient of variation*. We recall from [40] the precise definition of this concept. Let X be a random variable whose mean value and standard deviation are respectively $\mu > 0$ and $\sigma > 0$. The coefficient of variation of X is defined as the ratio of the standard deviation to the mean:

$$c_v(X) = \frac{\sigma}{\mu}.$$

We differentiate two types of generated values for actual weights of here-and-now items and expected weights of wait-and-see items. All these values are integers either in $[10, 100]$ (*sparse type*) or in $[20, 40]$ (*dense type*). We also distinguish two types of generated values for the standard deviations of wait-and-see items. For the first type, the standard deviations of uncertain items were generated in such a way that $c_v(a_i) \in \left[\frac{1}{10}, \frac{1}{3}\right]$ for all $i \in I^2$. The second type is with smaller coefficients of variation: $c_v(a_i) \in \left[\frac{1}{20}, \frac{1}{6}\right]$ for all $i \in I^2$.

To complete setting up problem instances for our computational study, we set values for the interval-control parameter δ. This parameter is used to implement the rules of building interval uncertainty set $\mathcal{S}_\mathcal{I}$ (cf. Section 4.2.2). In order to apply Rule 3 we must have $\delta \in]0, 0.5[$. Therefore, for Rule 3 we varied δ from 0.01 to 0.49 with step size 0.01. For Rules 1 and 2, we first estimated some value δ^{max} for δ at which the stacking result has 100% of security level. The idea for the estimation is as follows. We start with an arbitrary positive value of δ and compute a robust stacking solution to $(P_\mathcal{I})$ corresponding to that δ-value. If the stacking solution has 100% of security level, then we reduce the current value of the interval-control parameter to its half, and repeat this process until obtaining a stacking solution with security level less than 100%. Otherwise, we double the current value of δ and continue the process until the corresponding stacking solution is 100% secured. Our strategy for finding δ^{max} is given precisely in Algorithm 2.

Algorithm 2 Estimating δ^{max}.

```
Require: Apply to Rule 1 or Rule 2.
 1: $\delta_1 := 1$.
 2: Compute a robust stacking solution $x_1$ to $(P_\mathcal{I})$ corresponding to $\delta_1$.
 3: Compute the security level $sl(x_1)$.
 4: repeat
 5:     if $sl(x_1) = 1$ then
 6:         $\delta_2 = \frac{1}{2}\delta_1$
 7:     else
 8:         $\delta_2 = 2\delta_1$
 9:     end if
10:     Compute a robust stacking solution $x_2$ to $(P_\mathcal{I})$ corresponding to $\delta_2$.
11:     Compute the security level $sl(x_2)$.
12:     if $sl(x_1) = 1$ and $sl(x_2) = 1$ then
13:         $\delta_1 := \delta_2$
14:     end if
15: until $(sl(x_1) < 1$ and $sl(x_2) = 1)$ or $(sl(x_1) = 1$ and $sl(x_2) < 1)$.
16: if $sl(x_1) = 1$ then
17:     $\delta^{max} := \delta_1$
18: else
19:     $\delta^{max} := \delta_2$
20: end if
21: return $\delta^{max}$.
```

Depending on the value of δ^{max}, we chose an appropriate step size for varying δ as follows.

- If $\delta^{max} \leq 1$, then we vary δ from 0.01 to δ^{max} with stepsize 0.01.
- If $1 < \delta^{max} \leq 10$, then we vary δ from 0.1 to δ^{max} with stepsize 0.1.
- If $\delta^{max} > 10$, then we vary δ from 0.5 to δ^{max} with stepsize 0.5.

Each value of δ, together with a rule proposed in Section 4.2.2, determines an instance of the uncertainty set $\mathcal{S}_{\mathcal{I}}$. We computed a strictly robust stacking solution x to $(P_{\mathcal{I}})$ given the generated interval uncertainty set $\mathcal{S}_{\mathcal{I}}$ by using the bin packing formulation $(Bin{-}srP_{\mathcal{I}})$ (cf. Section 3.2).

In general, it is difficult to compute exactly the security level of a stacking solution x. We therefore estimated it by sampling a set $\tilde{\mathcal{A}}$ of potential weight realizations according to the probability density functions f_i (or the cumulative distribution functions F_i), and compute the cardinality of set $\tilde{\mathcal{A}}(x)$ of realizations $a \in \tilde{\mathcal{A}}$ for which x is a feasible solution. Then the exact value of $sl(x)$ can be estimated by

$$\tilde{sl}(x) = \frac{|\tilde{\mathcal{A}}(x)|}{|\tilde{\mathcal{A}}|}.$$

Once the weight distributions of uncertain items were specified, we generated an instance of the sample set $\tilde{\mathcal{A}}$ with cardinality $|\tilde{\mathcal{A}}| = 10^6$. Each vector $a \in \tilde{\mathcal{A}}$ was generated by independently sampling a value for each of its components $a_i (i \in I^2)$. Then we used that sample set for computing the estimated security levels of the robust stacking solutions corresponding to different choices of rules (Rules 1-3) and different values of the interval-control parameter δ.

We used Visual Basic .NET for implementing our algorithm. To sample $\tilde{\mathcal{A}}$ according to given distribution functions, we used Extreme Optimization (cf. [1]) which contains Math and Statistic Libraries for .NET. We implemented MIP formulations using ZIMPL 3.3.2 (cf. [60]). As a MIP solver we used GUROBI 6.5.1 and set 30 minutes as time limit for the solver. All experiments were conducted on a computer with a Core 2 Duo 2 * 2.1 GHz processor and 2 GB of RAM.

For the discussion in the next subsection, we use the following terminologies to shortly describe main characteristics of our tested instances.

- A test instance is said to have a *sparse (dense, resp.) range of weights* if the actual weights of here-and-now items and the expected weights of wait-and-see items are of sparse (dense, resp.) type.
- A test instance has a *wide range of coefficients of variation* (or *wide Cv* for short) if $c_v(a_i) \in \left[\frac{1}{10}, \frac{1}{3}\right]$ for all $i \in I^2$. It is said to have a *thin range of coefficients of variation* (or *thin Cv* for short) if $c_v(a_i) \in \left[\frac{1}{20}, \frac{1}{6}\right]$ for all $i \in I^2$.
- A test instance is said to have *lognormal type of weight distribution* if the actual weights of wait-and-see items I^2 are lognormally distributed. Similar to the cases of truncated normal and uniform distributions.
- We say that a test instance has *Rule i* $(i = 1, 2, 3)$ if we followed Rule i in order to generate intervals constituting uncertainty set $\mathcal{S}_{\mathcal{I}}$.

4.3.2 Numerical results

In this subsection, by *stacking solutions* we refer to the strictly robust solutions to $(P_{\mathcal{I}})$. We organized our experiments to figure out the following points.

- How does the interval-control parameter δ influence the quality of the stacking solutions, once we know the weight distributions of the uncertain items and fix a rule for constructing the uncertainty set $\mathcal{S}_{\mathcal{I}}$?
- Does the range of expected weights of the uncertain items have any impact on the quality of the stacking solutions? Is the quality of the stacking solutions affected by the range of variation coefficients of the uncertain items?
- How robust are the stacking solutions with respect to the type of weight distributions of the uncertain items? That is, has guesing a wrong type of weight distribution influence on the security levels of the solutions or not?

In the following, we discuss the stated points respectively.

Influence of interval-control parameter δ

Firstly, we would like to have an impression about how the quality of the stacking solutions is affected by the interval-control parameter δ. For that purpose, we visualize results from a selected test instance with $|I^{fix}| = |I^1| = |I^2| = 20$, $b = 5$, dense range of weights, and wide Cv.

As the first representative, we tested the selected instance with an uniform weight distribution and different rules 1-3. For this test instance, Figure 4.2 shows the quality of different stacking solutions obtained by varying δ according to the ways specified in Section 4.3.1. The quality of a stacking solution x is measured by its cost $\#St(x)$ (i.e., the number of used stacks) and its estimated security level $\widetilde{sl}(x)$. For two stacking solutions that are computed by the same rule and have the same number of used stacks, the one having higher estimated security level is said to be better than the other. Fixing a rule, if different stacking solutions corresponding to different values of δ have the same number of used stacks, then only the best one (i.e. the one having highest security level) is recorded. We represent each recorded stacking solution by a small circle in the figure. The number besides each circle is the value of the interval-control parameter δ corresponding to the stacking solution. Each line color corresponds to a rule of deriving interval uncertainty sets. The dotted lines give us an impression about how much the estimated security level changes from a recorded stacking solution to the next one when fixing a rule and varying the interval-control paramter δ.

For the second and the third representatives, we did the same procedure as the first representative but with a lognormal distribution and a truncated normal distribution, respectively. The recorded stacking solutions from testing the second and the third representatives are respectively presented in Figures 4.3 and 4.4. In the legends of the figures, ‘LN’ (‘N’, ‘U’, resp.) is an abbreviation for lognormal (truncated normal, uniform, resp.) type of weight distribution, while ‘R1’ (‘R2’, ‘R3’, resp.) abbreviates Rule 1 (2, 3, resp.).

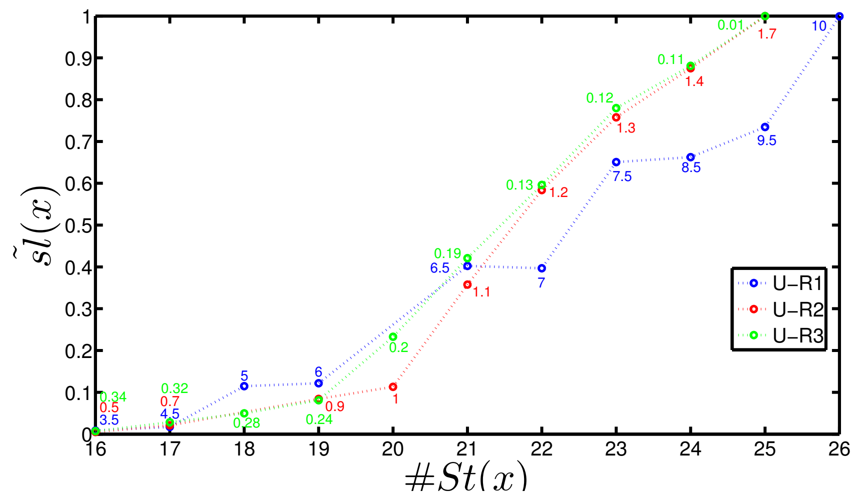


Figure 4.2: Visualization of the results from the first representative with uniform distribution and Rules 1-3.

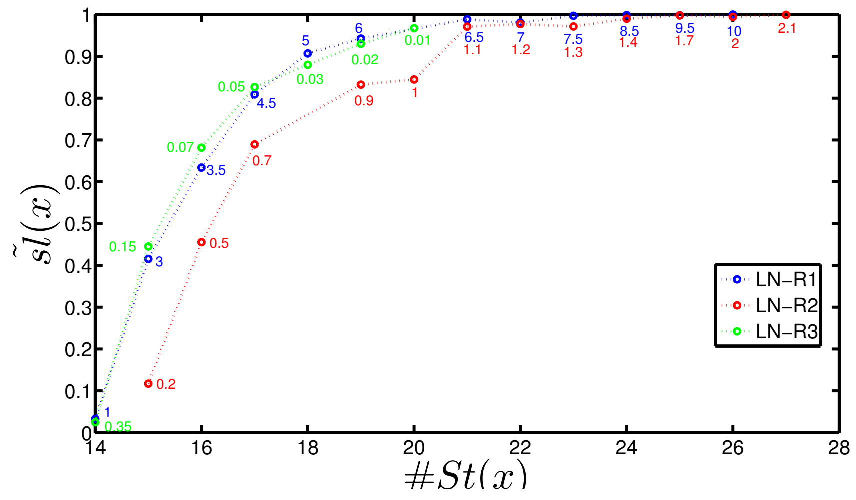


Figure 4.3: Visualization of the results from the second representative with lognormal distribution and Rules 1-3.

In the above tests, all MIP formulations corresponding to the specified values of δ were solved to optimality within the time limit. The following phenomenon can be seen from the tests. Using Rule 1 or Rule 2, when increasing the interval-control parameter δ, the obtained stacking solutions are more expensive (in the sense that they use more stacks) but higher secured. The same phenomenon occurs when decreasing δ in the cases that apply Rule 3. This phenomenon is due to the construction of scenario set $\mathcal{S}_{\mathcal{I}}$ in the rules. More precisely, when increasing δ in cases of Rules 1-2 and decreasing δ in case of Rule 3, we obtain larger specified intervals for scenarios of wait-and-see items. This reduces the possibility of certainly stacking a wait-and-see item on another item. Consequently, more stacks must be used to store the items, therefore the obtained stacking solutions become more costly. However, the more stacks that are used, the higher security level the solutions have. This kind of trade-off between the two quality measures of stacking solutions is because of the following reason. When more stacks are used, the average number per stack of wait-and-see items is reduced. As a consequence, the stacking configuration in each stack has more chances to be feasible in different data scenarios of the items.

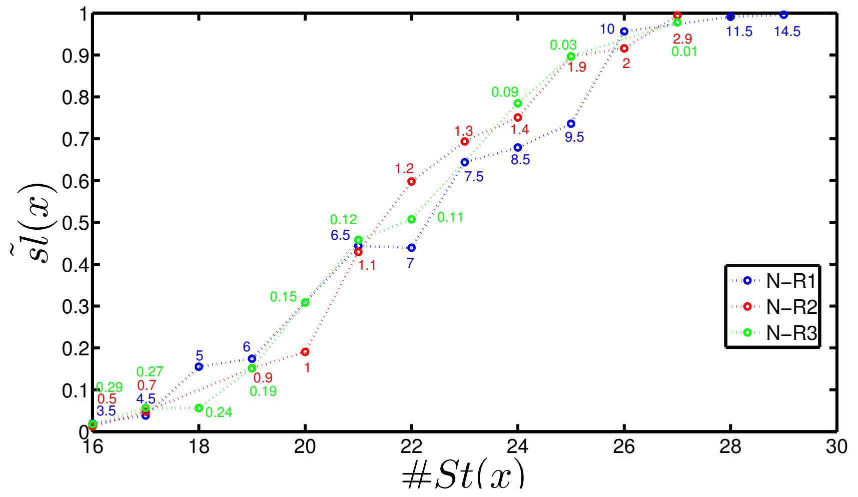


Figure 4.4: Visualization of the results from the third representative with truncated normal distribution and Rules 1-3.

It can be seen from the figures that there are few cases in which a stacking solution has slightly smaller security level than another one with less number of used stacks. This may due to the fact that in the construction of the interval uncertainty set $\mathcal{S}_{\mathcal{I}}$ we reset $a_{min} = 10^{-3}$ if $a_{min} \leq 0$ for some item i.

In term of the objective function $\#St$, the recorded stacking solutions corresponding to different rules have a similar range of cost values (for uniform and truncated normal types of weight distribution). It is different for the case of lognormal weight distribution (see Figure 4.3), in which the recorded stacking solutions cost from 14 to 20 stacks if Rule 3 is used, but they might cost up to 27 stacks if the other rules are used.

It is shown in the figures that, in most of the cases, the recorded stacking solutions of the same cost have a similar value of security level for different rules. A closer look at the numerical results shows that, if we change the rule of deriving interval uncertainty sets, we can find another stacking solution of the same cost with at most 20% in difference of security level for about 87% of the recorded stacking solutions. In the worst case, we may have to accept another stacking solution with 33% less secured if the rule is changed.

Knowing the impact of the interval-control parameter δ and fixing a rule in advance, one can find out appropriate values of this parameter to obtain a stacking solution with desired quality. Let us take the test with a truncated normal weight distribution and Rule 2 as an example (see Figure 4.4). If we would like to have a stacking solution with at least 90% of security level, then δ should be larger than 1.9 and the stacking solution should use at least 25 stacks. On the opposite direction, if we would like to have a stacking solution using at most 20 stacks, then δ should be smaller than 1.0 and such stacking solution can only be feasible in less than 20% possible realizations of wait-and-see items.

Impact of expected weights and coefficients of variation of uncertain items

For each uncertain item, the expected weight and coefficient of variation are important parameters determining the item's weight distribution. We now analyze the effect of these parameters on the quality of the stacking solutions. To do this analysis, we generated and tested different instances with $|I^{fix}| = |I^1| = |I^2| = 20$ and $b = 5$. We did two types of tests. The first type is to see how the range of weights influences the estimated security levels of stacking solutions. The second type is to understand the effect of different ranges of variation coefficients.

The detail setup for the first type is as follows. For each combination of a weight distribution with a rule, we first fixed a range of coefficients of variation. Then we generated 10 instances having sparse range of weights. Over these instances, we computed the average value $\overline{sl}_{sparse}(q)$ of estimated security levels of the recorded stacking solutions having the same number q of used stacks. We did the same procedure with 10 other instances having dense range of weights, and obtained the average values $\overline{sl}_{dense}(q)$ with similar spirit to $\overline{sl}_{sparse}(q)$. The left part of Table 4.1 reports the maximum, minimum, and average values of the difference $\overline{sl}_{sparse}(q) - \overline{sl}_{dense}(q)$ over possible solution costs q. We excluded the cases of solution cost q in which both $\overline{sl}_{sparse}(q)$ and $\overline{sl}_{dense}(q)$ equal 0 or 1.

Similarly, for the second type of tests, with each combination of a weight distribution and a rule, we fixed a range of weights and generated 10 thin-Cv instances as well as 10 wide-Cv instances. By the same way of computing $\overline{sl}_{sparse}(q)$, we obtained average (estimated) security levels $\overline{sl}_{thin}(q)$ for the thin-Cv instances and $\overline{sl}_{wide}(q)$ for the wide-Cv instances. After excluding the values of q where both $\overline{sl}_{thin}(q)$ and $\overline{sl}_{wide}(q)$ equal 0 or 1, we computed the differences $\overline{sl}_{thin}(q) - \overline{sl}_{wide}(q)$ for the remaining values of q. The maximum, minimum, and average of these differences are reported in the right part of Table 4.1.

Distribution	Sparse - Dense			Thin - Wide		
and Rule	Max	Min	Ave	Max	Min	Ave
Lognormal & Rule 1	0.672	-0.029	0.063	0.329	0.001	0.058
Lognormal & Rule 2	0.651	-0.005	0.122	0.389	0.001	0.086
Lognormal & Rule 3	0.687	-0.002	0.138	0.356	0.005	0.106
Truncated normal & Rule 1	0.222	-0.137	0.036	0.558	0.001	0.286
Truncated normal & Rule 2	0.250	-0.022	0.077	0.625	0.001	0.266
Truncated normal & Rule 3	0.267	-0.001	0.085	0.594	0.001	0.312
Uniform & Rule 1	0.256	-0.180	0.038	0.662	0.001	0.359
Uniform & Rule 2	0.288	-0.086	0.089	0.815	0.001	0.364
Uniform & Rule 3	0.320	-0.068	0.098	0.811	0.001	0.362

Table 4.1: Compare security levels of stacking solutions from tests with different ranges of weights and coefficients of variation.

The 'Min' columns of Table 4.1 show that different settings of the ranges of weights and variation coefficients may only have a small influence on the security levels of some stacking solutions. That should be the cases when stacking solutions with small cost are used (and therefore they have very small value of security level), or when we use stacking solutions with high cost and highly secured.

The 'Max' columns of Table 4.1 say that the ranges of weights and variation coefficients heavily affect the security level of the stacking solutions in the worst case, where the solutions have medium number of used stacks. In this case, if the uncertain items have a setting of sparse range of weights and thin Cv, the stacking solutions are more secured than in other settings.

In relation with the 'Max' columns, the 'Ave' columns of Table 4.1 give us an impression about the number of stacking solutions that are 'robustly secured' with respect to different settings of the ranges of weights and variation coefficients. The small values in the left 'Ave' column in the table mean a large number of stacking solutions robustly secured when changing the range of weights. The large values in the right 'Ave' column in the table mean that there are not so many stacking solutions robustly secured when changing the range of coefficients of variation.

Robustness of security level with respect to weight distribution

Does guesing a wrong type of weight distribution have any influence on the security levels of the stacking solutions? To answer this question, we organized our experiments as follows. We generated a set of different test instances with $|I^{fix}| = |I^1| = |I^2| = 20$ and $b = 5$. The input data for each test instance include a weight vector $\bar{a}$ (whose components are actual weights of here-and-now items and expected weights of wait-and-see items) and a standard deviation vector σ_a (whose components are standard deviations of actual weights of I^2-items). For each choice of rule, we first generated 20 such instances with the same weight vector $\bar{a}$ and the same standard deviation vector σ_a. Then we tested these instances for different types of weight distribution. Afterward, we computed the average over the 20 generated instances of estimated security levels of the recorded stacking solutions having the same number of used stacks.

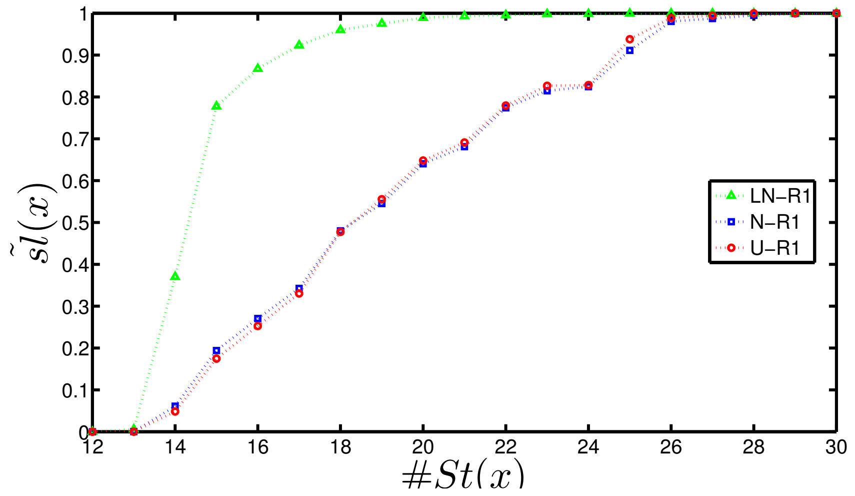


Figure 4.5: Average quality of stacking solutions over 20 test instances with Rule 1.

Figures 4.5-4.7 visualize the average quality of recorded stacking solutions over the test instances using Rules 1-3, respectively. In the legends of the figures, 'LN' ('N', 'U', resp.) abbreviates lognormal (truncated normal, uniform, resp.) type of weight distribution, while 'R1' ('R2', 'R3', resp.) abbreviates Rule 1 (2, 3, resp.). It is clear from the figures that the stacking solutions with lognormal type of distribution differ much from the ones with the other types. That is, if lognormal is a wrong type of weight distribution, then we may have to change the obtained stacking solution to another one with much less security level.

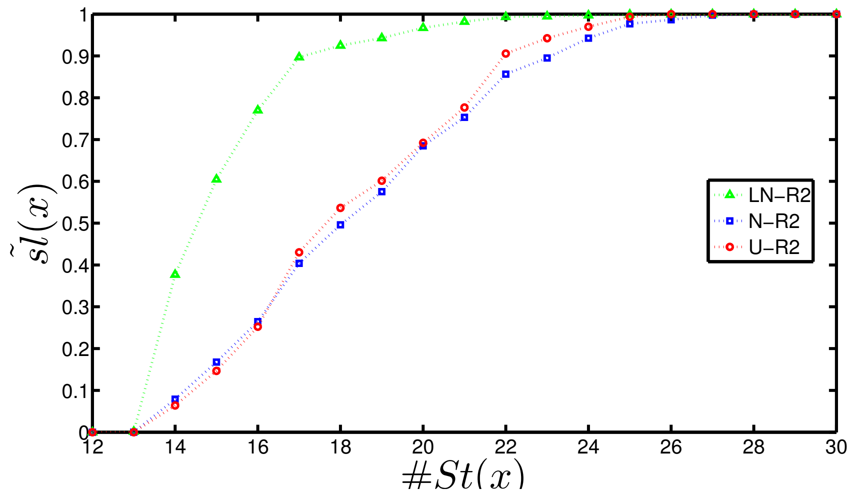


Figure 4.6: Average quality of stacking solutions over 20 test instances with Rule 2.

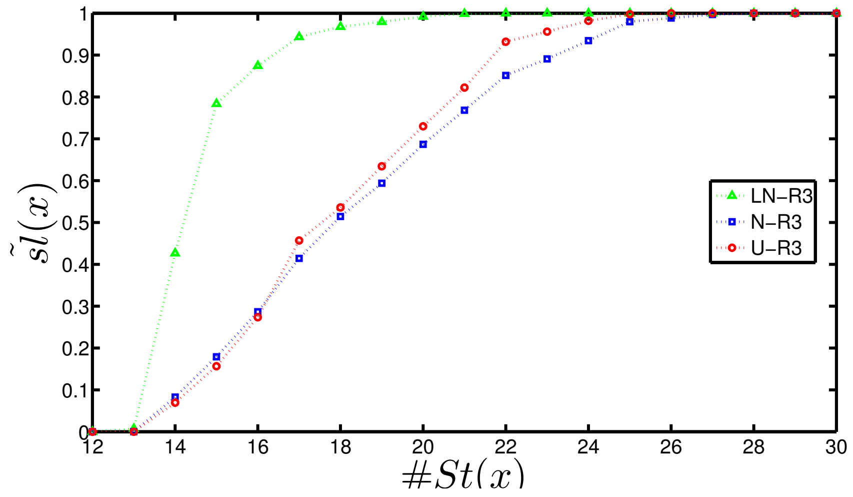


Figure 4.7: Average quality of stacking solutions over 20 test instances with Rule 3.

On the positive side, Figures 4.5-4.7 show that the stacking solutions with truncated normal distribution have similar quality to the ones with uniform distribution. In other words, if we predicted that uncertain items follow an uniform weight distribution but their realizations are truncated normally distributed (or vice versa), then we can change the computed stacking solution to another one with similar security level. In fact, in this circumstance, a closer look at the numerical results shows that the maximum change in value of security level is just 8.04 %.

4.4 Conclusions

In this chapter we considered a storage loading problem in which a set of items has to be loaded into a partly filled storage area, regarding stacking constraints on an associated parameter of each item, and taking into account stochastic data uncertainty of items arriving later. To obtain a solution to this problem in a robust sense, we derive from the items' stochastic data an interval uncertainty set. The idea behind is that the data realization of uncertain items will belong to the specified set with a certain probability. Given the derived uncertainty set, we can compute a robust stacking solution in the strict sense.

To evaluate the obtained robust stacking solution, apart from its cost (i.e. the number of used stacks), we proposed a concept so-called security level. Given a stacking solution, its security level is defined by the probability that the stacking solution is feasible when the uncertain items are realized.

To better understand the impact of the specified uncertainty sets on the quality of the robust stacking results, we considered different types of data distribution as well as various rules for deriving interval scenario sets. In a computational study on randomly generated instances, we analyzed the trade-off between security level and cost of the robust stacking solutions. Furthermore, we studied the influence of the interval-control paramter, the impact of parameters determining the data distribution, and robustness of the security level with respect to the type of data distribution.

Further research could focus on analyzing the impact of cardinality of each set $|I^{fix}|$, $|I^1|$, $|I^2|$. It would be interesting to apply the concept of adjustable robustness and compare the quality (i.e. security level and cost) of adjustable robust stacking solutions with strict ones.

Chapter 5

Storage loading with payload constraints

In this chapter 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.

This chapter is mainly based on our work in [49], and it is organized as follows. Section 5.1 is devoted to the motivation of introducing payload constraints. In Section 5.2, we describe the storage loading problem (with stacking and payload constraints) in the deterministic setting, 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 5.3, while adjustable counterparts are discussed in Section 5.4. Computational experiments are presented in Section 5.5. We close this chapter with some conclusions in Section 5.6.

5.1 Motivation of payload constraints

In many practical loading problems, apart from stacking constraints, additional stability issues are crucial. In load planning of trains (cf. [29, 30]) containers have to be loaded onto wagons so that the stability of each wagon is guaranteed. In one-dimensional balanced loading problems (cf. [7, 67]), 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. [81]), 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 [25] 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.

When loading containers onto a container ship, the stability of the ship is also an important issue that needs to be taken into account. It follows from a physical principle (see [53], 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 is, the more stable the ship is. Naturally, it is desirable to store the items onto the ship in such a way that achieves the lowest possible position of the gravity center of the loaded ship. The following lemma gives an impression how such a stacking configuration should be, in which by "hard stacking constraints on weights" we mean that heavier items must be put below lighter ones.

Lemma 5.1. *Given a number of items of possitive weights. Assume that the items have a common height and the weight of each item is uniformly distributed over the item's volume. Among all configurations of stacking all these items into a single stack, any stacking configuration satisfying the hard stacking constraints on weights has the lowest position of the gravity center of the loaded stack.*

Proof. Assume that we have n items of common height $h > 0$ and store them from level 1 to level n of the stack. Since the weight of each item is uniformly distributed over the item's volume, 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 5.1). Let G be the gravity center of the whole stack, and w_i the weight of the item stored at level i. According to [53] Chapter 12, the height of G is computed by

$$h_G = \frac{\sum_{i=1}^{n} w_i h_i}{\sum_{i=1}^{n} w_i}. \tag{5.1}$$

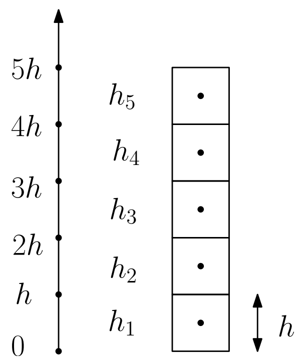


Figure 5.1: Stack with 5 items.

Let c be an arbitrary stacking configuration of the items in the stack. If in the configuration c there exist two items i_1 and i_2 with $w_{i_1} \geq w_{i_2}$ and $h_{i_1} > h_{i_2}$ (i.e., item i_1 is heavier and at a higher level than item i_2), then we swap these two items and keep the positions of all other items. Denote the new stacking configuration by c', and let $h_G(c)$ (resp., $h_G(c')$) be the height of gravity center of the stack in configuration c (resp., configuration c'). By applying (5.1) we get

$$\begin{aligned} h_G(c) - h_G(c') &= \frac{1}{\sum_{i=1}^{n} w_i} \left(w_{i_1} h_{i_1} + w_{i_2} h_{i_2} - w_{i_1} h_{i_2} - w_{i_2} h_{i_1} \right) \\ &= \frac{1}{\sum_{i=1}^{n} w_i} \left(w_{i_1} - w_{i_2} \right) \left(h_{i_1} - h_{i_2} \right) \\ &\geq 0. \end{aligned}$$

Thus, $h_G(c') \leq h_G(c)$, i.e., by swapping a heavier item in a higher level with a lighter item in a lower level we obtain a new configuration with lower position of G. By repeating this procedure, the lowest position of G is attained in stacking configurations where heavier items are put below lighter ones. □

It follows from Lemma 5.1 that to obtain the best stability 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. For example, such stacking constraints appear in the context of the master bay plan problem (MBPP) (cf. [5, 6, 73]). 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 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 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 gravity center 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, instead of imposing the hard stacking constraints on weights, a simple approach is to require that the total weight of the containers allocated in a stack is limited by a given bound. For example, this approach is applied in [35, 36] 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. [2, 3, 4, 58, 70, 71]). 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 sub-sections, then the containers are stored into these sub-sections in such a way that the gravity center of the ship is within a limited area.

In this chapter, we propose another approach to tackle the mentioned drawbacks of hard stacking constraints on weights and to control the stability of a ship. Our approach is to impose 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 is limited by aw_i, where a is a given positive parameter which may also depend on the stacks. The following lemma shows how the gravity center of a stack depends on the payload parameter a, in which we again assume that the items have a common height $h > 0$ and the weight of each item is uniformly distributed over the item's volume.

Lemma 5.2. *Consider a configuration of stacking n items into a single stack. Assume that the weights of the items can vary but always satisfy the payload constraints with payload parameter $a > 0$. Let $\bar{h}_G$ be the height of the highest possible position of gravity center G of the stack. Then $\bar{h}_G(a)$ is a monotonically increasing function.*

Proof. Let w_i (resp., h_i) be the weight (resp., the height of the gravity center) of the item stored at level i of the stack. As shown in proof of Lemma 5.1 we have $h_i = \left(i - \frac{1}{2}\right) h$. Due to the payload constraints we always have

$$\sum_{i=0}^{k-1} w_{n-i} \leq aw_{n-k} \qquad (k = 1, \ldots, n-1). \tag{5.2}$$

Regarding (5.2), it follows from (5.1) 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

$$\sum_{i=1}^{n} w_i = Aw_n,$$

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 formula (5.1) we have

$$\begin{aligned}\bar{h}_G &= \frac{\sum_{i=1}^{n} w_i h_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n} (i-\frac{1}{2})hw_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n} ihw_i}{\sum_{i=1}^{n} w_i} - \frac{1}{2}h \\ &= \frac{\sum_{i=1}^{n-1} ih\frac{(a+1)^{n-i-1}}{a^{n-i}}w_n + nhw_n}{\left(\frac{a+1}{a}\right)^{n-1} w_n} - \frac{1}{2}h \\ &= \sum_{i=1}^{n-1} i\frac{1}{a}\left(\frac{a}{a+1}\right)^i h + n\left(\frac{a}{a+1}\right)^{n-1} h - \frac{1}{2}h. \end{aligned} \tag{5.3}$$

If we set $u := \frac{a}{a+1}$, then $0 < u < 1$ and $a = \frac{u}{1-u}$. Moreover, we can rewrite (5.3) as follows:

$$\begin{aligned}\bar{h}_G &= \sum_{i=1}^{n-1} i\frac{1-u}{u}u^i h + nu^{n-1}h - \frac{1}{2}h \\ &= \sum_{i=1}^{n-1} i(1-u)u^{i-1}h + nu^{n-1}h - \frac{1}{2}h \\ &= \sum_{i=1}^{n-1} i(u^{i-1} - u^i)h + nu^{n-1}h - \frac{1}{2}h \\ &= \left(\frac{1}{2} + \sum_{i=1}^{n-1} u^i\right)h. \end{aligned} \tag{5.4}$$

It is an immediate consequence of (5.4) 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. □

As a consequence of Lemma 5.2, 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 the gravity center of the stack, 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. Our approach using payload constraints takes advantage of using knowledge of the actual weights of the items, rather than binning the items into groups and assigning to each item the average weight of its group.

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. [85]). 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.

In this chapter we consider storage loading problems where a set of incoming items has to be assigned to stacks in a way that satisfies hard stacking constraints and soft payload constraints, and minimizes the total payload violation. As we have already introduced, stacking constraints are often given in form of a stacking matrix to describe binary relations on items (i.e., which items can be stacked onto each other according to practical requirements such as departure times or incompatible dimensions of items), while payload constraints limit the payload that can be put on each item for stability reason. Since in real-world applications, often not all data are known exactly during the planning stage, we consider the storage loading problems under data uncertainty. More precisely, 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 (where the location of each item needs to be fixed before its actual weight becomes known), and adjustable robustness (where each item must only be assigned to a stack, but its position within the stack can be decided once the weight is known).

5.2 Problem formulation

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

5.2.1 Nominal problem

In the following, we describe the nominal (deterministic) problem in more detail. We are given a storage area consisting of m stacks, where each stack can hold at most b items and the position of each stack in the storage area is fixed. The set of all items to be stored is denoted by $I = \{1, 2, \dots, n\}$ 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, due to stacking constraints on departure times or lengths of items). 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 or may have an arbitrary structure. For the storage loading problems considered in this chapter, 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 $(i_k, \dots, i_1)$, where i_l denotes the item stacked at level l and $l = 1$ corresponds to the ground level. Such a tuple is feasible with respect to stacking matrix (s_{ij}) if $k \leq b$ and $s_{i_{l+1}, i_l} = 1$ for all $l = 1, \dots, k-1$.

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 chapter, the payload constraints are assumed to be soft, i.e., payload violations are allowed, but the total payload violation has 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. Section 1.1.5) 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). It was shown in Theorem 2.11 that deciding whether a feasible solution exists is strongly NP-complete for $b \geq 3$ 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 chapter 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.

5.2.2 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. We denote by $Q := \{1, \ldots, m\}$ the set of stacks and $L := \{1, \ldots, b\}$ 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.

$$\min \sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v_{ql} \tag{5.5}$$

$$\text{s.t.} \sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 \qquad \forall i \in I \tag{5.6}$$

$$\sum_{i \in I} x_{iql} \leq 1 \qquad \forall q \in Q, l \in L \tag{5.7}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x_{jq,l-1} \geq x_{iql} \qquad \forall i \in I, q \in Q, l \in L \setminus \{1\} \tag{5.8}$$

$$\sum_{j \in I} \sum_{h=l+1}^{b} w_j x_{jqh} - a \sum_{i \in I} w_i x_{iql} \leq v_{ql} \qquad \forall q \in Q, l \in L \setminus \{b\} \tag{5.9}$$

$$x_{iql} \in \{0,1\} \qquad \forall i \in I, q \in Q, l \in L \tag{5.10}$$

$$v_{ql} \geq 0 \qquad \forall q \in Q, l \in L \setminus \{b\} \tag{5.11}$$

According to (5.5) the sum of all payload violations is minimized. Constraints (5.6) guarantee that all items are stored. Constraints (5.7) ensure that at most one item is stored at each level of each stack. Due to (5.8) the stacking constraints s_{ij} are satisfied and no item is placed to a location where no item is stacked below. Inequalities (5.9) ensure that the payload violations v_{ql} are computed correctly.

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

$$\mathcal{X} := \left\{ x \in \{0,1\}^{|I| \times |Q| \times |L|} \mid x \text{ satisfies (5.6)-(5.8)} \right\}$$

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

5.2.3 Uncertainties

We consider two kinds of uncertainties for the item weights that affect the payload constraints:

- *Interval uncertainty:* 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

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

  to denote an interval-based uncertainty set. An element $w \in \mathcal{S}_{\mathcal{I}}$ 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 $\mathcal{S}_{\mathcal{I}}$.

- *Finite uncertainty:* 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

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

  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 $\mathcal{S}$ to denote the uncertainty set. Note that $\mathcal{S}_{\mathcal{F}}$ is a finite set, while $\mathcal{S}_{\mathcal{I}}$ contains infinitely many possible outcomes. This leads to different solution approaches for the robust models we consider in this paper.

5.3 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 strict robustness approach (cf. Section 1.2.1), we focus on strictly robust solutions where the worst-case payload violation over all scenarios is minimized. The strictly robust counterpart $(SR, \mathcal{S})$ of the optimization storage loading problem (P_a) under affection of uncertainty set $\mathcal{S}$ is

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

We first consider the case of finite uncertainty in Section 5.3.1, afterwards the more elaborate case of interval-based uncertainty is discussed in Section 5.3.2.

5.3.1 Finite uncertainty

In the following, we modify the problem formulation (P_a) 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 $(SR, \mathcal{S}_\mathcal{F})$ when the usage of uncertainty set $\mathcal{S}_\mathcal{F}$ should be emphasized, and obtain the following MIP formulation:

$$(SRF) \qquad \min v \tag{5.12}$$

$$\sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 \qquad \forall i \in I \tag{5.13}$$

$$\sum_{i \in I} x_{iql} \leq 1 \qquad \forall q \in Q, l \in L \tag{5.14}$$

$$\sum_{j \in I \setminus \{i\}} s_{ij} x_{jq,l-1} \geq x_{iql} \qquad \forall i \in I, q \in Q, l \in L \setminus \{1\} \tag{5.15}$$

$$\sum_{j \in I} \sum_{h=l+1}^{b} w_j^k x_{jqh} - a \sum_{i \in I} w_i^k x_{iql} \leq v_{ql}^k \qquad \forall q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} \tag{5.16}$$

$$\sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v_{ql}^k \leq v \qquad \forall k \in \mathcal{N} \tag{5.17}$$

$$x_{iql} \in \{0, 1\} \qquad \forall i \in I, q \in Q, l \in L \tag{5.18}$$

$$v_{ql}^k \geq 0 \qquad \forall q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} \tag{5.19}$$

$$v \geq 0 \tag{5.20}$$

The objective (5.12) is to minimize the largest total payload violation over all scenarios. As in the nominal problem, constraints (5.13) ensure that every item is stored, constraints (5.14) model that at most one item is assigned to every location, and constraints (5.15) take care of the stacking constraints s_{ij}. Furthermore, according to constraints (5.16) the payload violations v_{ql}^k are correctly computed for every scenario k.

Finally, constraints (5.17) (together with the minimization in (5.12)) 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.

5.3.2 Interval uncertainty

We now consider $(SR, \mathcal{S}_{\mathcal{I}})$, i.e., the strictly robust model with interval-based uncertainty sets $\mathcal{S}_{\mathcal{I}}$. We denote this problem as (SRI) for short. Due to the continuous nature of the uncertainty set $\mathcal{S}_{\mathcal{I}}$, there are infinitely many scenarios and we cannot include all these scenarios into a MIP formulation as we have done with (SRF). We propose two approaches to tackle this issue. The first approach iteratively choose scenarios from $\mathcal{S}_{\mathcal{I}}$ to include them in a finite uncertainty set, until the optimal stacking solution corresponding to that finite uncertainty set is also the optimal solution to (SRI). The second approach points out a characterization of worst-case scenarios in $\mathcal{S}_{\mathcal{I}}$, that can be used to formulate a compact model (MIP formulation) for (SRI).

An iterative algorithm

We now discuss the first approach in detail. Similar ideas to this approach have been also successfully applied in [26, 68, 84]. The general procedure is shown in Algorithm 3. We start with an arbitrary scenario $w^1 \in \mathcal{S}_{\mathcal{I}}$. In each iteration k, we find a best stacking configuration x^k with respect to the current (finite) set of scenarios $\mathcal{S}^k$, i.e., x^k is a solution to $(SR, \mathcal{S}^k)$. Then we determine a worst-case scenario $w^{k+1} \in \mathcal{S}_{\mathcal{I}}$ 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 3 Exact algorithm for (SRI).

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

Theorem 5.3. *Algorithm 3 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.,

$$f^* = \min_{x \in \mathcal{X}} \max_{w \in \mathcal{S}_{\mathcal{I}}} f(x, w). \tag{5.21}$$

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

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

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

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

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 $w^{k+1} \notin \mathcal{S}^k$, so that the algorithm never enters a cyclic loop. Indeed, assume to the contrary that $w^{k+1} \in \mathcal{S}^k$. As defined in Step 6 of the algorithm, we have

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

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

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 5.3.1. We now discuss how Step 6 can be realized. Given a stacking configuration $x \in \mathcal{X}$, we need to find a vector $w \in \mathcal{S}_\mathcal{I}$ 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, i.e., we need to find

$$\max_{w\in\mathcal{S}_\mathcal{I}} f(x,w) = \max_{w\in\mathcal{S}_\mathcal{I}} \sum_{q\in Q} v_q(x,w) = \max_{w\in\mathcal{S}_\mathcal{I}} \sum_{q\in Q(x)} v_q(x,w), \tag{5.22}$$

where $Q(x)$ is the set of stacks (in the given stacking configuration x) containing at least two items, and $v_q(x,w)$ is the total payload violation of stack q in scenario $w \in \mathcal{S}_\mathcal{I}$. Note that the stacking configuration x is fixed, therefore we have

$$\max_{w\in\mathcal{S}_\mathcal{I}} \sum_{q\in Q(x)} v_q(x,w) = \sum_{q\in Q(x)} \max_{w\in\mathcal{S}_\mathcal{I}} v_q(x,w). \tag{5.23}$$

In turn, we consider an arbitrary stack $q \in Q(x)$ in the given configuration x and focus on the problem of finding item weights w that maximize the total payload violation in stack q, i.e.

$$(V_q) \qquad \max_{w \in \mathcal{S}_{\mathcal{I}}} v_q(x,w).$$

Let $I(q)$ be the set of items contained in stack q and $L(q) := \{1, \ldots, |I(q)|\}$. Since $q \in Q(x)$, we have $|I(q)| \geq 2$. For the sake of simplicity, we denote the weight of the item at level $l \in L(q)$ by $w_{[l]}$.

Lemma 5.4. *The total payload violation of stack q in the stacking configuration $x \in \mathcal{X}$, given by*

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

is a convex function with respect to w.

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. □

Due to the convexity of $v_q(x,\cdot)$, there is always an optimal solution to (V_q) where each item weight is at its lower or upper bound. We can make use of this fact to formulate a binary linear programming formulation for (V_q). To do this, for all $l \in L(q)$, we introduce continuous variables $w_{[l]}$ determining the weight of the item at level l in stack 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, (V_q) can be formulated as follows.

$$(V_q) \quad \max \sum_{l \in L(q)} \left(\sum_{h>l} w_{[h]} - a w_{[l]} \right) \alpha_l \tag{5.24}$$

$$\text{s.t.} \quad w_{[l]} = \underline{w}_{[l]} + (\overline{w}_{[l]} - \underline{w}_{[l]}) \beta_l \qquad \forall l \in L(q) \tag{5.25}$$

$$\alpha_l, \beta_l \in \{0,1\} \qquad \forall l \in L(q) \tag{5.26}$$

$$w_{[l]} \geq 0 \qquad \forall l \in L(q) \tag{5.27}$$

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 (5.25) in the objective function, and get the equivalent model

$$\max \quad \sum_{l \in L(q)} \sum_{h>l} \left(\underline{w}_{[h]} + \left(\overline{w}_{[h]} - \underline{w}_{[h]} \right) \beta_h \right) \alpha_l - \sum_{l \in L(q)} a \left(\underline{w}_{[l]} + \left(\overline{w}_{[l]} - \underline{w}_{[l]} \right) \beta_l \right) \alpha_l \tag{5.28}$$

$$\text{s.t.} \quad \alpha_l, \beta_l \in \{0,1\} \qquad \forall l \in L(q) \tag{5.29}$$

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

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

$$+ \sum_{l \in L(q)} \left(\sum_{h>l} (\overline{w}_{[h]} - \underline{w}_{[h]})\gamma_{lh} - a(\overline{w}_{[l]} - \underline{w}_{[l]})\gamma_{ll} \right) \tag{5.30}$$

$$\text{s.t.} \quad \alpha_l + \beta_h - 1 \leq \gamma_{lh} \leq \frac{1}{2}(\alpha_l + \beta_h) \qquad \forall l, h \in L(q) \tag{5.31}$$

$$\alpha_l, \beta_l \in \{0,1\} \qquad \forall l \in L(q) \tag{5.32}$$

$$\gamma_{lh} \in \{0,1\} \qquad \forall l, h \in L(q) \tag{5.33}$$

The objective function (5.30) is the same as in (5.28) after substituting and reordering terms. The additional constraints (5.31) are used to ensure that γ_{lh} is 1 if and only if both α_l and β_h are 1. Solving problem (5.30)-(5.33) independently for each stack $q \in Q(x)$ hence gives the desired solution to Step 6 of Algorithm 3.

A compact MIP formulation

For the second approach to solve (SRI), we start by revising the problem (V_q), i.e. the problem of finding item weights $w \in \mathcal{S}_\mathcal{I}$ that maximize the total payload violation in stack $q \in Q(x)$. Recall from Lemma 5.4 that the objective function of (V_q) is convex with respect to w. Note that maximizing a convex function over a convex domain in general is an NP-hard problem (cf. [18]). However, we can show that for the case we are considering an efficient solution algorithm exists.

Theorem 5.5. *For any value of the payload parameter a, the problem (V_q) can be solved by evaluating $\mathcal{O}(|I(q)|^{2\delta-1})$ scenarios of $w \in \mathcal{S}_\mathcal{I}$, where $\delta := \min\{\lceil a \rceil, \lfloor \frac{|I(q)|}{2} \rfloor\}$.*

Proof. As mentioned above, due to the convexity of $v_q(x, \cdot)$, to find an optimal solution to (V_q) it is sufficient to consider only scenarios where the weights of all items in $I(q)$ are at 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 the left part of Figure 5.2. 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.

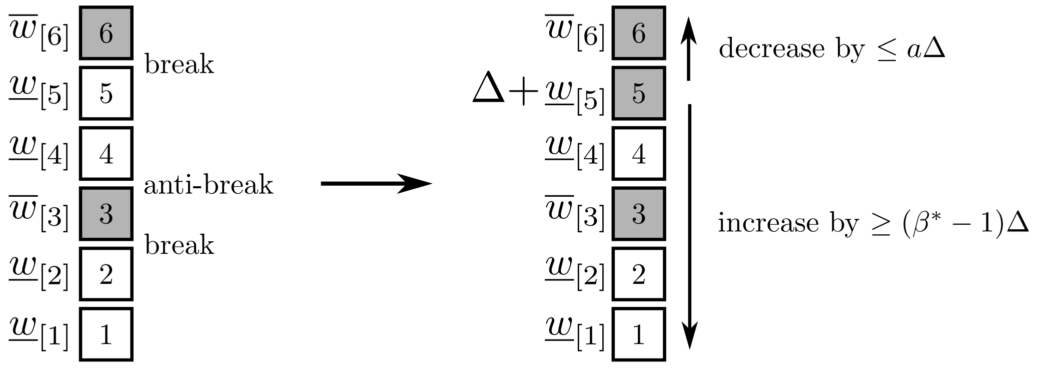


Figure 5.2: Illustration of the proof for Theorem 5.5.

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_q), 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 $v_q(x, w)$.

We now show that there is an optimal solution to (V_q) 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_q) with at most $\lceil a \rceil$ breaks. Indeed, let w^* be an optimal solution to (V_q) 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 $\Delta = \overline{w}_{[l^*]} - \underline{w}_{[l^*]}$, then the item at level l^* has the weight $\overline{w}_{[l^*]}$ and the break at level l^* therefore disappears. Moreover, the payload violation at level l^* decreases by at most $a\Delta$. However, there are at least $\beta^* - 1$ breaks beneath level l^*. As discussed above, these $\beta^* - 1$ breaks correspond to $\beta^* - 1$ more payload violations, which result in an increase of $v_q(x, w)$ by at least $(\beta^* - 1)\Delta$ (see the right part of Figure 5.2). 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 $v_q(x, w)$. 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 weight scenarios of items in stack q with $\beta \in \{1, \ldots, \delta\}$ 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 $\beta = 1, 2, \ldots, \delta$ gives $\mathcal{O}(|I(q)|^{2\delta-1})$ possible scenarios that have to be tested. □

Note that if the payload parameter 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_q) can be solved in polynomial time.

Thanks to Theorem 5.5, Step 6 of Algorithm 3 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 5.3.1.

We present this compact formulation of (SRI) for the case $a \leq 1$. Firstly, following the above discussion, for solving (V_q) it is sufficient to consider $|I(q)| - 1$ scenarios of weights of items in stack q, where in scenario k only one break occurs at level k ($k = 1, \ldots, |I(q)| - 1$) and the weights of all items in $I(q)$ are at their lower or upper bounds. Secondly, it is known from the proof of Theorem 5.5 that there is always an optimal solution to (V_q) where the bottom item is as light as possible and the top item is as heavy as possible. Note that the objective of (V_q) is to maximize the total payload violation in stack q. Therefore for finding an optimal solution to (V_q) we can restrict our consideration to scenarios in which all items up to a level $k \leq |I(q)| - 1$ of stack q have their lightest possible weights, and all items at higher levels have their heaviest possible weights. Furthermore, it follows from (5.21)-(5.23) that the optimal objective value of (SRI) can be computed as follows.

$$\begin{aligned} f^* &= \min_{x\in\mathcal{X}} \max_{w\in\mathcal{S}_\mathcal{I}} f(x,w) \\ &= \min_{x\in\mathcal{X}} \max_{w\in\mathcal{S}_\mathcal{I}} \sum_{q\in Q} v_q(x,w) \\ &= \min_{x\in\mathcal{X}} \left(\sum_{q\in Q} \max_{w\in\mathcal{S}_\mathcal{I}} v_q(x,w) \right). \end{aligned}$$

To the end, for the compact formulation of (SRI) in case $a \leq 1$, we introduce auxiliary variables $w^k_{ql} \geq 0$ denoting the weight of the item in stack q at level l in the stacking solution to the scenario in which stack q has a unique break at level k. Modifying the formulation (SRF), we get the following MIP formulation for (SRI):

$$\min \sum_{q\in Q} v_q \tag{5.34}$$

$$\text{s.t.} \quad (5.6) - (5.8)$$

$$w^k_{ql} = \sum_{i\in I} \underline{w}_i x_{iql} \qquad \forall q \in Q, k, l \in L, l \leq k \tag{5.35}$$

$$w^k_{ql} = \sum_{i\in I} \overline{w}_i x_{iql} \qquad \forall q \in Q, k, l \in L, l > k \tag{5.36}$$

$$\sum_{h>l} w^k_{qh} - a w^k_{ql} \leq v^k_{ql} \qquad \forall q \in Q, k, l \in L \setminus \{b\} \tag{5.37}$$

$$\sum_{l\in L\setminus\{b\}} v^k_{ql} \leq v_q \qquad \forall q \in Q, k \in L\setminus\{b\} \tag{5.38}$$

$$x_{iql} \in \{0,1\} \qquad \forall i \in I, q \in Q, l \in L \tag{5.39}$$

$$v^k_{ql} \geq 0 \qquad \forall q \in Q, k, l \in L \setminus \{b\} \tag{5.40}$$

$$w^k_{ql} \geq 0 \qquad \forall q \in Q, k, l \in L \tag{5.41}$$

$$v_q \geq 0 \qquad \forall q \in Q \tag{5.42}$$

Constraints (5.6)-(5.8) determine a feasible stacking solution x with respect to stacking constraints (s_{ij}). In scenario k of weights of items in stack q of such a stacking solution x, all items up to level k are assumed to be as light as possible (constraints (5.35)), while all items at higher levels are as heavy as possible (constraints (5.36)). The payload violation in the solution for scenario k for stack q and level l is measured by the variable v^k_{ql} in constraints (5.37). The worst-case over all scenarios is computed with the help of the variables v_q and constraints (5.38).

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. In the following we present a compact formulation for (SRI) when $1 < a \leq 2$. For this formulation, new variables $w^{k_1k_2k_3}_{ql}$ represent the weight of the item in stack q at level l when there are two breaks at levels k_1 and k_3, as well as an anti-break at level k_2. Additionally, variables $v^{k_1k_2k_3}_{ql}$ are used to measure the payload violation in this scenario. We denote $L^* := \{(k_1, k_2, k_3) \in (L \setminus \{b\}) \times (L \setminus \{b\}) \times (L \setminus \{b\}) : k_1 < k_2 < k_3\}$.

$$\min \sum_{q\in Q} v_q \tag{5.43}$$

s.t. (5.6) – (5.8)

$$w_{ql}^k = \begin{cases} \sum_{i\in I} \underline{w}_i x_{iql} & \text{if } l \le k \\ \sum_{i\in I} \overline{w}_i x_{iql} & \text{if } k < l \end{cases} \qquad \forall q \in Q, l \in L, k \in L \setminus \{b\} \tag{5.44}$$

$$v_{ql}^k \ge \sum_{h>l} w_{qh}^k - a w_{ql}^k \qquad \forall q \in Q, k, l \in L \setminus \{b\} \tag{5.45}$$

$$v_q \ge \sum_{l\in L\setminus\{b\}} v_{ql}^k \qquad \forall q \in Q, k \in L \setminus \{b\} \tag{5.46}$$

$$w_{ql}^{k_1k_2k_3} = \begin{cases} \sum_{i\in I} \underline{w}_i x_{iql} & \text{if } l \le k_1 \\ \sum_{i\in I} \overline{w}_i x_{iql} & \text{if } k_1 < l \le k_2 \\ \sum_{i\in I} \underline{w}_i x_{iql} & \text{if } k_2 < l \le k_3 \\ \sum_{i\in I} \overline{w}_i x_{iql} & \text{if } k_3 < l \end{cases} \qquad \forall q \in Q, l \in L, (k_1,k_2,k_3) \in L^* \tag{5.47}$$

$$v_{ql}^{k_1k_2k_3} \ge \sum_{h>l} w_{qh}^{k_1k_2k_3} - a w_{ql}^{k_1k_2k_3} \qquad \forall q \in Q, l \in L, (k_1,k_2,k_3) \in L^* \tag{5.48}$$

$$v_q \ge \sum_{l\in L\setminus\{b\}} v_{ql}^{k_1k_2k_3} \qquad \forall q \in Q, (k_1,k_2,k_3) \in L^* \tag{5.49}$$

$$x_{iql} \in \{0,1\} \qquad \forall i \in I, q \in Q, l \in L \tag{5.50}$$

$$v_{ql}^k \ge 0 \qquad \forall q \in Q, l, k \in L \setminus \{b\} \tag{5.51}$$

$$v_{ql}^{k_1k_2k_3} \ge 0 \qquad \forall q \in Q, l, (k_1,k_2,k_3) \in L^* \tag{5.52}$$

$$w_{ql}^k \ge 0 \qquad \forall q \in Q, l \in L, k \in L \setminus \{b\} \tag{5.53}$$

$$w_{ql}^{k_1k_2k_3} \ge 0 \qquad \forall q \in Q, l \in L, (k_1,k_2,k_3) \in L^* \tag{5.54}$$

$$v \ge 0 \tag{5.55}$$

Similar to the previous MIP formulation, constraints (5.6)-(5.8) determine a feasible stacking solution x with respect to stacking constraints (s_{ij}). Constraints (5.44)-(5.46) compute the worst-case total payload violation for stack q over all scenarios in which this stack has exactly one break. Constraints (5.47)-(5.49) compute the worst-case total payload violation for stack q over all scenarios in which this stack has exactly two breaks with one anti-break in between.

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

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

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 $w \in \mathcal{S}_{\mathcal{I}}$ 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 $w \in \mathcal{S}_\mathcal{I}$ which dominates all other scenarios from $\mathcal{S}_\mathcal{I}$ with respect to this constraint. Hence, this robust payload constraint can be equivalently written as

$$\sum_{j\in I}\sum_{h=l+1}^{b} \overline{w}_j x_{jqh} - a\sum_{i\in I} \underline{w}_i x_{iql} \le 0 \qquad \forall q \in Q, l \in L \setminus \{b\} \tag{5.56}$$

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 (5.5)-(5.8),(5.56),(5.10).

5.4 Adjustable robustness

Following the adjustable robustness approach (cf. Section 1.2.1), 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 destinations of the items). As another example we refer to the following setting from [79]: 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.

Let $\mathcal{Z}$ be the set of here-and-now decisions, i.e., each element $z \in \mathcal{Z}$ is a partition of the set of items I into stacks. For each stack in such a partition z one has to make sure that there exists at least one ordering of the stack's items (i.e., an assignment of the items to levels) satisfying stacking constraints s_{ij}. Let $\mathcal{P}_z$ be the set of such orderings from all stacks in $z \in \mathcal{Z}$. Then each pair (z, π) with $\pi \in \mathcal{Z}$ defines a complete stacking solution of all items in I. Let $g(z, \pi, w)$ be the total payload violation of the stacking solution (z, π) given the weight vector w of all items. Then the adjustable robust counterpart of $(AR, \mathcal{S})$ of the optimization storage loading problem (P_a) under affection of uncertainty set $\mathcal{S}$ is

$$(AR, \mathcal{S}) \qquad \min_{z\in\mathcal{Z}} \max_{w\in\mathcal{S}} \min_{\pi\in\mathcal{P}_z} g(z, \pi, w).$$

In Section 5.4.1, we present a MIP formulation for the adjustable robust counterpart in case of finite uncertainty sets. For the case of interval uncertainty sets we give an exact algorithm in Section 5.4.2 and propose some heuristic algorithms in Section 5.4.3.

5.4.1 Finite uncertainty

In this subsection we consider the adjustable robust problem with finite uncertainty $(AR, \mathcal{S}_{\mathcal{F}})$, shortly denoted as (ARF). As in Section 5.3, 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.

$$
\begin{aligned}
(ARF) \qquad & \min \; v && && (5.57)\\
\text{s.t.} \qquad & \sum_{q \in Q} z_{iq} = 1 && \forall\, i \in I && (5.58)\\
& \sum_{i \in I} z_{iq} \le b && \forall\, q \in Q && (5.59)\\
& \sum_{i \in I} x^k_{iql} \le 1 && \forall\, q \in Q, l \in L, k \in \mathcal{N} && (5.60)\\
& \sum_{l \in L} x^k_{iql} = z_{iq} && \forall\, q \in Q, i \in I, k \in \mathcal{N} && (5.61)\\
& \sum_{j \in I \setminus \{i\}} s_{ij} x^k_{jq,l-1} \ge x^k_{iql} && \forall\, i \in I, q \in Q, l \in L \setminus \{1\}, k \in \mathcal{N} && (5.62)\\
& \sum_{j \in I} \sum_{h=l+1}^{b} w^k_j x^k_{jqh} - a \sum_{i \in I} w^k_i x^k_{iql} \le v^k_{ql} && \forall\, q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} && (5.63)\\
& \sum_{q \in Q} \sum_{l \in L \setminus \{b\}} v^k_{ql} \le v && \forall\, k \in \mathcal{N} && (5.64)\\
& z_{iq} \in \{0,1\} && \forall\, i \in I, q \in Q && (5.65)\\
& x^k_{iql} \in \{0,1\} && \forall\, i \in I, q \in Q, l \in L, k \in \mathcal{N} && (5.66)\\
& v^k_{ql} \ge 0 && \forall\, q \in Q, l \in L \setminus \{b\}, k \in \mathcal{N} && (5.67)\\
& v \ge 0 && && (5.68)
\end{aligned}
$$

Constraints (5.58) model that every item has to be assigned to some stack, while every stack contains at most b items (constraints (5.59)). Constraints (5.60) 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^k_{iql} in constraints (5.61): 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. Note that constraints (5.58) and (5.61) together ensure that for each scenario every item has to be assigned to exactly one stack and level in the corresponding solution. Constraints (5.62)-(5.64) 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.

5.4.2 Interval uncertainty

We now consider the adjustable robust problem with interval uncertainty $(AR, \mathcal{S}_{\mathcal{I}})$, shortly denoted as (ARI). We follow the idea of the iterative algorithm proposed in Section 5.3.2. More precisely, we iteratively choose scenarios from $\mathcal{S}_{\mathcal{I}}$ to include in a finite uncertainty set. In each iteration, we solve the relaxed problem of (ARI) corresponding to the finite uncertainty set, and determine a scenario of item weights maximizing the total payload violation for the solution of the relaxed problem. This worst-case scenario is added to the current finite uncertainty set. We terminate the process at the iteration in which the optimal objective value of the relaxed problem equals the worst-case total payload violation.

The subproblem of generating worst-case scenarios now becomes more complex, since the worst-case scenario generation also needs to take the possible wait-and-see decisions into account. Recall that the here-and-now decisions assign each item to some stack, while the wait-and-see decisions determine the ordering of items in each stack (when the weights of all items are known). We apply the iterative approach to solve the subproblem of generating worst-case scenarios. Roughly speaking, this approach concerns solving the following 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.

More precisely, we consider a fixed stack $q \in Q$ and let $I(q)$ be the set containing all items which are assigned to q according to the here-and-now decisions z_{iq}. Furthermore, let $L(q) := \{1, \dots, |I(q)|\}$ 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 satisfying stacking constraints s_{ij}. 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 $\mathcal{P}' \subseteq \mathcal{P}(I_q)$ of permutations for the items in stack q we search for a worst-case weight scenario $w \in \mathcal{S}_{\mathcal{I}}$ that is a solution to

$$(AV1, I(q), \mathcal{P}') \qquad \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}'} v_q(\pi, w).$$

The second subproblem 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 to

$$(AV2, I(q), w) \qquad \min_{\pi \in \mathcal{P}(I_q)} v_q(\pi, w).$$

The weight vector computed from the first subproblem $(AV1, I(q), \mathcal{P}')$ is used as input for the second subproblem $(AV2, I(q), w)$. The ordering of items in $I(q)$ resulting from the latter subproblem is then added to the set $\mathcal{P}'$. These two subproblems are iteratively 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 4.

Theorem 5.6. *Algorithm 4 terminates after a finite number of iterations and yields an optimal solution x to (ARI).*

Algorithm 4 Exact algorithm for (ARI).

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

Proof. We first prove that the inner loop from Step 11 to Step 18 indeed yields one of the worst-case scenarios w (corresponding to stack q) that is used for the outer loop (from Step 3 to Step 23). More precisely, we need to show that after executing the inner loop we will have

$$v_q^\ell(w^\ell) = v_q^* := \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}(I(q))} v_q(\pi, w).$$

Indeed, on one hand, since $\mathcal{P}^\ell \subseteq \mathcal{P}(I(q))$ for all ℓ we have

$$\overline{v}_q(\mathcal{P}^\ell) = \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}^\ell} v_q(\pi, w) \geq \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}(I(q))} v_q(\pi, w) = v_q^*.$$

On the other hand, it is obvious that for all ℓ we have

$$v_q^\ell(w^\ell) = \min_{\pi \in \mathcal{P}(I(q))} v_q(\pi, w^\ell) \leq \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}(I(q))} v_q(\pi, w) = v_q^*.$$

Therefore, once $\overline{v}_q(\mathcal{P}^\ell) = v_q^\ell(w^\ell)$, it is straightforward that $v_q^\ell(w^\ell) = v_q^*$. In turn, we need to show that the inner loop terminates after a finite number of iterations $\ell \in \mathbb{N}$. Indeed, since

$|I(q)|$ is finite, so is $|\mathcal{P}(I(q))|$. Note that by construction we have $\mathcal{P}^{\ell} \subseteq \mathcal{P}^{\ell+1} \subseteq \mathcal{P}(I(q))$, so there must be some iteration ℓ^* at which we have $\mathcal{P}^{\ell^*} = \mathcal{P}^{\ell^*+1}$, or in other words, $\pi^{\ell^*} \in \mathcal{P}^{\ell^*}$. Then we have

$$\begin{aligned}
v_q^{\ell^*}(w^{\ell^*}) &= v_q(\pi^{\ell^*}, w^{\ell^*}) \\
&= \min_{\pi \in \mathcal{P}(I(q))} v_q(\pi, w^{\ell^*}) && \text{(by definition of } \pi^{\ell^*}) \\
&= \min_{\pi \in \mathcal{P}^{\ell^*}} v_q(\pi, w^{\ell^*}) && \text{(since } \pi^{\ell^*} \in \mathcal{P}^{\ell^*} \subseteq \mathcal{P}(I(q))) \\
&= \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}^{\ell^*}} v_q(\pi, w) && \text{(by definition of } w^{\ell^*}) \\
&= \bar{v}_q(\mathcal{P}^{\ell^*}).
\end{aligned}$$

We have shown that there exists ℓ^* such that $v_q^{\ell^*}(w^{\ell^*}) = \bar{v}_q(\mathcal{P}^{\ell^*})$. That means the inner loop terminates after iteration ℓ^*, and w^{ℓ^*} is the worst-case weight scenario we need to plug in the outer loop. In turn, by a similar proof to Theorem 5.3, it follows that the outer loop terminates after a finite number of iterations and the stacking solution generated in the last iteration is an optimal solution to (ARI). □

How to solve Step 5 of the algorithm was shown in Section 5.4.1. We now discuss how subproblems in Step 13 and Step 15 can be solved. We consider the first subproblem

$$(AV1, I(q), \mathcal{P}^{\ell}) \qquad \max_{w \in \mathcal{S}_{\mathcal{I}}} \min_{\pi \in \mathcal{P}^{\ell}} v_q(\pi, w).$$

We assume that the set $\mathcal{P}^{\ell}$ contains K permutations $\pi^1, \dots, \pi^K$ 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 $\mathcal{K} := \{1, \dots, K\}$. 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 $(AV1, I(q), \mathcal{P}^{\ell})$ can be formulated as follows.

$$\max \ v \tag{5.69}$$

$$\text{s.t.} \sum_{l \in L'(q)} v_l^k \geq v \qquad \forall k \in \mathcal{K} \tag{5.70}$$

$$\left(\sum_{h > l} \sum_{i \in I(q)} p_{ih}^k w_i - a \sum_{i \in I(q)} p_{il}^k w_i \right) \alpha_l^k = v_l^k \qquad \forall k \in \mathcal{K}, l \in L'(q) \tag{5.71}$$

$$\underline{w}_i \leq w_i \leq \overline{w}_i \qquad \forall i \in I(q) \tag{5.72}$$

$$\alpha_l^k \in \{0, 1\} \qquad \forall k \in \mathcal{K}, l \in L'(q) \tag{5.73}$$

$$v_l^k \geq 0 \qquad \forall k \in \mathcal{K}, l \in L'(q) \tag{5.74}$$

$$v \geq 0. \tag{5.75}$$

According to (5.69) the total payload violation of the stack is maximized. Constraints (5.70) ensure that v equals the smallest payload violation over all item permutations. Due to (5.71), the payload violations are correctly computed. Finally, (5.72) guarantees that the weight variables w_i are contained in the given intervals.

Note that constraints (5.71) 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 MIP formulation for $(AV1, I(q), \mathcal{P}^\ell)$.

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

We now consider the second subproblem

$$(AV2, I(q), w) \qquad \min_{\pi \in \mathcal{P}(I_q)} v_q(\pi, w),$$

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. 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 $(AV2, I(q), w)$ can be formulated by the following MIP.

$$\min \sum_{l \in L'(q)} v_l \tag{5.76}$$

$$\text{s.t.} \quad \sum_{\substack{h \in L(q) \\ h>l}} \sum_{j \in I(q)} w_j p_{jh} - \sum_{i \in I(q)} a w_i p_{il} \leq v_l \qquad \forall l \in L'(q) \tag{5.77}$$

$$\sum_{i \in I(q)} p_{il} = 1 \qquad \forall l \in L(q) \tag{5.78}$$

$$\sum_{l \in L(q)} p_{il} = 1 \qquad \forall i \in I(q) \tag{5.79}$$

$$p_{i,l+1} + p_{jl} \leq 1 \qquad \forall i, j \in I(q) \text{ with } s_{ij} = 0, l \in L'(q) \tag{5.80}$$

$$p_{il} \in \{0,1\} \quad \forall i \in I(q), l \in L(q) \tag{5.81}$$

$$v_l \geq 0 \qquad \forall l \in L'(q). \tag{5.82}$$

Due to (5.76) the total payload violation is minimized. Constraints (5.77) are used to determine the violations v_l at the different levels. The assignment constraints (5.78) and (5.79) ensure that at each level exactly one item is stored and that each item is assigned to exactly one level, respectively. Due to constraints (5.80) the item permutation is feasible with respect to the stacking constraints s_{ij}.

5.4.3 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.

Pattern generation

For our first approach, we consider an extended problem formulation for (ARI). Let $\mathcal{C}$ be the collection of 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). Each set in the collection $\mathcal{C}$ is called a *pattern.* For each pattern $C \in \mathcal{C}$ we introduce a binary variable y_C to decide whether the pattern C is used in the optimal stacking solution ($y_C = 1$) or not ($y_C = 0$), and let

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

be the worst-case payload violation of the pattern C. The value of $v(C)$ can be determined using $(AV1, C, \mathcal{P}(C))$ and $(AV2, C, w)$ iteratively (as discussed in the previous section). We use binary variables χ_j^C to indicate whether $j \in I$ is contained in C with $\chi_j^C = 1$ if and only if $j \in C$, and $\chi_j^C = 0$ otherwise. Then, we get the following equivalent formulation $(ARI - PG)$ for (ARI).

$$(ARI - PG) \qquad \min \sum_{C \in \mathcal{C}} v(C) y_C \tag{5.83}$$

$$\text{s.t.} \sum_{C \in \mathcal{C}} \chi_j^C y_C = 1 \qquad \forall j \in I \tag{5.84}$$

$$\sum_{C \in \mathcal{C}} y_C \leq m \tag{5.85}$$

$$y_C \in \{0, 1\} \qquad \forall C \in \mathcal{C}. \tag{5.86}$$

Constraints (5.84) ensure that every item is contained in one pattern, and constraint (5.85) bounds the number of available patterns by the number of available stacks in the storage area.

As there are potentially exponentially many relevant patterns $C \in \mathcal{C}$, we use the following algorithm to solve $(ARI - PG)$ heuristically. We keep a working collection $\mathcal{C}^*$ of patterns, which is initially filled with the patterns that are given by some feasible starting solution. In every iteration, we generate a collection $\bar{\mathcal{C}}$ 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. The collection $\mathcal{C}'$ of candidate patterns in the current iteration consists of

- patterns in collection $\bar{\mathcal{C}}$,
- patterns from the current working collection $\mathcal{C}^*$,
- patterns that are generated by randomly merging patterns from the current working collection $\mathcal{C}^*$.

Patterns that have once been part of the working collection are never removed. Hence, we generate new patterns in the fashion of a genetic algorithm. Then, we solve $(ARI - PG)$

using the heuristic collection $\mathcal{C}'$ instead of the full collection $\mathcal{C}$. Patterns of the resulting solution that are not part of the working collection yet are added to $\mathcal{C}^*$. We then begin with the next algorithm iteration by generating new patterns $\bar{\mathcal{C}}$.

Note that problems of type $(ARI - PG)$ are easy to solve with current commercial MIP solvers (constraints (5.84) are special ordered set constraints, and constraint (5.85) 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 4 within the same amount of time. However, no quality bounds (or even a proof of optimality) are produced.

Local search

In our second approach, we follow an iterative improvement local search heuristic. Given a feasible assignment of items to stacks (with respect to stacking constraints s_{ij}), we consider the following two moves: (i) an item is moved from one stack to another stack with sufficient capacity, or (ii) two items from two stacks are swapped. A move is called feasible if it results in a feasible assignment of items to stacks. We then iteratively use the subproblems of types $(AV1)$ and $(AV2)$ to calculate the worst-case total payload violation of the new item-stack assignment resulting by each feasible move. All feasible moves are evaluated in random order until an improving move (i.e. a move that results in an item-stack assignment having smaller worst-case total payload violation) is found. The first found improving move is then performed. The process is repeated until either a local optimum or some time limit is reached.

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 of items to stacks (with respect to stacking constraints s_{ij}), we randomly select one stack with a high objective value (i.e. the total payload violation) 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 4. 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 4 once a gap of 3% is reached or 10 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 not only evaluate an assignment of items to two stacks, but also find a best possible assignment.

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 $x_{ij} \in \{0, \ldots, b\}$ 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:

$$\sum_{i \in I} y_i \leq m \tag{5.87}$$

$$\sum_{\substack{j \in I \\ s_{ji}=1}} x_{ji} - \sum_{\substack{j \in I \\ s_{ij}=1}} x_{ij} + b y_i \geq 1 \qquad \forall\, i \in I \tag{5.88}$$

$$x_{ij} \leq b z_{ij} \qquad \forall i, j \in I \tag{5.89}$$

$$\sum_{\substack{j \in I \\ s_{ij}=1}} z_{ij} \leq 1 \qquad \forall i \in I \tag{5.90}$$

$$\sum_{\substack{j \in I \\ s_{ji}=1}} z_{ji} \leq 1 \qquad \forall i \in I \tag{5.91}$$

$$x_{ij} \in \{0, \ldots, b\} \qquad \forall i, j \in I : s_{ij} = 1 \tag{5.92}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in I : s_{ij} = 1 \tag{5.93}$$

$$y_i \in \{0, 1\} \qquad \forall i \in I. \tag{5.94}$$

Constraints (5.87) ensure that at most m stacks can be used, while constraints (5.88) model the integer flow. Constraints (5.89) are used to determine whether an arc is used or not, while constraints (5.90) and (5.91) 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.

5.5 Computational experiments

To test the performance of the models and algorithms introduced in this chapter, 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.

5.5.1 Small instances

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 $\mathcal{S}_{\mathcal{I}}$ or a finite uncertainty set $\mathcal{S}_{\mathcal{F}}$.

Setup

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).

For interval uncertainty sets, 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, at first for each item i we generated a lower bound $\underline{w}_i$ and an upper bound $\overline{w}_i$ on its weight in the same way as the construction of interval uncertainty sets above. Then, for each item i, its weight w_i^k in scenario k was generated uniformly in its interval $[\underline{w}_i, \overline{w}_i]$.

In the first experiment we consider problem instances with few high stacks. The second experiment considers problem instances with many small stacks. The third experiment 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 3. 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 4. This is denoted as "AI".
- The adjustable model with interval-based uncertainty, based on the formulation $(ARI - PG)$, using the pattern generation heuristic with 1000 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 ([76]) to solve all MIP formulations. 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.

Experiment 1: 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.

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 5.1: Experiment 1, median computation times in seconds.

We present the median computation times (in seconds) in Table 5.1. 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. 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.

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

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 5.2, 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). 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.

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

The average gaps for adjustable objective values are presented in Table 5.3. 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. 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.

Experiment 2: 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 5.1, 5.2, and 5.3 as Tables 5.4, 5.5, and 5.6.

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 5.4: 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.

Also the adjustable objective gaps increase, and show larger differences between the solution approaches than before. On these instances, the heuristic approaches tend to

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

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

Summary for Experiments 1 and 2

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.

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 5.7: Experiment 3, median computation times in seconds.

Experiment 3: impact of payload parameter

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 5.7 – 5.11.

Note that computation times, presented in Table 5.7, are not monotone in a. Instead, solving problems with a being in the vicinity of 1 tends to take longer 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.

Considering the strict objective values (Table 5.8), 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 the gap for $a \geq 1$. Absolute objective values are smaller for increasing a, which may also lead 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 5.8: Experiment 3, average gaps for strict objective values in percent.

We show the adjustable objective value gaps in Table 5.9. Note that overall gaps tend to be smaller than for strict robustness, which is in agreement with the observations made in the discussion of the results of Experiments 1 and 2. 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.

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

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

In Tables 5.10 and 5.11, we investigate the sensitivity of solutions regarding the parameter a. In Table 5.10, we evaluate solutions calculated for varying values of a as if they were solutions to $a = 0.5$, and to $a = 1.5$ in Table 5.11. We find that those solutions that have the smallest gap in Table 5.9 (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.

5.5.2 Large instances

In this experiment, we tested the performance of our heuristic algorithms on larger datasets. The considered instance sizes are given in Table 5.12, 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 5.5.1, using $d = 0.6$ and $a = 1$.

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

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 5.12: 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 (5.87)-(5.94), which is denoted as "Start". For the large instances, it is not possible to compute the lower bounds, that are used in computing algorithm gaps, by exact algorithms within time limit as for the small instances. Therefore, to have a better evaluation on quality of the proposed heuristic methods, 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 5.13, 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 tends 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 5.13: Experiment 4, average gaps and comparison to best (in percent) for adjustable objective values.

Note that the lower bound used for Table 5.13 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 adjustable objective value found on every instance in the right columns of Table 5.13.

5.6 Conclusions

In this chapter 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, one where adjustments in the item order within each stack 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 Bertsimas and Sim (see [21]), where the total relative deviation of item weights from their nominal values is bounded by some parameter Γ. Using such an uncertainty set, Algorithms 3 and 4 are still applicable, with only slight differences in the computation of worst-case scenarios.

Finally, the adjustable approach presented in this chapter 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., [65]). 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).

Chapter 6

Conclusions

This chapter summarizes the main results of this thesis. We also discuss some possibilities for future work on the topic of storage loading problems under uncertainty.

Thesis summary

In this thesis we studied some storage loading problems motivated from several practical contexts, under different types of uncertainty on the items' data. To have robust stacking solutions against the data uncertainty, we applied the concepts of strict and adjustable robustness.

In Chapter 2, complexity of various storage loading problems with stacking constraints was studied. Apart from the known results from [31], we showed some other polynomial solvable as well as NP-hard cases of the deterministic problems. We proved that complexity of some special cases of the robust problems can be derived from their corresponding deterministic versions. We furthermore pointed out some interesting settings in which the adjustable robust problems can be solved more efficiently than the strict ones.

In Chapter 3 we proposed different MIP formulations based on different modelling approaches for some deterministic and robust storage loading problems. Furthermore, by a comprehensive computational study, our contributions are to figure out the performance profiles of the proposed formulations, as well as to point out which formulation efficiently performs for which data setting.

In Chapter 4 we proposed a robust optimization approach dealing with a storage loading problem under stochastic uncertainty. To evaluate the stacking solution, we introduced a concept so-called security level, which is defined by the probability that the stacking solution is feasible when the uncertain items are realized. We offered several rule-based ways of scenario generation to derive different uncertainty sets, and analyzed the impact of various factors on the trade-off between security level and cost of the robust stacking solutions.

In Chapter 5 we introduced a novel approach in dealing with stability issues of stacking configurations. Our key idea is to impose a limited payload on each item depending on its weight. We then studied a storage loading problem with the interaction of stacking and payload constraints, as well as uncertainty on the weights of items. We proposed exact decomposition and heuristic solution algorithms as solution approaches for the robust counterparts of the problem.

Future work

There are several directions for future work. The first direction could be to complete Table 2.2 about complexity results of solving the stated robust problems. For instance, the adjustable robust counterpart of the uncertain problem with stacking height $b = 2$ and discrete uncertainty is an interesting open problem. Another open problem is the strictly robust counterpart of the uncertain problem with stacking height $b \geq 3$ and interval uncertainty. In relation with the first direction, future research could focus on designing approximation algorithms for solving the known NP-hard storage loading problems as well as the ones with open complexity.

The second direction could be to consider storage loading problems with other types of uncertainty sets, such as in the model of Bertsimas and Sim [21], where the actual data of only a limited number Γ of items may differ from their nominal values. Furthermore, other objective functions could be considered. Such an objective would, for example, be to minimize the number of unordered stackings with respect to given soft stacking constraints. Another objective would be to find a stacking solution for here-and-now items maximizing the expected number of wait-and-see items that can be stored into the storage area (given the data distribution of the items).

Another possible direction of future work could be to apply other types of robustness, such as recoverable robustness [65] or recovery-to-optimality [51], to the area of storage loading problems. It would be also interesting to consider storage loading problems with a view from stochastic programming.

Bibliography

[1] http://www.extremeoptimization.com/documentation/statistics/default.aspx.

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

[3] 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.

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

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

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

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

[8] A. Atamtürk and M. Zhang. Two-stage robust network flow and design under demand uncertainty. *Operations Research*, 55(4):662–673, 2007.

[9] J. Behnamian and B. Eghtedari. Storage system layout. In R. Z. Farahani and M. Hekmatfar, editors, *Facility Location: Concepts, Models, Algorithms and Case Studies*, Contributions to Management Science, pages 419–450. Physica-Verlag Heidelberg, 2009.

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

[11] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming Series A*, 99(2):351–376, 2004.

[12] A. Ben-Tal, D. Hertog, and J.-P. Vial. Deriving robust counterparts of nonlinear uncertain inequalities. *Mathematical Programming*, 149(1):265–299, 2015.

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

[14] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

[15] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424, 2000.

[16] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* Society for Industrial and Applied Mathematics, Philadelphia, 2001.

[17] A. Ben-Tal, A. Nemirovski, and C. Roos. Robust solutions of uncertain quadratic and conic-quadratic problems. *SIAM Journal on Optimization*, 13(2):535–560, 2002.

[18] 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.

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

[20] D. Bertsimas, D. Pachamanova, and M. Sim. Robust linear optimization under general norms. *Operations Research Letters*, 32(6):510–516, 2004.

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

[22] U. Blasum, M. R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, and T. Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, 1999.

[23] O. Boni and A. Ben-Tal. Adjustable robust counterpart of conic quadratic problems. *Mathematical Methods of Operations Research*, 68(2):211–233, 2008.

[24] O. Boni, A. Ben-Tal, and A. Nemirovski. Robust solutions to conic quadratic problems and their applications. *Optimization and Engineering*, 9(1):1–18, 2008.

[25] 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.

[26] 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.

[27] N. Boysen and S. Emde. The parallel stack loading problem to minimize blockages. *European Journal of Operational Research*, 249(2):618–627, 2016.

[28] N. Boysen, M. Fliedner, F. Jaehn, and E. Pesch. Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, 220(1):1–14, 2012.

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

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

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

[32] C. Büsing, S. Knust, and X. T. Le. Trade-off between robustness and cost for a storage loading problem: rule-based scenario generation. *Working paper.*

[33] H. J. Carlo, I. F. A. Vis, and K. J. Roodbergen. Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2):412–430, 2014.

[34] R. Dekker, P. Voogd, and E. van Asperen. Advanced methods for container stacking. *OR Spectrum*, 28(4):563–586, 2006.

[35] 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.

[36] 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.

[37] D. Ding and M. C. Chou. Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. *European Journal of Operational Research*, 246(1):242–249, 2015.

[38] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming Series A*, 91(2):201–213, 2002.

[39] S. Even and O. Kariv. An $\mathcal{O}(n^{2.5})$ algorithm for maximum matching in general graphs. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–112. IEEE, New York, 1975.

[40] B. S. Everitt and A. Skrondal. *The Cambridge dictionary of statistics, Fourth edition.* Cambridge University Press, 2010.

[41] J. E. Falk. Exact solutions of inexact linear programs. *Operations Research*, 24(4):783–787, 1976.

[42] V. Gabrel, C. Murat, and A. Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014.

[43] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[44] L. El Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

[45] L. El Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998.

[46] A. H. Gharehgozli, Y. Yu, R. de Koster, and J. T. Udding. A decision-tree stacking heuristic minimising the expected number of reshuffles at a container terminal. *International Journal of Production Research*, 52(9):2592–2611, 2014.

[47] A. Ghoniem and H. D. Sherali. Models and algorithms for the scheduling of a doubles tennis training tournament. *Journal of the Operational Research Society*, 61(5):723–731, 2010.

[48] A. Ghoniem and H. D. Sherali. Set partitioning and packing versus assignment formulations for subassembly matching problems. *Journal of the Operational Research Society*, 62(11):2023–2033, 2011.

[49] M. Goerigk, S. Knust, and X. T. Le. Robust storage loading problems with stacking and payload constraints. *European Journal of Operational Research*, 253(1):51–67, 2016.

[50] M. Goerigk and A. Schöbel. A note on the relation between strict robustness and adjustable robustness. Technical report, Institute for Numerical and Applied Mathematics, University of Göttingen, 2012.

[51] M. Goerigk and A. Schöbel. Recovery-to-optimality: A new two-stage approach to robustness with an application to aperiodic timetabling. *Computer and Operations Research*, 52(A):1–15, 2014.

[52] R. Gould and C. Ryan. *Introductory statistics: exploring the world through data, 2nd edition.* Pearson, 2016.

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

[54] J. Jacod and P. Protter. *Probability essentials.* Springer, 2004.

[55] R. Jans. Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. *INFORMS Journal on Computing*, 21(1):123–136, 2009.

[56] R. Jans and J. Desrosiers. Efficient symmetry breaking formulations for the job grouping problem. *Computers and Operations Research*, 40(4):1132–1142, 2013.

[57] 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.

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

[59] 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.

[60] T. Koch. *Rapid mathematical programming.* PhD thesis, Technische Universität Berlin, 2004.

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

[62] X. T. Le and S. Knust. MIP-based approaches for robust storage loading problems with stacking constraints. *Computers and Operations Research*, 78:138–153, 2017.

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

[64] 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.

[65] 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, 2012.

[66] M. Mani, A. K. Sing, and M. Orshansky. Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization. In *ICCAD '06 Proceedings of the 2006 IEEE/ACM international conference on computer-aided design*, pages 19–26, New York, NY, USA, 2006. ACM.

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

[68] 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.

[69] T. Nishi and M. Konishi. An optimisation model and its effective beam search heuristics for floor-storage warehousing systems. *International Journal of Production Research*, 48(7):1947–1966, 2010.

[70] 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 J. W. Böse, H. Hu, C. Jahn, X. Shi, R. Stahlbock, and S. Voß, editors, *Computational logistics*, volume 6971 of *Lecture Notes in Computer Science*, pages 286–301. Springer-Verlag Berlin Heidelberg, 2011.

[71] D. Pacino, A. Delgado, R. M. Jensen, and T. Bebbington. An accurate model for seaworthy container vessel stowage planning with ballast tanks. In H. Hu, X. Shi, R. Stahlbock, and S. Voß, editors, *Computational logistics*, volume 7555 of *Lecture Notes in Computer Science*, pages 17–32. Springer-Verlag Berlin Heidelberg, 2012.

[72] B. Rouwenhorst, B. Reuter, V. Stockrahm, G. J. van Houtum, R. J. Mantel, and W. H. M. Zijm. Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122(3):515–533, 2000.

[73] 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.

[74] A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[75] A. L. Soyster. A duality theory for convex programming with set-inclusive constraints. *Operations Research*, 22(4):892–898, 1974.

[76] IBM ILOG CPLEX Optimization Studio. *CPLEX User's Manual Version 12 Release 6*, 2013.

[77] A. Takeda, S. Taguchi, and R. H. Tütüncü. Adjustable robust optimization models for a nonlinear two-period system. *Journal of Optimization Theory and Applications*, 136(2):275–295, 2008.

[78] D. J. Thuente. Duality theory for generalized linear programs with computational methods. *Operations Research*, 28(4):1005–1011, 1980.

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

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

[81] 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.

[82] J. Wiese, L. Suhl, and N. Kliewer. Mathematical models and solutions methods for optimal container terminal yard layouts. *OR Spectrum*, 32(3):427–452, 2010.

[83] T. Winter and U. T. Zimmermann. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96(1):287–315, 2000.

[84] 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.

[85] 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 B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, editors, *Proceedings of the 2010 Winter Simulation Conference*, pages 1976–1989. Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ, 2010.

[86] C. Zhang, W. Chen, L. Shi, and L. Zheng. A note on deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 205(2):483–485, 2010.