UNIVERSITÄT OSNABRÜCK

Optimizing the retrieval effort in stack-based storage systems

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

vorgelegt von

Sven Boge, M.Sc.

28. Oktober 2022

Contents

Chapter 1

Introduction

International trade has steadily increased in recent decades because of the globalization of consumer markets and production chains. Due to its low costs, the transport of goods by sea accounts for over 90% of the world trade, and moreover 13% of the transported goods (in dead-weight tons) are shipped by container ships carrying their goods in standard-size containers (cf. [54, 108]). The world maritime trade increased from less than 6 billion tons carried trade in the year 2000 to more than 12 billion tons in 2020 and is expected to increase further to almost 18 billion tons in 2030 (cf. [55]). The carried goods in containers are transshipped in huge container terminals for onward transport where container ships, trains and trucks deliver and collect a huge number of containers. Furthermore, not only the volume of world trade increased, but also the associated production capacities, so that operators of production facilities in the steel industry, for example, are faced with storage challenges. Additionally to the already mentioned container terminals and container ships in maritime transport, transshipment points play an important role in the transport regardless of the goods considered like in warehouses or tram depots, cf. the survey in [73]. Increasing the efficiency may lead to faster access times on stored goods, could reduce the energy consumption of used machines, and hence the operational costs.

Often a system of multiple stacks is used to store a large quantity of items in a storage area of limited size. Indeed, stack-based storage systems are easy and inexpensive in operation but only the topmost item of every stack can be retrieved at a time during the retrieval process due to the last–in, first–out (LIFO) policy. In this context, an efficient operation of these storages is essential in competition since a large number of items are handled in these areas, so that inefficient processing would turn them into a bottleneck. Usually, in storage areas the following optimization problems are considered: In *loading problems*, incoming items arrive at a storage area and have to be assigned to appropriate locations. In *premarshalling problems*, items already in a storage area are sorted to increase efficiency. In *unloading problems*, some (or even all) items have to be retrieved in a given sequence to be further transported or processed, respectively. The described loading and premarshalling problems aim to reduce the retrieval effort during the unloading phase as much as possible so that these two problems always include additional available information of the subsequent problem. On the one hand, the retrieval effort includes cost-causing effects as energy consumption, machine wear, etc. and, on the other hand, the time required for retrieving the desired items which cause cost for employees and directly influences the waiting time for downstream processes. In turn, the time required for retrieving all demanded items is mainly influenced by relocations of items within the storage system during the retrieval process. Items must be relocated as a consequence of the predefined retrieval sequence and the LIFO policy of a stack-based storage system. If an item that has to be retrieved later on is placed above an item that has to be retrieved earlier, then the former item must be relocated to another stack to allow access to the latter item, i.e., the former item blocks access to the latter item. Farther, the relocated item may block another items again and so on. A smartly guided retrieval strategy as well as an advantageous starting configuration of the storage due to premarshalling or convenient loading offers the possibility of reducing the retrieval effort. Obviously, all three optimization problems are important to consider in order to guarantee efficient storage operations. Hence, all three problems are incorporated in this thesis and a brief overview is given in the following paragraphs (for a more detailed overview cf. [73]).

Typical for loading problems is that items arrive at a storage area of limited size in a given sequence and must be stored in the given order without allowing further relocations (movements of items to other stacks) after items are placed in the storage. In the elementary loading problem, called *parallel stack loading problem* (PSLP), the storage is initially empty and item priorities of the subsequent unloading stage (attributes that determine the retrieval order of the items) are taken into account additionally. [15, 31] investigated the problem with different (surrogate) objective functions. Moreover, the problem of loading items is considered under various conditions. For instance, [11, 27] solved it as an online problem using heuristics, whereas others focused on offline problem variants and considered different robust settings (cf., e.g., [16, 69]) or restrictions on items regarding stackability as well as items that are already placed in the storage (cf., e.g., [18, 74, 83, 120]).

The *premarshalling problem* (PMP) aims to optimize the performance of a storage area before the main unloading phase starts. Usually, in the practical optimization process, the items stay inside the storage and no additional space exists. All relocations between the stacks are counted equally in the objective function and the final layout is always perfectly sorted in such a way that no additional relocations are necessary in a later unloading phase. A common assumption is that the time for the resorting is not limited and the goal is to perform the rearrangement with as few relocations as possible, i.e., in the shortest possible time. Overall, the problem occupies an important phase in the optimization process, which exploits free time windows without loading or unloading operations and thus relieves the important and highly time-constrained unloading phase. In this context, a wide range of heuristics covering different methods have been developed, e.g. local search approaches (cf. [21, 47, 50, 61, 72]), opening procedures (cf. [35, 39, 46, 51, 58, 115]), or even machine learning approaches (cf. [49]) as well as exact algorithms such as *integer programs* (IPs) (cf. [26, 70, 84]), branch-and-bound (cf. [86, 97, 98, 105, 122]) or branch-and-price approaches (cf. [17]). Moreover, different aspects of the problem have been considered which include crane scheduling of more than one crane (cf. [24]), uncertainty models (cf. [80, 89, 105]), transport costs instead of counting relocations ([25]), temporary additional space for placing items during the sorting process (cf. [115]), and limited time for performing the premarshalling process (cf. [126]).

As already mentioned, during the unloading phase, items located in a storage shall leave it in a specific order related to a given sequence and one has to decide which items leave the storage in which order and which relocations are performed. To access and retrieve a certain item, it may be necessary to relocate some blocking items above, which results in unproductive relocations. The goal of this optimization problem is to perform the retrieval of all relevant items with as few relocations as possible, just as in the PMP. The usually used representative of unloading problems is the *blocks relocation problem* (BRP), also known as *container relocation problem* (CRP), and a large number of publications have already been published in connection with this problem. A classification scheme for different BRP variants can be found in [79]. Common assumptions are that all items must be retrieved in a fixed order and all relocations are counted equally. Again, a large number of heuristics have been proposed, e.g., various opening procedures (cf. [19, 20, 33, 57, 58, 59, 60, 64, 85, 109, 112]), branch-and-bound based heuristics (cf. [3, 39, 106, 107]), local search algorithms (cf. [6, 22, 37, 92]), and machine learning approaches (cf. [119]) as well as exact approaches such as IPs (cf. [19, 26, 43, 78, 79, 85, 100, 112, 117, 118]), branch-and-bound, -cut, -price (cf. [2, 33, 34, 78, 87, 96, 99, 107, 118, 121, 124]). Furthermore, several facets of the problem have been investigated, including uncertainty or incomplete information (cf. [4, 12, 13, 42, 68, 125]), travel times instead of uniform relocation costs (cf. [1, 6, 52, 53, 71, 76, 93, 94, 92, 109, 111]), moving multiple items at a time (cf. [76, 123]), due dates and waiting times (cf. [77]), a restriction to exactly two priority classes (multiple items with the same priority, cf. [82]) and also various other theoretical investigations.

A generalization of the BRP is the *blocks relocation problem with item families* (BRPIF), in the literature also called *slab stack shuffling* (SSS) problem. In contrast to the BRP, all items belong to a certain family indicating the main type of the item and not all items must be retrieved, but rather items with certain item families must be selected in a predefined order. Usually, there exist multiple items with the same item family, which implies that the selection process is a very important part of the problem. For this problem variant, opening procedures (cf. [103, 116]), a population based algorithm (cf. [23]) and local search algorithms (cf. [23, 91, 95, 101, 102]) have been developed as well as exact algorithms based on IPs (cf. [36, 38, 88, 91, 103, 116]) or dynamic programming (cf. [7, 103]). In the literature, other problems as crane scheduling (cf. [88]), a lot-building problem (cf. [36]), and a PMP ([44, 45]) are investigated in connection with the BRPIF.

The three problem areas of loading, premarshalling and unloading play an important role in running warehouses with stack-based storage systems and have therefore been treated in many different aspects. Regarding the high complexity of the problems, often additional simplifying assumptions such as unrealistic many free space or very restrictive constraints on item relocations in the storage are considered while possibly relevant aspects as uncertainty are omitted. For example, in several cases, items have to be moved at most once or items must be moved back to their departure stack after retrieving the current target item in the unloading process. Both limitations affect the model very strongly, since large parts of the practically achievable solution space are cut off. However, the main target is to reduce the retrieval effort as much as possible to run the storage efficiently which is directly linked to the particular relocations performed within the storage. In this context, surrogate objective functions are used to be able to apply exact algorithms to problem instances of non-trivial size. For example, often only blocking items or adjacent blocking items are counted instead of concentrating on the actual relocations that are necessary to retrieve the items in the current setting. However, the solutions of these algorithms are then used afterwards, not in relation to the surrogate objective but in the context of the originally intended more complex objectives. Furthermore, in practice one has to deal with the impact of omitted aspects of the problem and constraints that are different than expected. One target of this thesis is to cope directly with the actually desired objectives and not with simplified versions of the originally intended problem as well as to incorporate decisive components if possible. This in turn allows using more realistic models and achieving better results w.r.t. the more precisely defined objective functions. In particular, the problem complexity can be traced back to the subsequent unloading problem and it will become apparent in the course of this thesis that it may often be useful to divide the problems into two partial aspects and to solve them in two stages. On the one hand, this decomposition allows several aspects of a problem to be evaluated hierarchically and, on the other hand, to deal with the desired objective function instead of falling back to surrogate objective functions. For example, the most critical aspect of a problem may be varied in the first stage with a local search approach while in the second stage a reduced problem with fixed decisions and hence a reduced complexity can be solved quickly.

Organization of the thesis

This thesis is organized as follows. In Chapter 2, we investigate the elementary loading problem, the PSLP. We consider the PSLP with the objective to minimize the *number of* *reshuffles* (reshuffles is a alternatively used term for *relocation* in the literature) in the unloading phase. Since in the PSLP the incoming items have to be stored according to a fixed arrival sequence and further adjusting relocations are not allowed, some relocations cannot be avoided later on. In this context, we give a strengthened $\mathcal{NP}$-completeness proof and study the two surrogate objective functions *total number of unordered stackings of adjacent items* and *total number of badly placed items*. These objective functions are established in the literature to estimate the number of reshuffles and we compare them theoretically as well as in a computational study with the desired objective function number of reshuffles. For this purpose, *mixed-integer programming* (MIP) formulations and a *simulated annealing* (SA) algorithm are proposed. Altogether, it turned out that minimizing the number of reshuffles directly with a two-stage SA approach is superior to the application of exact MIP approaches to the surrogate objective function mentioned above.

Within Chapter 3, we consider the PMP, where items in a storage area have to be sorted for convenient retrieval. A new model for uncertainty is introduced, where the priority values induced by the retrieval sequence of the items are uncertain. We propose a robust optimization approach for this setting, study complexity issues and provide different MIP formulations. Moreover, we investigate helpful properties which can speed up the solution process in several cases. In a two-stage approach we use the results from the theoretical analysis and MIP formulations to calculate optimal objective values of optimally robust storage configurations in a first step and compute premarshalling solutions with a minimal number of reshuffles to reach optimal configurations in a second step. In a computational study using a wide range of benchmark instances from the literature, we investigate both the efficiency of the approach as well as the benefit and cost of robust solutions. We find that it is possible to achieve a considerably improved level of robustness by using just a few additional relocations in comparison to solutions which do not take uncertainty into account.

In Chapter 4 we consider the process of unloading items from a storage (e.g., a warehouse, depot, etc.) in the case of the BRPIF. For a given sequence of families, it has to be decided which item of each demanded family is unloaded from the storage with the objective to minimize the total number of reshuffles. Besides new complexity results, we propose IP formulations and a two-stage SA algorithm. Computational results are presented for real-world data from a company, benchmark instances from the literature, and randomly generated instances with different characteristics.

In Chapter 5 we conclude the thesis with a comprehensive summary, highlight the main contributions and give an outlook for further research.

Chapter 2

The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage

In this chapter, we discuss the *parallel stack loading problem* (PSLP) as an important basic problem of the loading stage minimizing the number of reshuffles in the unloading stage. The study in this chapter has already been published in Boge and Knust [8] and is organized as follows. First, we give an introduction in Section 2.1. In Section 2.2 we describe the problem setting more precisely, in Section 2.3 we prove that the problem is strongly $\mathcal{NP}$-hard for all considered objective functions. In Section 2.4 we derive some bounds for the differences between different objective functions estimating the largest errors which can occur. Additionally, we propose new lower bounds. Section 2.5 is devoted to *mixed-integer programming* (MIP) formulations and a *simulated annealing* (SA) algorithm. In Section 2.6 we report results of a computational study. Finally, some concluding remarks can be found in Section 2.7.

2.1 Introduction

Storing items in a storage area of limited size is one of the mentioned three major optimization problems regarding transshipment points or warehouses of items which are organized in stacks. Usually, the process is organized in two stages: in the *loading stage* incoming items arrive at a storage area and have to be assigned to appropriate locations in stacks. Each item has some associated data (e.g., its weight, size, destination, need for special equipment, expected retrieval time) taken into account in different stacking policies (cf. Dekker et al. [27]). Later on, in the *unloading stage* some (or even all) items have to be retrieved in a certain sequence (to be further transported by train, ship or truck). Since usually only the topmost items in the stacks can immediately be retrieved, this leads to relocations (reshuffles) of items blocking the requested items. In order to increase the efficiency of the storage area, such (unproductive) moves should be avoided as much as possible.

A lot of literature has tackled the unloading problem minimizing the number of reshuffles (cf., e.g., Lehnfeld and Knust [73]). However, often reshuffles cannot be avoided in the unloading stage since the items have not been stacked well in the previous loading stage. This is caused by the fact that during the loading process usually the items arrive consecutively (e.g., by trucks or trains) and have to be stored according to this fixed order. Additionally, often at this time it is also not known in which sequence the items will be retrieved later on. For this reason, reshuffles cannot be exactly planned in the loading stage, but should be anticipated somehow. One possible approach considered in the literature is to minimize the number of expected reshuffles (Kang et al. [61] and Kim et al. [65]). In both publications it is assumed that reshuffles are caused by storing items in a wrong order w.r.t. their weights and some weight distribution function is known.

Boysen and Emde [15] considered a surrogate objective function in the loading phase, trying to minimize the total number of so-called “blockages” which occur if an item to be retrieved later is stacked directly on top of another more urgent item which is to be retrieved earlier. However, as stated in their conclusion “an investigation of which surrogate objective shows greatest accuracy in approximating the retrieval effort” would be interesting. In this chapter, we try to deal with this research question in more detail. We tackle some theoretical issues and provide results of a computational study comparing different reshuffle-oriented objective functions for storage loading problems.

2.2 Problem formulation

In this section, we describe the considered problem more formally and introduce the used notations. We are given a storage area which consists of m stacks $\mathcal{Q} = \{1, \dots, m\}$, each stack contains b levels (i.e., at most b items can be stored in each stack). In the so-called PSLP considered by Boysen and Emde [15], n items from a set $\mathcal{I} = \{1, \dots, n\}$ arrive in a sequence π^{in} and have to be stored in this sequence (which means that an item arriving later cannot be stacked under an item arriving earlier). Each item i has a priority value $p_i \in \{1, \dots, n\}$ derived from its (estimated) retrieval time. If $p_i < p_j$, item i is expected to be retrieved earlier than item j. The goal is to determine a feasible storage configuration c in which each item $i \in \mathcal{I}$ is assigned to a location in a stack $q \in \mathcal{Q}$ such that no stack contains more than b items.

In practice, often some other constraints may have to be respected, for example, stacking constraints (not every item may be stacked on every other one), items must be stored in groups (e.g., w.r.t. destinations, sizes or weights), some items need additional special equipment (e.g., reefer containers).

For a given storage configuration, an item is called “blocked” if one or more items with later retrieval time (so-called “blocking” items) are stacked above it in the same stack (such a situation is sometimes also called a “mis-overlay” Voß [110] or an “overstow” Delgado et al. [28]). Before retrieving a blocked item, each blocking item has to be removed from this stack and relocated to another stack. Such an unproductive move is also called a “reshuffle”.

To evaluate a storage configuration with respect to blockings and the number of reshuffles, the following measurements have been suggested:

- *total number of unordered stackings of adjacent items* (US_{adj}) mentioned in Boysen and Emde [15] and Lehnfeld and Knust [73]: Every pair of *adjacent* items in a stack is counted as unordered if the upper item blocks the one below.

- *total number of badly placed items* (BI): The term is also called “number of confirmed relocations” in Kim and Hong [64]. An item is “badly placed” (cf. Forster and Bortfeldt [39]) if it is blocking an item placed below in the same stack (not only the adjacent item directly below). We call an item “well placed” if it is not badly placed.

- RS: The actual number of reshuffles needed to unload all items from the stacks according to their priorities. We do not count the retrieval operations here (since all n items have to be retrieved in the unloading process, we always have a constant number of n retrievals).

Example 2.1. *To illustrate the difference between the objective functions* US_{adj} *and* BI*, consider a storage area with* $n = 9$ *items in* $m = 3$ *stacks of height* $b = 5$*. For the configuration* c *shown in Figure 2.1 we have* $US_{adj}(c) = 3$ *(items* $5, 7, 8$*) and* $BI(c) = 4$ *(items* $4, 5, 7, 8$*).* □

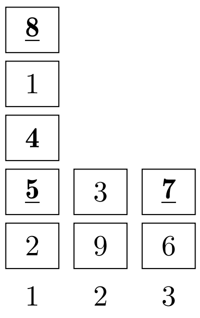


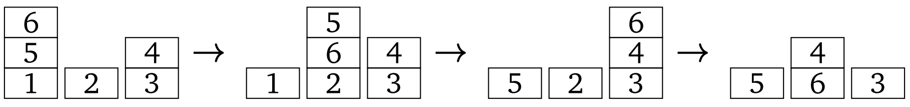
Figure 2.1: Unordered stackings (underlined) and badly placed items (in bold).

Obviously, the values $US_{adj}(c)$ and $BI(c)$ define lower bounds on the total number of reshuffles since each blocking item has to be relocated at least once. Furthermore, it is easy to see that for each storage configuration c the inequality $US_{adj}(c) \leqslant BI(c)$ holds, since all unordered stackings are also counted in the total number of badly placed items. However, as shown by Boysen and Emde [15], finding a solution for the PSLP minimizing US_{adj} is already strongly $\mathcal{NP}$-hard.

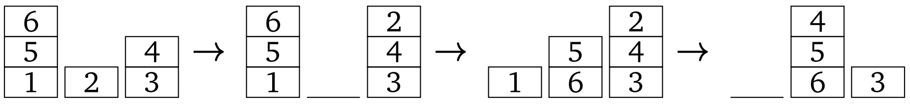
Later on, after all items have been stored in some way, in the so-called *blocks relocation problem* (BRP), the items have to be retrieved from the storage in an order respecting their priorities. Usually, for the priorities, two different situations are considered. While in some situations all priorities are different (i.e., the retrieval sequence is uniquely determined), in other situations also duplicates occur (i.e., several items may have the same priority value, for example, if they have to be loaded onto the same ship or train). In this case, all items with priority 1 have to be retrieved first (in an arbitrary order), then all items with priority 2, etc.

Furthermore, in the literature the BRP appears in two different versions (cf., e.g., Caserta et al. [19]): in the "restricted" BRP only so-called "forced moves" are allowed which means that only blocking items on top of the next item to be retrieved may be reshuffled. On the other hand, in the "unrestricted" BRP also so-called "voluntary moves" are allowed (which means that arbitrary items may be reshuffled). In the following, we use the notations RS_r and RS_u to denote the number of reshuffles in the restricted and the unrestricted version, respectively.

Example 2.2. *To illustrate the difference between the restricted and the unrestricted BRP, consider a storage area with $n = 6$ items in $m = 3$ stacks of height $b = 3$.*



(a) $RS_r = 6$ reshuffles needed.



(b) $RS_u = 4$ reshuffles needed.

Figure 2.2: Restricted and unrestricted BRP.

For the restricted BRP (cf. Figure 2.2a), at first, items 5 and 6 have to be reshuffled to get access to item 1. After retrieving this item, items 5 and 6 have to be reshuffled again to free item 2. Finally, after retrieving item 2, two more reshuffles of items 4 and 6 have to be done. Hence, in total $RS_r = 6$ reshuffles are needed. On the other hand, for the unrestricted BRP (cf. Figure 2.2b), *only* $RS_u = 4$ *reshuffles are needed. If we first do a voluntary move of item* 2*, items* 5 *and* 6 *have to be reshuffled only once.* □

While for the unrestricted version less reshuffles may be necessary, computing an optimal solution is much more time consuming due to the larger solution space (cf. Tanaka and Mizuno [96]). Therefore, a lot of heuristics only deal with the restricted version. However, it was shown by Caserta et al. [19] that for both versions minimizing the number of reshuffles for a given storage configuration is strongly $\mathcal{NP}$-hard. Scholl et al. [90] suggested different measures to predict the number of reshuffles and compared them in a computational study.

2.3 $\mathcal{NP}$-hardness

Boysen and Emde [15] already proved that the PSLP minimizing the objective function US_{adj} is strongly $\mathcal{NP}$-hard. In their proof the more general situation with duplicate priorities and an unrestricted b (as part of the input) is considered. In the following, we strengthen the result by showing that the PSLP is already strongly $\mathcal{NP}$-hard for unique priorities and any fixed $b \geqslant 6$. Moreover, since we prove $\mathcal{NP}$-completeness for the decision value 0, we cover all three objective functions US_{adj}, BI, RS. We give a reduction from the problem *mutual exclusion scheduling* (MES) on permutation graphs (cf. Jansen [56]), similar to the reduction to the BRP used in Caserta et al. [19].

Theorem 2.1. *For the PSLP and each fixed* $b \geqslant 6$ *it is strongly* $\mathcal{NP}$*-complete to decide whether all items can be stacked without any blockings.*

Proof: Obviously, the PSLP belongs to the class $\mathcal{NP}$. We show $\mathcal{NP}$-completeness by a reduction from MES on permutation graphs. In the decision version of MES we are given a set $\mathcal{V}$ of n jobs with unit processing times, M machines and T time slots. Additionally, there is an undirected conflict graph $G = (\mathcal{V}, \mathcal{E})$ where an edge $\{i, j\}$ means that jobs i and j cannot be processed in the same time slot simultaneously. We ask whether there is a partition of $\mathcal{V}$ into at most T independent sets U_t ($t = 1, \dots, T$) (independent means that there is no edge $\{i, j\} \in \mathcal{E}$ for any pair $i, j \in U_t$) with $|U_t| \leqslant M$ for all t. Here, U_t corresponds to the set of jobs processed in time slot t. Since we have M machines, at most M jobs can be processed in each time slot. In Jansen [56] it has been shown that MES is strongly $\mathcal{NP}$-complete for permutation graphs and any fixed $M \geqslant 6$. Recall that a permutation graph is a graph $G^{\psi} = (\mathcal{V}, \mathcal{E}^{\psi})$ associated with a permutation $\psi = (\psi_1, \dots, \psi_n)$ where an edge $\{i, j\} \in \mathcal{E}^{\psi}$ exists if $i < j$ and $\psi_i > \psi_j$.

Given an arbitrary instance $\mathcal{I}$ of MES with a permutation ψ, T time slots and M machines, a corresponding instance $\mathcal{I}'$ of the PSLP is constructed where for each job $i \in \mathcal{V}$ one item i with priority $p_i = i$ is introduced. Moreover, we have $m := T$ stacks of height $b := M$, and the arrival sequence for the n items is $\pi^{in} := (\psi_n, \dots, \psi_1)$. Then, for item $i < j$ (which implies $p_i < p_j$) there is an edge $\{i, j\} \in \mathcal{E}^{\psi}$ in the conflict graph if and only if $\pi^{in}_i < \pi^{in}_j$, i.e., j becomes badly placed if it is put to the same stack as i.

We illustrate this construction by a small example in Figure 2.3. In $\mathcal{I}$ there are $M = 4$ machines, $T = 3$ time slots, and $n = 10$ jobs in the permutation $\psi = (4, 1, 3, 10, 2, 6, 5, 8, 7, 9)$. Figure 2.3 shows the corresponding conflict permutation graph G^{ψ}. The constructed instance $\mathcal{I}'$ of the PSLP has $n = 10$ items $i = 1, \dots, 10$ with priorities $p_i = i$, $m = 3$ stacks of height $b = 4$, and the arrival sequence is given by $\pi^{in} = (9, 7, 8, 5, 6, 2, 10, 3, 1, 4)$.

For example, a feasible solution for $\mathcal{I}$ consists of the three independent sets $U_1 = \{4, 5, 7, 9\}$, $U_2 = \{2, 6, 8\}$, and $U_3 = \{1, 3, 10\}$. If we put each of these sets into one stack of the storage area according to π^{in}, we get a feasible solution for $\mathcal{I}'$ without any blockings (cf. Figure 2.4).

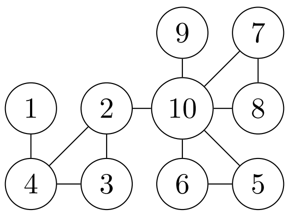


Figure 2.3: Conflict permutation graph G^{ψ} for the permutation $\psi = (4, 1, 3, 10, 2, 6, 5, 8, 7, 9)$.

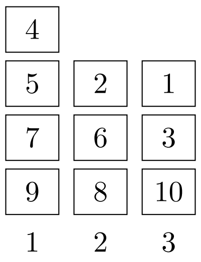


Figure 2.4: Feasible solution for the PSLP without blocking items.

In the following, we show that a feasible solution for an arbitrary instance $\mathcal{I}$ of MES exists if and only if a feasible solution without blockings exists for the corresponding instance $\mathcal{I}'$ of the PSLP.

"$\Rightarrow$": First assume that a feasible solution for $\mathcal{I}$ exists with at most T independent sets U_t, each containing at most M jobs. Two arbitrarily chosen jobs $i < j$ of an independent set U_t must satisfy $\psi_i < \psi_j$ since otherwise there would be an edge in the conflict graph. Thus, we may put all items of one independent set into one stack according to the arrival sequence π^{in} and do not obtain any blocking. Furthermore, due to $U_t \leqslant M = b$ for all t, the stack height is not violated. Hence, we get a feasible solution for $\mathcal{I}'$ consisting of at most $m = T$ stacks.

"$\Leftarrow$": Conversely, assume that there is a feasible solution for $\mathcal{I}'$ without blocking items. Let U_t be the set of items put into stack t. Each set contains at most $M = b$ elements because of the stack height. Since the items are stacked according to π^{in} and in the solution no blockings occur, for two arbitrarily chosen items $i < j$ in one stack we must have $\psi_i < \psi_j$. This implies that there cannot be an edge $\{i, j\}$ in the conflict graph. Thus, all items of one stack are an independent set consisting of at most b elements and all stacks together form a partition into at most T independent sets. □

Theorem 2.1 implies that minimizing any of the objective functions US_{adj}, BI, RS_r, RS_u is strongly $\mathcal{NP}$-hard. Furthermore, since the corresponding decision problems are hard for the decision value 0, we cannot expect any polynomial time approximation algorithm with constant performance ratio unless $\mathcal{P} = \mathcal{NP}$.

2.4 Bounds

Finding a storage configuration in the loading process for which the number of reshuffles in the retrieval process is minimized, is a very complex problem and, as far as we know, has not been tackled in the literature so far. In Boysen and Emde [15], the simpler surrogate objective function US_{adj} has been used instead, but also the number of badly placed items BI could be used as approximation. In this section, we derive some bounds for the differences between the different objective functions estimating the largest errors which can occur. We are interested in the question how good the lower bounds US_{adj} and BI are, and how many reshuffles are really needed later on if we have optimized according to these simpler surrogate objectives. Additionally, we propose new lower bounds for the objectives US_{adj} and BI.

Let an arrival sequence $\pi^{in} = (\pi_1, \dots, \pi_n)$ be given. A subsequence σ of π^{in} is a sequence $\sigma = (\pi_{j_1}, \dots, \pi_{j_h})$ with $1 \leqslant j_\lambda < j_\mu \leqslant n$ for all $1 \leqslant \lambda < \mu \leqslant h$. Such a subsequence is called "increasing" if the priorities satisfy $p_{\pi_{j_\lambda}} < p_{\pi_{j_\mu}}$ for all $1 \leqslant \lambda < \mu \leqslant h$. Analogously, a subsequence is called "decreasing" if $p_{\pi_{j_\lambda}} > p_{\pi_{j_\mu}}$ for all $1 \leqslant \lambda < \mu \leqslant h$. A "longest increasing subsequence" is an increasing subsequence of maximal length. For example, for $\pi^{in} = (2,7,5,6,4,8,3,1,9)$ and $p_i = i$ the subsequence $\sigma = (2,7,8,9)$ is increasing, while the subsequence $\sigma' = (7,5,4,3,1)$ is decreasing. $\sigma'' = (2,5,6,8,9)$ is a longest increasing subsequence.

Lemma 2.1. *Let σ be an increasing subsequence of π^{in} with length $|\sigma| > m$. Then $|\sigma| - m > 0$ is a lower bound on the total number of unordered stackings of adjacent items.*

Proof: Let $(i_1, \dots, i_\lambda)$ be an arbitrary subsequence of σ. We claim that putting them in the same stack, leads to at least $\lambda - 1$ unordered stackings (only item i_1 does not necessarily cause an unordered stacking because it may be put on the bottom of the stack or above items with larger priority values). If the items $i_1, \dots, i_\lambda$ are put consecutively into one stack, this is obvious. But it is also possible that other items are put between two items $i_{\nu-1}, i_\nu$ of σ. Let us suppose that items $j_1, \dots, j_\rho \notin \sigma$ are below item i_ν and above $i_{\nu-1}$ such that $(i_\nu, j_\rho, \dots, j_1, i_{\nu-1})$ (from top to bottom) is part of a stack. Let us assume that these items can be chosen in such a way that no unordered stacking is caused by them, which implies $p_{i_\nu} < p_{j_\rho} < \dots < p_{j_1} < p_{i_{\nu-1}}$. Since $p_{i_\nu} < p_{i_{\nu-1}}$ contradicts the fact that σ is an increasing subsequence, our assumption must be wrong and it is not possible to choose any other items of π^{in} such that $i_{\nu-1}$ and i_ν do not cause an unordered stacking. Thus, if $\lambda_q \geqslant 1$ items of σ are stacked in stack q, they result in at least $\lambda_q - 1$ unordered stackings. Adding these numbers in all stacks, gives

$$\sum_{q \in \mathcal{Q}} (\lambda_q - 1) = \sum_{q \in \mathcal{Q}} \lambda_q - m = |\sigma| - m$$

and hence $|\sigma| - m$ is a lower bound on the total number of unordered stackings. □

The best lower bound which can be obtained from Lemma 2.1, is given by a longest increasing subsequence, which can be found in $\mathcal{O}(n \log n)$ as shown in Fredman [40]. We will denote this bound by LB_{LIS}.

In Boysen and Emde [15] another lower bound $LB_{P\text{-}\infty}$ on the number of unordered stackings was proposed. It can be obtained by considering the relaxation PSLP-∞ where the stacking height b is relaxed and it is assumed that in each stack an infinite number of items can be stored. This relaxation can be solved by finding a perfect matching with minimum costs in a bipartite graph. Thus, the time complexity is $\mathcal{O}(n^3)$. Since the length of a longest increasing subsequence is independent of the stacking height b, the value LB_{LIS} is also a lower bound for problem PSLP-∞ and hence it is always dominated by $LB_{P\text{-}\infty}$.

In the following, we show that for the objective function BI, the bound LB_{LIS} can be strengthened by iteratively calculating increasing subsequences.

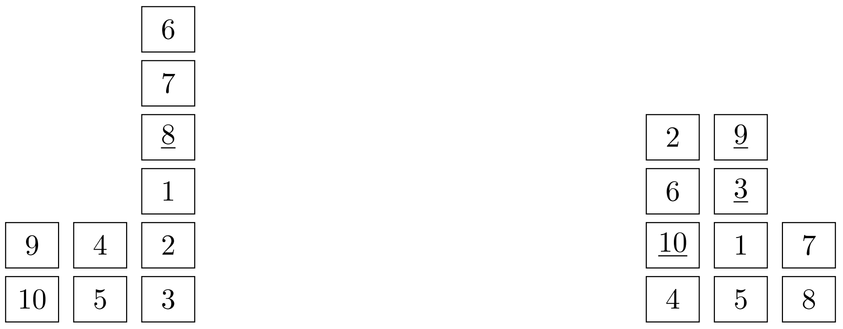
Lemma 2.2. *Let σ be an increasing subsequence of π^{in} with length $|\sigma| > m$ and $\hat{\pi}^{in} = \pi^{in} \setminus \sigma$ be the arrival sequence π^{in} reduced by all items of σ. If $\hat{\sigma}$ is an increasing subsequence of $\hat{\pi}^{in}$ with length $|\hat{\sigma}| > m$, then $|\sigma| + |\hat{\sigma}| - 2m > 0$ is a lower bound on the number of badly placed items* BI.

Proof: According to Lemma 2.1, the values $|\sigma| - m$ and $|\hat{\sigma}| - m$ are both lower bounds on the number of badly placed items (independent of any other items to be stored). Since σ and $\hat{\sigma}$ do not have any items in common, also the sum $|\sigma| + |\hat{\sigma}| - 2m$ must be a lower bound on BI. □

The proof shows that Lemma 2.1 can be used multiple times to increase the lower bound by iteratively finding longest increasing subsequences in the reduced arrival sequence. We denote by LB_{ILIS} the lower bound obtained in this way. Because every longest increasing subsequence must have a length of at least $m + 1$, we can repeat the search for a longest increasing subsequence at most $b - 1$ times. Hence, calculating LB_{ILIS} can be done in $\mathcal{O}(nb \log n)$.

In contrast to the simpler bound LB_{LIS} which is always dominated by $LB_{P\text{-}\infty}$, we do not have any dominance between the bounds LB_{ILIS} and $LB_{P\text{-}\infty}$, which is shown by the following example.

Example 2.3. *Consider the sequence* $\pi^{in} = (3, 5, 2, 1, 8, 7, 10, 4, 6, 9)$ *with* $p_i = i$ *and* $m = 3$. *There are two longest increasing subsequences of length 4, namely* $(3, 5, 8, 10)$ *and* $(1, 4, 6, 9)$. *Thus,* $LB_{ILIS} = (4 - 3) + (4 - 3) = 2$, *while* $LB_{P\text{-}\infty} = 1$ *(cf. Figure 2.5a).*



(a) $\pi^{in} = (3, 5, 2, 1, 8, 7, 10, 4, 6, 9)$. (b) $\pi^{in} = (4, 5, 8, 10, 1, 3, 6, 7, 9, 2)$.

Figure 2.5: Optimal solutions of PSLP-∞ for Example 2.3.

However, we may also have $LB_{P\text{-}\infty} > LB_{ILIS}$, *which is shown by the following instance. Consider the sequence* $\pi^{in} = (4, 5, 8, 10, 1, 3, 6, 7, 9, 2)$ *with* $p_i = i$ *and* $m = 3$. *The longest increasing subsequence is* $\sigma = (4, 5, 6, 7, 9)$ *and there is no other increasing subsequence with a length greater than* m. *Thus,* $LB_{ILIS} = 5 - 3 = 2$, *while* $LB_{P\text{-}\infty} = 3$ *(cf. Figure 2.5b).* □

Now we focus on the question how large the difference between the two surrogate objectives US_{adj} and BI can be.

Lemma 2.3. *For any feasible configuration* c *of the PSLP we have*

$$BI(c) - US_{adj}(c) \leqslant m(b - 2)$$

and this bound is tight. Furthermore, there are instances where tightness also holds for configurations optimized according to US_{adj}.

Proof: For an arbitrary configuration c let $\mathcal{Q}^+ \subseteq \mathcal{Q}$ be the subset of stacks with at least one badly placed item, B_q be the number of badly placed items in stack q and U_q the number of unordered stackings in stack q. Due to the maximum stack height b, we have $B_q \leqslant b - 1$ because the item at the bottom of a stack cannot be badly placed. Thus, the total number of badly placed items is bounded from above by

$$BI(c) = \sum_{q \in \mathcal{Q}} B_q = \sum_{q \in \mathcal{Q}^+} B_q \leqslant \sum_{q \in \mathcal{Q}^+} (b - 1) = |\mathcal{Q}^+| \cdot (b - 1). \tag{2.1}$$

On the one hand, a stack q holding a badly placed item, contains also at least one unordered stacking $U_q \geqslant 1$ (among the badly placed items in q consider the one stored in the lowest level). On the other hand, a stack without any badly placed item also does not contain any unordered stacking. Thus, the total number of unordered stackings is bounded from below by

$$\mathrm{US}_{adj}(c) = \sum_{q\in\Omega} U_q = \sum_{q\in\Omega^+} U_q \geqslant |\Omega^+|. \tag{2.2}$$

By combining the inequalities in (2.1) and (2.2), and due to $|\Omega^+| \leqslant m$, we get

$$\mathrm{BI}(c) - \mathrm{US}_{adj}(c) \leqslant |\Omega^+| \cdot (b-2) \leqslant m \cdot (b-2).$$

In the following, we show tightness of this bound. Especially, tightness does not only hold for an arbitrary configuration c, but also for configurations which have been obtained by optimizing w.r.t. US_{adj}. For this purpose, consider the following family of instances. We have $n = m \cdot b$ items with priorities $p_i = i$ for $i = 1, \dots, n$ which arrive according to the sequence

$$\begin{aligned}\pi^{in} = (&1, 2, \dots, m|\\ &n-m+1, n-m+2, \dots, n|\\ &n-m, n-m-1, \dots, 2m+1|\\ &m+1, m+2, \dots, 2m).\end{aligned}$$

This sequence consists of two increasing subsequences $\sigma_1 = (1, 2, \dots, m)$ and $\sigma_2 = (n-m+1, n-m+2, \dots, n)$ which contain the items with the m smallest and the m largest priority values, one decreasing subsequence $\sigma_3 = (n-m, n-m-1, \dots, 2m+1)$ and again one increasing subsequence $\sigma_4 = (m+1, m+2, \dots, 2m)$. Since σ_1 and σ_2 together constitute a longest increasing subsequence, $|\sigma_1| + |\sigma_2| - m = m$ is a lower bound on US_{adj} according to Lemma 2.1.

Now we construct a solution c^* achieving this lower bound value. At first we put all m items of σ_1 to the lowest level of the stacks, afterwards all m items of σ_2 above them in the second lowest level. Then we place all items of σ_3 in an arbitrary order into the stacks without occupying the topmost level. Finally, the last m items from σ_4 are put to the topmost level in the stacks. This leads to a configuration c^* with m unordered stackings occurring between the items in the lowest and the second lowest level. According to the lower bound, this solution must be optimal w.r.t. US_{adj}.

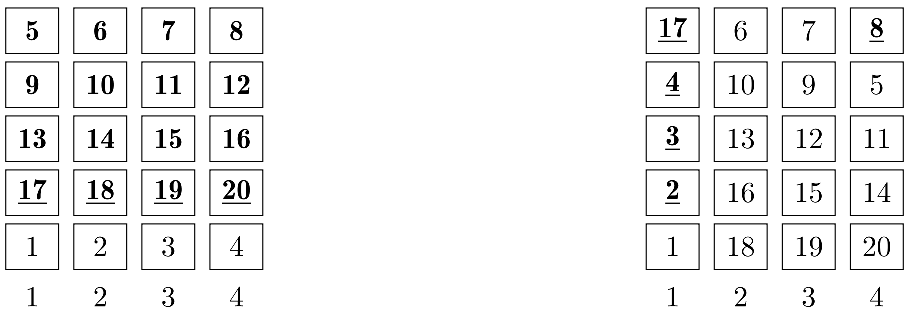
On the other hand, the number of badly placed items in such a configuration is $\mathrm{BI}(c^*) = m(b-1)$ because the m items $i \in \sigma_1$ with the m smallest priority values are placed in the lowest level of the stacks and all $m(b-1)$ remaining items j stored above satisfy $p_j > p_i$. □

Example 2.4. *As example for the above construction, consider the case $m = 4, b = 5$ with $n = 20$ items and the arrival sequence*

$$\pi^{in} = (1, 2, 3, 4|17, 18, 19, 20|16, 15, 14, 13, 12, 11, 10, 9|5, 6, 7, 8).$$

An optimal configuration c^* is shown in Figure 2.6a. It has $\mathrm{US}_{adj}(c^*) = m = 4$ unordered stackings, but $\mathrm{BI}(c^*) = m(b-1) = 16$ badly placed items.

However, if we optimize w.r.t. BI instead of US_{adj}, an optimal configuration $\hat{c}^*$ has $\mathrm{BI}(\hat{c}^*) = \mathrm{US}_{adj}(\hat{c}^*) = m + 1 = 5$ (cf. Figure 2.6b). The lower bound according to Lemma 2.2 gives only the value m because the first increasing subsequence σ_1 combined with σ_2 is a longest increasing subsequence of length $2m$. The last increasing subsequence σ_4 cannot be combined with σ_1 or σ_2 using Lemma 2.2, and σ_4 has length m, which does not increase the lower bound. But in this special case the item with the smallest priority value (item 1) is part of the first increasing subsequence. Item 1 must be placed in some stack and conflicts with all items in σ_4. This means that either stacking one item of σ_4 in the same stack as item 1 results in one badly placed item or the stack of item 1 must be avoided by all items of σ_4, and all m items of this increasing



(a) Configuration c^* , optimized w.r.t. US_{adj} has $US_{adj}(c^*) = 4$ and $BI(c^*) = 16$.

(b) Configuration $\hat{c}^*$, optimized w.r.t. BI has $BI(\hat{c}^*) = US_{adj}(\hat{c}^*) = 5$.

Figure 2.6: Solutions for Example 2.4, unordered stackings underlined, badly placed items in bold.

subsequence must be stacked in $m - 1$ *stacks, which also results in one badly placed item. Thus,* $BI(\hat{c}^*) = m + 1$ *is optimal.*

This shows that the constructed instances do not necessarily lead to configurations with a large number of badly placed items (as in the case when optimizing w.r.t. the simpler objective function US_{adj}*), but that the optimization criterion is important.* □

Due to Lemma 2.3, optimizing w.r.t. US_{adj} can lead to solutions where the absolute difference to BI is $m(b - 2)$, i.e., the difference between these two surrogate objective functions may be arbitrarily large. Since BI is only a lower bound on the actual number of reshuffles, in the worst case a solution optimized w.r.t. US_{adj} may need a much larger number of reshuffles than indicated by its objective value. How much larger the number of reshuffles can be, will be discussed in the following.

Lemma 2.4. *For any feasible configuration* c *let* $\mathcal{B} \subset I$ *be the set of badly placed items and* $l_i \in \{1, \dots, b\}$ *be the level of item* $i \in \mathcal{B}$ *in* c*. Then all items can be retrieved with at most* $\sum_{i \in \mathcal{B}} 2(l_i - 1)$ *restricted relocations.*

Proof: First we prove the result for the unrestricted BRP by constructing a sequence of moves which needs at most $\sum_{i \in \mathcal{B}} 2 \cdot (l_i - 1)$ relocations. In the restricted and unrestricted BRP, every item $j \in I$ must be retrieved from the storage. If item j in stack q and level l is the next item to be retrieved, let $\mathcal{B}_j$ be the set of badly placed items located above j at levels $l + 1, \dots, l + |\mathcal{B}_j|$ in stack q (this set may also be empty). All these items $\mathcal{B}_j$ have to be relocated to another stack before j can be retrieved. In the unrestricted BRP, afterwards, we are allowed to relocate all of these items back to stack q. As a result, the retrieved item j is now missing in stack q and all items $\mathcal{B}_j$ are located at levels $l, \dots, l + |\mathcal{B}_j| - 1$, i.e., their levels are decreased by one.

As an example, consider the first two retrievals for the storage shown in Figure 2.7. At first, item 1 with $\mathcal{B}_1 = \{7\}$ has to be retrieved. After relocating item 7 from the first to the third stack and retrieving item 1, we push item 7 back to the first stack. Then, item 2 with $\mathcal{B}_2 = \{5, 6\}$ has to be retrieved. After relocating items $6, 5$ to other stacks and retrieving item 2, we push items $5, 6$ back to the second stack.

If we consider the complete retrieval process, for every badly placed item $i \in \mathcal{B} = \cup_{j=1}^{n} \mathcal{B}_j$ at most $2 \cdot (l_i - 1)$ relocations are induced by at most $l_i - 1$ retrievals of items below, before item i reaches the bottom of its initial stack. However, item i may also be retrieved earlier, which decreases the number of necessary reshuffles even further.

Now we prove the result for the restricted BRP where in each iteration only items above the next item to be retrieved may be relocated. The previous approach uses the freedom of

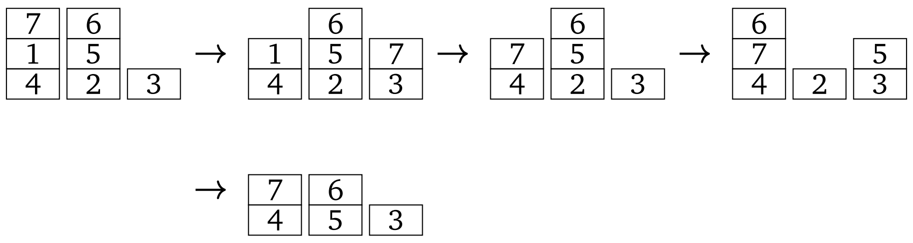


Figure 2.7: Example for pushing back badly placed items in the unrestricted BRP.

the unrestricted BRP to relocate all items back to their initial stack after one item is retrieved, which is not allowed in the restricted BRP.

Nevertheless, only badly placed items are relocated, while all well placed items are retrieved from their initial position. If we look at one specific badly placed item $i \in \mathcal{B}_j$, then this item is initially located in some stack and after every second relocation of i, it returns back to this stack. Let $\theta_{i,j}$ be the corresponding slot of item $i \in \mathcal{B}_j$ after the retrieval of item j and pushing back item i in the unrestricted BRP. Furthermore, we denote by $\mathcal{T}_i$ the set of initially well placed items below item i.

An important observation of the proof for the unrestricted BRP is that every badly placed item i can be relocated to another stack and pushed back until item i reaches the bottom of its initial stack where it remains at its final position before it is retrieved. This means that the number of necessary relocations of item i is independent from all other item priorities and depends only on l_i. Thus, all badly placed items can be seen as "equivalent" items with the property that these items block all other items below, regardless of their locations. To "simulate" the algorithm for the unrestricted BRP in the restricted setting, we claim that after the retrieval of an item j it is possible to move a badly placed item i to another slot $\theta_{h,k}$ of items $h \in \mathcal{B}, k \in I$ instead of pushing item i back to the slot $\theta_{i,j}$.

Before proving this claim, we again look at the example from Figure 2.7, now considered in the restricted setting (cf. Figure 2.8). At first, again item 7 is relocated from the first to the third stack (where it stays), and item 1 is retrieved. To retrieve item 2, we move item 6 to the slot $\theta_{7,1}$ used by item 7 in the first stack in the unrestricted BRP and afterwards relocate item 5 to the third stack. After the retrieval of item 2, items $5, 7$ are relocated to the second stack, where item 5 is relocated to slot $\theta_{5,2}$ and item 7 to slot $\theta_{6,2}$. Compared to the unrestricted setting, we now have the same situation, only items 6 and 7 changed their positions. We see that for this example, the modified approach does not require more relocations than pushing back all items immediately.

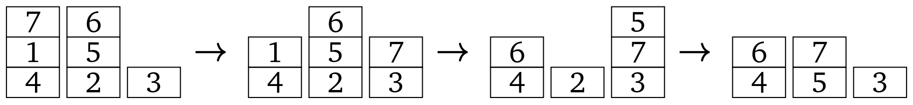


Figure 2.8: Example for exchanging positions of badly placed items in the restricted BRP.

We claim that at least every second relocation of an item i can be performed to a slot $\theta_{h,k}$ of an item $h \in \mathcal{B}_k$ after the retrieval of an item k. This is true because after every retrieval of an item k, every badly placed item h defines a slot $\theta_{h,k}$ where it was pushed back in the algorithm for the unrestricted setting. Thus, there are always as many θ-slots of badly placed items as badly placed items that must be relocated to such a slot.

Based on this observation, we prove that any two items $i \neq h$ belonging to different stacks can change the slots $\theta_{i,j}, \theta_{h,k}$ after the retrieval of items j, k without increasing the number of reshuffles. We count every relocation of an item i by its destination slot $\theta_{h,k}$. Let $\mathbb{1}_{(i,\theta_{h,k})} = 1$

if item i is pushed to $\theta_{h,k}$ and zero otherwise. Then, $\sum_{k\in\mathcal{T}_h} \mathbb{1}_{(i,\theta_{h,k})}$ is the number of all relocations of item i to any slot $\theta_{h,k}$ of item h. Since at least every second relocation of any badly placed item i may go to a slot $\theta_{h,k}$ of a badly placed item h, the total number of all relocations of item i cannot exceed $\sum_{h\in\mathcal{B}} 2 \cdot \sum_{k\in\mathcal{T}_h} \mathbb{1}_{(i,\theta_{h,k})}$. Thus, we are able to bound the maximum number of relocations by

$$\mathrm{RS}_r \leqslant \sum_{i\in\mathcal{B}} \left(\sum_{h\in\mathcal{B}} 2 \sum_{k\in\mathcal{T}_h} \mathbb{1}_{(i,\theta_{h,k})} \right) = \sum_{h\in\mathcal{B}} 2 \left(\sum_{i\in\mathcal{B}} \sum_{k\in\mathcal{T}_h} \mathbb{1}_{(i,\theta_{h,k})} \right).$$

The left side of the equation includes the sum over all item $i \in \mathcal{B}$ and counts the number of relocations of item i. The right side of the equation sums over all items $h \in \mathcal{B}$ and counts the number of all relocations to any slot $\theta_{h,k}$ of item h. An item located at a slot $\theta_{h,k}$ is relocated exactly once and there are at most $l_h - 1$ slots for an item h because of at most $l_h - 1$ items with a smaller priority value below. We conclude that at most $l_h - 1$ relocations are possible. Thus, $\sum_{i\in\mathcal{B}} \sum_{k\in\mathcal{T}_h} \mathbb{1}_{(i,\theta_{h,k})} \leqslant l_h - 1$, which implies

$$\mathrm{RS}_r \leqslant \sum_{h\in\mathcal{B}} 2(l_h - 1). \tag{2.3}$$

□

If we consider the relationship between the number of restricted reshuffles and the total number of badly placed items, due to $l_h \leqslant b$ for all $h \in \mathcal{B}$ and $|\mathcal{B}| = \mathrm{BI}(c)$, condition (2.3) implies

$$\frac{\mathrm{RS}_r(c)}{\mathrm{BI}(c)} \leqslant 2(b-1) \text{ for all configurations } c. \tag{2.4}$$

In the following, we present an example showing that this bound can asymptotically be reached, even for configurations optimized w.r.t. BI.

Example 2.5. *Consider the following family of instances with $n = m \cdot b$, $p_i = i$ for $i = 1, \dots, n$, and arrival sequence*

$$\begin{aligned} \pi^{in} = (&n - 2m + 2, \dots, n - m + 1, && (2.5)\\ &n - 3m + 2, \dots, n - 2m + 1, && (2.6)\\ &\dots, \\ &m + 2, \dots, 2m + 1, && (2.7)\\ &2, \dots, m + 1, && (2.8)\\ &n - m + 2, \dots, n, && (2.9)\\ &1). && (2.10)\end{aligned}$$

This arrival sequence consists of $b - 1$ increasing subsequences (2.5)–(2.8) of length m with priorities $2, \dots, n - m + 1$, one increasing subsequence (2.9) of length $m - 1$ with the largest $m - 1$ priority values and ends with the item with priority value 1. The increasing subsequence (2.9) together with any increasing subsequence from (2.5)–(2.8) builds a longest increasing subsequence of length $2m - 1$, i.e., according to Lemma 2.1, the value $m - 1$ is a lower bound on BI. We construct a solution c achieving this lower bound value.

All items of the increasing subsequences with length m can be stacked in the m available stacks without resulting in any badly placed item by choosing different stacks for these items in each subsequence. Furthermore, the items of (2.9) must be stacked as badly placed items at the topmost level in the stacks, which leads to $l_i = b$ for all $i \in \{n - m + 2, \dots, n\}$ and

$\mathrm{BI}(c) = m - 1$.

For this configuration c*, the optimal number of restricted reshuffles is*

$$\mathrm{RS}_r(c) = (b-1)(\mathrm{BI}(c)+1)$$

since each badly placed item $n-m+2, \ldots, n$ *has to move* $b-1$ *times (for each item in its stack it moves one level down). Additionally,* $b-1$ *moves must be performed because every time all badly placed items are located at the same level, one item must be moved from stack 1 to stack* m *to free the next item to be retrieved. Since* $(b-1)(\mathrm{BI}(c)+1) \in \Theta(2(b-1)\mathrm{BI}(c))$*, the bound in* (2.4) *can asymptotically be reached.*

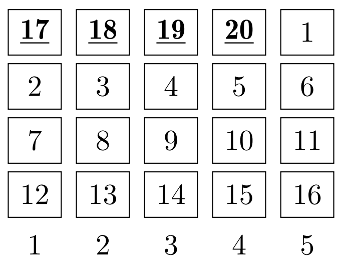


Figure 2.9: Configuration c from Example 2.5 with $m = 5, b = 4$.

In Figure 2.9, we illustrate the case $m = 5, b = 4$ *with* $n = 20$ *and the arrival sequence*

$$\pi^{in} = (12, 13, 14, 15, 16|7, 8, 9, 10, 11|2, 3, 4, 5, 6|17, 18, 19, 20|1).$$

An optimal configuration c *has* $\mathrm{BI}(c) = m - 1 = 4$ *badly placed items. Item 1 is retrieved immediately, then item 17 moves to stack 5 and item 2 is retrieved. Afterwards, items* $18, 19, 20, 17$ *have to move one level down, and this procedure repeats until all badly placed items reach the bottom of a stack. In total,* $\mathrm{RS}_r(c) = (b-1)m = 15$ *restricted reshuffles are necessary to retrieve all items.* □

In the following, we consider the relationship between the number of restricted reshuffles and the total number of unordered stackings. From equations (2.1) and (2.2) in the proof of Lemma 2.3, we obtain

$$\frac{\mathrm{BI}(c)}{\mathrm{US}_{adj}(c)} \leqslant b - 1 \text{ for all configurations } c, \tag{2.11}$$

i.e., $\mathrm{BI}(c) \leqslant (b-1)\mathrm{US}_{adj}(c)$, and from equation (2.4) follows

$$\mathrm{RS}_r(c) \leqslant 2(b-1)\mathrm{BI}(c) \leqslant 2(b-1)^2\mathrm{US}_{adj}(c). \tag{2.12}$$

In the following, we present an example showing that this bound can asymptotically be reached, even for configurations optimized w.r.t. US_{adj}.

Example 2.6. *Consider the following family of instances with* $n = m \cdot b$*,* b *an even number,* $p_i = i$ *for* $i = 1, \ldots, n$*, and arrival sequence*

$$\pi^{in} = (m\left(\frac{b}{2}-1\right)+1+\frac{b}{2}, \ldots, m\frac{b}{2}+\frac{b}{2}, \tag{2.13}$$

$$m\left(\frac{b}{2}-2\right)+1+\frac{b}{2}, \ldots, m\left(\frac{b}{2}-1\right)+\frac{b}{2}, \tag{2.14}$$

$$\ldots,$$

$$m+1+\frac{b}{2},\dots,2m+\frac{b}{2}, \tag{2.15}$$
$$1+\frac{b}{2},\dots,m+\frac{b}{2}, \tag{2.16}$$
$$\frac{b}{2},\dots,1, \tag{2.17}$$
$$mb-(m-1)+1,\dots,mb \tag{2.18}$$
$$mb-2(m-1)+1,\dots,mb-(m-1) \tag{2.19}$$
$$\dots,$$
$$mb-\left(\frac{b}{2}-1\right)(m-1)+1,\dots,mb-\left(\frac{b}{2}-2\right)(m-1) \tag{2.20}$$
$$mb-\frac{b}{2}(m-1)+1,\dots,mb-\left(\frac{b}{2}-1\right)(m-1)\Big). \tag{2.21}$$

The sequence π^{in} consists of three parts. The first part (2.13)–(2.16) consists of $\frac{b}{2}$ increasing subsequences $\sigma_1,\dots,\sigma_{\frac{b}{2}}$ of length m containing the priority values $\frac{b}{2}+1,\dots,(m+1)\frac{b}{2}$. The p_i-values of all items in a subsequence σ_v are always smaller than those of all items in the subsequences $\sigma_1,\dots,\sigma_{v-1}$. The second part (2.17) contains the items $1,\dots,\frac{b}{2}$ which can be retrieved without any relocations. The last part (2.18)–(2.21) consists of $\frac{b}{2}$ increasing subsequences $\hat{\sigma}_1,\dots,\hat{\sigma}_{\frac{b}{2}}$ of length $m-1$ containing the priority values $(m+1)\frac{b}{2}+1,\dots,mb$. Similar to the first part, all items in a subsequence $\hat{\sigma}_v$ have smaller p_i-values than all items in the subsequences $\hat{\sigma}_1,\dots,\hat{\sigma}_{v-1}$.

Any subsequence of the first part together with any subsequence of the third part builds a longest increasing subsequence, i.e., according to Lemma 2.1, the value $m+(m-1)-m=m-1$ is a lower bound on US_{adj}.

If we optimize w.r.t. US_{adj}, we get an optimal configuration c where every item of the first increasing subsequence of the last part $\hat{\sigma}_1$ results in an unordered stacking. All other items of the increasing subsequences $\hat{\sigma}_2,\dots,\hat{\sigma}_{\frac{b}{2}}$ are badly placed but do not lead to any more unordered stackings. Obviously, every badly placed item of the increasing subsequences $\hat{\sigma}_1,\dots,\hat{\sigma}_{\frac{b}{2}}$ has to move $\frac{b}{2}$ times since $\frac{b}{2}$ well placed items must be retrieved from each stack. Thus, for such a configuration c, the optimal number of restricted reshuffles $RS_r(c)$ is at least $\frac{b^2}{4}US_{adj}(c)$. Since $\frac{b^2}{4}US_{adj}(c)\in\Theta(2(b-1)^2US_{adj}(c))$, the bound in (2.12) can asymptotically be reached.

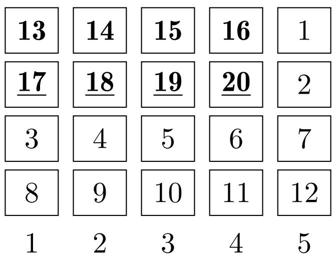


Figure 2.10: Configuration c from Example 2.6 with $m=5, b=4$.

In Figure 2.10, we illustrate the case $m=5, b=4$ with $n=20$ and the arrival sequence

$$\pi^{in}=(8,9,10,11,12|3,4,5,6,7|2,1|17,18,19,20|13,14,15,16).$$

An optimal configuration c has $US_{adj}(c)=m-1=4$. Items $1,2$ can immediately be retrieved, then items $13,17$ move to stack 5, and item 3 is retrieved. Afterwards, items $14,18,15,19,16,20$, and $17,13$ move one level down, and this procedure repeats until all badly placed items reach the *bottom level of a stack. In total,* $RS_r(c) = 20$ *restricted reshuffles are necessary to retrieve all items.* □

Analyzing the relationship between badly placed items/unordered stackings and unrestricted reshuffles in a similar way as for restricted reshuffles, seems to be much more difficult since allowing also voluntary moves increases the complexity of the problem drastically. We leave this for further research.

2.5 Solution approaches for the PSLP

This section is devoted to solution approaches for the PSLP with different objective functions. At first, we describe a MIP formulation from Boysen and Emde [15] to solve the PSLP with the surrogate objective function US_{adj}. Afterwards, we present a new MIP formulation for the objective function BI and a SA algorithm trying to minimize the actual number of reshuffles RS.

2.5.1 Objective US_{adj}

For the objective function US_{adj}, we use the formulation PSLP-IP2 of Boysen and Emde [15]. W.l.o.g. assume that the jobs are numbered according to their arrivals, i.e., $\pi^{in} = (1, 2, \dots, n)$. Besides the items in $I = \{1, \dots, n\}$, we need two dummy elements $0, n+1$, representing the bottom and top of the stacks. For all $i, j \in \{0, \dots, n+1\}$ with $i > j$ we have binary variables

$$x_{ij} = \begin{cases} 1, & \text{if item } i \text{ is stacked directly on top of item } j \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, for each $i \in \{0, \dots, n\}$ there is a variable $l_i \geqslant 0$ indicating the level where item i is stored. Let $\mathcal{A}^u := \{(i,j) \in I \times I, i > j, p_i > p_j\}$ be the set of all item pairs (i, j) where i can be stacked directly on top of j, but which leads to an unordered stacking. Then, the MIP reads as follows.

$$\begin{aligned}
(\mathrm{IP}_{US}) \quad \min \quad & \sum_{(i,j)\in\mathcal{A}^u} x_{ij} && && (2.22)\\
\text{s.t.} \quad & \sum_{j=0}^{i-1} x_{ij} = 1 && \forall i \in I && (2.23)\\
& \sum_{i=j+1}^{n+1} x_{ij} = 1 && \forall j \in I && (2.24)\\
& \sum_{i=1}^{n} x_{i0} \leqslant m && && (2.25)\\
l_j + 1 - b(1 - x_{ij}) & \leqslant l_i && \forall i \in I, j \in \{0, \dots, i-1\} && (2.26)\\
1 \leqslant l_i & \leqslant b && \forall i \in I && (2.27)\\
l_0 & = 0 && && (2.28)\\
x_{ij} & \in \{0, 1\} && \forall i \in \{1, \dots, n+1\}, j \in \{0, \dots, i-1\} && (2.29)
\end{aligned}$$

In the objective (2.22), the total number of unordered stackings of adjacent items is minimized. Due to (2.23) each item i is stacked on top of another item $j < i$ or on the dummy element 0, indicating the bottom level. Similarly, constraints (2.24) ensure that on each item j another item $i > j$ (or the dummy element $n + 1$, indicating the top level) is stacked. Constraint

(2.25) guarantees that at most m stacks are used. In constraints (2.26), the variables x_{ij} and the levels l_i are linked, by requiring $l_i \geqslant l_j + 1$ if i is stacked on top of j. Note that l_i may be chosen larger than necessary, i.e., this value is only an upper bound for the actual level of i. However, as long as (2.27) (guaranteeing that each stack contains at most b items) is satisfied, this does not lead to any problems. Furthermore, no integrality constraints are required for the variables l_i, i.e., they may be introduced as continuous variables in (2.27). Finally, (2.28) sets the level of the bottom item 0 to 0. The arrival sequence π^{in} is implicitly taken into account since the variables x_{ij} are only defined for $i > j$.

2.5.2 Objective BI

Now we consider the objective function BI and again w.l.o.g. assume $\pi^{in} = (1, 2, \dots, n)$. In contrast to the previous MIP formulation for US_{adj}, we do not need variables indicating the levels of the items. If an assignment of items to a stack is determined, due to the arrival sequence π^{in} all items in the stack can only be ordered in one feasible way. For each item $i \in I$ let $B_i := \{j \in I \mid j < i, p_j < p_i\}$ be the set of all items $j < i$ (meaning that i can be stacked on top of j) with a smaller priority value. Thus, if an item $j \in B_i$ is placed below i in the same stack, item i becomes badly placed.

For all $i \in I, q \in \mathcal{Q}$ we introduce binary variables

$$\beta_{iq} = \begin{cases} 1, & \text{if item } i \text{ is put into stack } q \text{ as a badly placed item} \\ 0, & \text{otherwise} \end{cases}$$

$$\alpha_{iq} = \begin{cases} 1, & \text{if item } i \text{ is put into stack } q \text{ as a well placed item} \\ 0, & \text{otherwise.} \end{cases}$$

Then, our new *integer program* (IP) reads as follows.

$$(\mathrm{IP_{BI}}) \quad \min \sum_{i \in I} \sum_{q \in \mathcal{Q}} \beta_{iq} \tag{2.30}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{Q}} (\beta_{iq} + \alpha_{iq}) = 1 \qquad \forall i \in I \tag{2.31}$$

$$\sum_{i \in I} (\beta_{iq} + \alpha_{iq}) \leqslant b \qquad \forall q \in \mathcal{Q} \tag{2.32}$$

$$\beta_{jq} + \alpha_{jq} + \alpha_{iq} \leqslant 1 \qquad \forall i \in I, q \in \mathcal{Q}, j \in B_i \tag{2.33}$$

$$\beta_{iq}, \alpha_{iq} \in \{0, 1\} \qquad \forall i \in I, q \in \mathcal{Q} \tag{2.34}$$

In the objective (2.30), the number of badly placed items is minimized. Due to (2.31) each item is assigned to exactly one stack and due to (2.32) at most b items are assigned to each stack. Constraints (2.33) force $\alpha_{iq} = 0$ if an item $j \in B_i$ exists for which $\beta_{jq} + \alpha_{jq} = 1$. This means that item i cannot be well placed if an item $j \in B_i$ is assigned to the same stack (which must be put below i and has $p_j < p_i$ due to the definition of B_i).

2.5.3 Objectives RS_r and RS_u

As already mentioned above, finding a configuration in the loading process for which the number of reshuffles in the unloading process is minimized, is a very complex problem. Hence, solving the combined loading and unloading problem with objective RS exactly, seems to be intractable for larger instances. For a fixed storage configuration, in the unloading process, the BRP arises as subproblem which is known to be difficult to solve even for small instances (cf. Caserta et al. [19]).

Therefore, we decided to use a heuristic approach for the combined loading/unloading problem and implemented a SA algorithm that tries to find a storage configuration for which the total number of reshuffles in the unloading stage is small. Our heuristic proceeds in two stages. While in the outer stage the current storage configuration is changed, in the inner stage, the new configuration is evaluated by solving the corresponding BRP instance with some (fast) BRP solver. Then, the new configuration is accepted based on a standard simulated annealing acceptance criterion.

We start with a randomly generated starting configuration or a solution obtained by an algorithm for the PSLP. To represent solutions, as in the IP from Section 2.5.2, we take advantage of the fact that each subset of items assigned to the same stack has an unique ordering due to the fixed arrival sequence π^{in}. Thus, we can represent a configuration simply by disjoint subsets $(I_1, \dots, I_m)$ where $I_q \subset I$ is the subset of items assigned to stack q. For each item i we denote by $q(i)$ the current stack it is assigned to. To derive feasible levels of the items in the stacks, each set I_q has only to be sorted w.r.t. the arrival sequence.

To generate neighbors of a configuration in the outer stage, we used a combined "shift" and "swap" neighborhood. In this context, a "shift" means that an item i and a new target stack $q \neq q(i)$ with at least one empty slot are chosen. Then, i is removed from $q(i)$ and assigned to q. A "swap" means that two items i, j currently assigned to different stacks $q(i) \neq q(j)$ are chosen. Then i is shifted from $q(i)$ to $q(j)$, and j is shifted from $q(j)$ to $q(i)$. Together, both operators define a neighborhood that is connected, i.e., every solution $(I'_1, \dots, I'_m)$ can be reached from any other solution $(I_1, \dots, I_m)$ by a finite number of moves in the neighborhood. To achieve this, at first items are shifted until $|I_q| = |I'_q|$ holds for each stack $q \in \mathcal{Q}$. Afterwards, items are swapped until all items are in the desired stack, i.e., $I_q = I'_q$ for all $q \in \mathcal{Q}$.

To evaluate the generated configurations, in the inner stage of the heuristic, we used (re-implemented) heuristics known from the literature for the restricted/unrestricted BRP. While the unrestricted variant is solved with the bottom level heuristic of Jin et al. [57], the restricted variant is solved with the look-ahead heuristic of Petering and Hussein [85] using a look-ahead value of one, which restricts the solver to the restricted BRP. These heuristics were chosen because of their high solution quality combined with a very fast runtime, which is advantageous to evaluate a large number of configurations in the outer stage. We also tested the (exact) state-of-the-art *iterative deepening A** (IDA) algorithm of Tanaka and Mizuno [96] to evaluate configurations in the inner stage, but the results were worse (since within the same time limit less configurations could be generated). However, the final solution was always exactly evaluated with the IDA-algorithm.

2.6 Computational study

In this section, we report results of our computational study. We used an Intel(R) Core(TM) i7-2600 CPU with 3.4 GHz and 10 GB RAM. Both MIPs were solved with CPLEX 12.6.1, the SA algorithm was implemented in Java.

For our experiments, we used test data of Boysen and Emde [15] for the PSLP with $n \in \{30, 120, 500\}$. Additionally, we generated further instances with $n \in \{40, 60\}$ from the instances with 120 items by using only the first 40 and 60 items, respectively. Furthermore, we created instances with $n = 40$ and duplicate priorities. For this purpose, we introduced a parameter d, denoting the number of items with the same priority value. Then, from the original priorities p_i new priorities p'_i were derived by setting $p'_i := \lfloor p_i/d \rfloor$. The used test instances and all our results can be found at `http://www2.informatik.uos.de/kombopt/data/pslp/`.

2.6.1 MIP formulations

In a first experiment, we solved each instance with the MIP formulations $(\mathrm{IP_{US}})$ and $(\mathrm{IP_{BI}})$ for the objective functions US_{adj} and BI imposing a time limit of 30 minutes. Table 2.1 summarizes the obtained results for the smaller instances with $n \in \{30, 40, 60\}$ and the larger ones with $n = 120$. The large instances with $n = 500$ could not be solved by the MIPs within 30 minutes, for them, the MIPs do not even provide a feasible solution. In each row, we report average values over 20 instances with the same storage parameters: the number of items n, the number of stacks m, and the maximum stack height b, set to $b = \lceil n/m \rceil$ as in Boysen and Emde [15]. We call all instances with the same parameters an “instance group”. If the value n is marked with a $*$, the storage area is not completely filled, i.e., $n < mb$.

The columns US^*_{adj} and BI^* show the average objective values for unordered stackings and badly placed items obtained by the MIPs. Furthermore, we indicate for how many instances a feasible solution could be found, how many instances could be verified to be optimally solved within the time limit, and report the average computation times (in seconds). While the new IP formulation $\mathrm{IP_{BI}}$ was able to find a feasible solution for every instance, the formulation $\mathrm{IP_{US}}$ could not find a feasible solution within the time limit for 11 instances with $n = 120$ and $m \in \{12, 15, 20\}$. For a fair comparison, these instances were not considered in the average values (for US_{adj} and BI) and also eliminated for further investigations.

Table 2.1: Solutions optimized w.r.t. US_{adj} and BI.

n	m	b	US^*_{adj}	feas	ver	time	BI^*	feas	ver	time
30	5	6	4.00	20	20	1.4	5.65	20	20	0.1
30	6	5	2.90	20	20	0.1	3.60	20	20	0.2
*30	8	4	1.05	20	20	0.1	1.05	20	20	0.1
30	10	3	0.40	20	20	9.9	0.40	20	20	0.2
40	5	8	4.90	20	20	0.2	9.50	20	20	0.5
*40	7	6	2.70	20	20	0.4	3.85	20	20	1.2
40	8	5	1.80	20	20	2.5	2.40	20	20	0.9
40	10	4	0.75	20	20	254.0	0.90	20	20	0.8
60	6	10	6.80	20	20	0.9	14.20	20	20	3.4
60	10	6	2.40	20	18	183.8	3.35	20	20	19.3
60	12	5	1.05	20	13	664.3	1.40	20	20	22.5
60	15	4	0.50	20	11	811.5	0.60	20	20	2.9
60	20	3	0.15	20	17	270.9	0.20	20	20	1.3
			2.26	260	239	169.2	3.62	260	260	4.1
120	5	24	18.40	20	20	6.1	57.90	20	20	47.0
120	8	15	11.70	20	20	31.8	33.00	20	20	205.9
120	10	12	9.05	20	20	306.1	21.80	20	19	602.8
120	12	10	7.05	19	19	600.1	14.42	20	9	1531.5
120	15	8	4.77	13	10	1255.9	6.69	20	1	1800.0
120	20	6	1.29	17	3	1617.8	1.76	20	13	1045.5
120	24	5	0.65	20	9	1120.4	0.80	20	17	439.4
120	30	4	0.40	20	13	679.3	0.40	20	20	146.4
120	40	3	0.05	20	19	109.3	0.10	20	20	17.8
			5.93	169	133	636.3	15.21	180	139	648.5

As it can be seen from the table, the new formulation $\mathrm{IP_{BI}}$ performs much better than

$\mathrm{IP_{US}}$. On the one hand, more solutions are verified to be optimally solved (399 versus 372 from 440). On the other hand, also the computational times are smaller. The instances with $n \in \{30, 40, 60\}$ could be solved in 4.1 seconds on average (at most 300 seconds), while $\mathrm{IP_{US}}$ needs 169.2 seconds on average and reached the time limit of 1800 seconds several times. For the larger instances with $n = 120$, the average computation times for both formulations are similar. While $\mathrm{IP_{US}}$ is faster for $b \geqslant 8$, $\mathrm{IP_{BI}}$ is better for $b \leqslant 6$. The runtimes of the MIP formulations increase with the number of items n. Furthermore, a very large or small stack height b leads to faster runtimes, while combinations of a medium stack height with a medium number of stacks need longer.

As expected, the results show that for larger stack heights b (which imply a smaller number of stacks m if n is fixed), the values of US^*_{adj} and BI^* are also larger. This is due to the fact that the items can be distributed in a more flexible way if more stacks are available. If we have a look at the absolute differences between the values US^*_{adj} and BI^*, we see that for the instances with $n \in \{30, 40, 60\}$, on average $\mathrm{BI}^* - \mathrm{US}^*_{adj} = 1.36$; for the instances with $n = 120$, the average deviation is already 9.28 (maximum 47 for an instance with $n = 120$ and $b = 24$). Hence, perhaps a little bit surprising, the new IP formulation $\mathrm{IP_{BI}}$ provides much better lower bounds for the actual number of reshuffles in a shorter amount of time.

In a second experiment, to evaluate the MIP solutions w.r.t. the actual number of reshuffles, we used them as initial storage configurations for the BRP. Then we solved the corresponding unloading problems minimizing RS_r and RS_u with the state-of-the-art IDA-algorithm of Tanaka and Mizuno [96] for the restricted and unrestricted version of the BRP imposing a time limit of two hours. In order to avoid infeasible situations (it may happen that there is not enough free space to perform the necessary relocations inside the storage area), we always added one additional empty stack to the storage before solving the BRP.

In Table 2.2, the results are reported. The column “inst” indicates how many instances were considered in the corresponding instance group (as mentioned above, we used only the 429 instances for which both MIP formulations found a feasible solution). The values $\Delta\mathrm{RS}_u(\mathrm{US}^*_{adj})$ and $\Delta\mathrm{RS}_r(\mathrm{US}^*_{adj})$ are the absolute differences between the numbers of reshuffles and US^*_{adj} (unrestricted and restricted version, respectively). Analogously, the values $\Delta\mathrm{RS}_u(\mathrm{BI}^*)$ and $\Delta\mathrm{RS}_r(\mathrm{BI}^*)$ are the differences between the number of reshuffles and BI^*. Furthermore, in brackets, we denote for how many instances optimal solutions could be verified by the IDA-algorithm within two hours. It can be seen that most solutions could be verified. As expected, the unrestricted BRP is more difficult to solve than the restricted BRP. Furthermore, the solutions optimized w.r.t. BI are easier to solve than those optimized w.r.t. US_{adj}.

We can see that the average differences $\Delta\mathrm{RS}_{u/r}(\mathrm{US}^*_{adj}/\mathrm{BI}^*)$ increase when the values US^*_{adj} and BI^* increase. This can be explained by the fact that the computed storage configurations (which define the corresponding BRP) have a greater complexity in terms of reshuffles if these values are large. Thus, this higher complexity makes it hard to solve the BRP with the IDA-algorithm and the number of optimally solved instances reduces as well. On the other hand, instances with $\mathrm{BI}^* = 0$ can usually be solved very fast.

For instances with $b \leqslant 6$ (reflecting the common situation of real-world container terminals), the average value of $\Delta\mathrm{RS}_u(\mathrm{BI}^*)$ is at most 0.65, while the maximum over all instances is 3. The average value of $\Delta\mathrm{RS}_u(\mathrm{US}^*_{adj})$ is at most 10.25, the maximum is 20. The values for the restricted BRP are overall larger, but the situation is similar. The average value of $\Delta\mathrm{RS}_r(\mathrm{BI}^*)$ is at most 1.65 (maximum 5), while $\Delta\mathrm{RS}_r(\mathrm{US}^*_{adj})$ is at most 13.10 (maximum 25). This indicates that in the case $b \leqslant 6$ the number of badly placed items is a good lower bound and estimates the actual number of reshuffles much better than the number of unordered stackings.

However, for the most difficult instance groups with $n = 120$ and $b \in \{12, 15, 24\}$ the

Table 2.2: Actual reshuffles for solutions optimized w.r.t. US_{adj} and BI.

n	m	b	inst	$\Delta RS_u(US^*_{adj})$		$\Delta RS_r(US^*_{adj})$		$\Delta RS_u(BI^*)$		$\Delta RS_r(BI^*)$	
30	5	6	20	10.25	(20)	13.10	(20)	0.65	(20)	1.65	(20)
30	6	5	20	6.55	(20)	7.85	(20)	0.50	(20)	1.10	(20)
*30	8	4	20	1.25	(20)	1.25	(20)	0	(20)	0.05	(20)
30	10	3	20	0.20	(20)	0.20	(20)	0	(20)	0	(20)
40	5	8	20	24.00	(9)	28.40	(20)	1.50	(20)	3.35	(20)
*40	7	6	20	8.20	(20)	8.90	(20)	0	(20)	0.20	(20)
40	8	5	20	4.25	(20)	4.90	(20)	0.05	(20)	0.05	(20)
40	10	4	20	0.95	(20)	0.95	(20)	0	(20)	0	(20)
60	6	10	20	42.50	(0)	50.25	(16)	3.65	(20)	8.30	(20)
60	10	6	20	8.70	(19)	9.50	(20)	0.25	(20)	0.45	(20)
60	12	5	20	2.80	(20)	2.90	(20)	0.15	(20)	0.20	(20)
60	15	4	20	0.75	(20)	0.75	(20)	0.05	(20)	0.05	(20)
60	20	3	20	0.10	(20)	0.10	(20)	0	(20)	0	(20)
120	5	24	20	263.75	(0)	286.15	(0)	66.20	(0)	94.15	(0)
120	8	15	20	151.35	(0)	175.65	(0)	10.50	(8)	24.50	(14)
120	10	12	20	94.65	(0)	111.80	(2)	4.55	(19)	12.20	(19)
120	12	10	19	55.21	(0)	65.32	(6)	1.84	(19)	5.84	(19)
120	15	8	13	27.46	(2)	31.46	(11)	0.62	(13)	1.62	(13)
120	20	6	17	4.71	(17)	5.24	(17)	0.35	(17)	0.41	(17)
120	24	5	20	1.95	(20)	2.00	(20)	0.10	(20)	0.15	(20)
120	30	4	20	0.80	(20)	0.85	(20)	0	(20)	0	(20)
120	40	3	20	0.10	(20)	0.10	(20)	0.05	(20)	0.05	(20)
			429		(307)		(352)		(396)		(402)

differences between the lower bounds and the actual numbers of reshuffles are much higher: the average values for $\Delta RS(BI^*)$ are between 4.55 and 66.20 (maximum 94) in the unrestricted, and between 12.20 and 94.15 (maximum 145) in the restricted case. Furthermore, the average values of $\Delta RS(US^*_{adj})$ range from 94.65 to 263.75 (maximum 305) in the unrestricted and even from 111.8 to 286.15 (maximum 350) in the restricted case. Thus, for such instances the lower bounds US^*_{adj} and BI^* are not suitable to estimate the actual number of reshuffles.

In Table 2.3, results for instances with duplicate priorities are shown. Here, we report only results for the restricted BRP since the IDA-algorithm of Tanaka and Mizuno [96] can only solve this version of the BRP. All instances could be solved to optimality by every algorithm. We see that a larger value of d (denoting the number of items with the same priority value) leads to easier instances. On the one hand, the values of US^*_{adj} and BI^* decrease with larger d; on the other hand, also the Δ-values decrease since items with the same priority value can be stacked without conflicts more easily.

In a third experiment, we wanted to study the influence of the optimal configuration obtained by CPLEX using it as initial configuration for the BRP. Usually, IP_{BI} has many optimal solutions and the question is whether we have always chosen a "good" or "bad" one (in terms of the total number of reshuffles) and how large the deviations may be. For this purpose, we tried to generate all optimal solutions w.r.t. BI and calculated the optimal values of RS_r and RS_u for each of them.

Basically, it does not matter which item is assigned to which stack because all stacks are interchangeable. It is only important which items are grouped together and we do not have to know the indices q of the assigned stacks. Hence, we may reduce the number of solutions

Table 2.3: Instances with duplicate priorities: number of reshuffles for solutions optimized w.r.t. US_{adj} and BI.

n	m	b	d	US^*_{adj}	BI^*	$\Delta RS_r(US^*_{adj})$	$\Delta RS_r(BI^*)$
40	4	10	1	6.35	13.95	36.65	7.95
40	5	8	1	4.90	9.50	28.40	3.35
*40	7	6	1	2.70	3.85	8.90	0.20
40	8	5	1	1.80	2.40	4.90	0.05
40	10	4	1	0.75	0.90	0.95	0
40	4	10	2	5.80	12.35	31.00	6.40
40	5	8	2	4.30	7.90	20.60	2.40
*40	7	6	2	2.25	2.80	6.80	0.15
40	8	5	2	1.40	1.70	2.95	0.20
40	10	4	2	0.50	0.60	0.60	0.05
40	4	10	4	4.30	8.60	20.70	2.70
40	5	8	4	2.65	4.55	10.40	0.40
*40	7	6	4	0.65	0.70	1.30	0
40	8	5	4	0.35	0.35	0.20	0
40	10	4	4	0	0	0	0

by a factor of $m!$, not considering the order of the stacks. This can be done by adding the constraints

$$\sum_{i \in I} i \cdot (\beta_{i,q-1} + \alpha_{i,q-1}) - \sum_{i \in I} i \cdot (\beta_{i,q} + \alpha_{i,q}) \leqslant 0 \qquad \forall q \in \mathcal{Q} \setminus \{1\} \tag{2.35}$$

to the formulation IP_{BI}, ensuring that the bottom elements in the stacks are ordered according to increasing indices.

With this extended formulation, we generated all optimal solutions for each instance from the instance group with $n = 30, m = 5, b = 6$. In Table 2.4 the results are shown. The column “No.” contains the instance number, “# solutions” is the number of optimal solutions, and BI^* the optimal objective value. It can be seen that these instances are quite heterogeneous in terms of the number of optimal solutions, ranging from 4 to nearly 6 million different optimal solutions. For the evaluation of each solution w.r.t. the actual number of reshuffles, we limited our considerations to the first 200,000 generated solutions for each instance. Each solution was used as initial configuration of the BRP and solved with the IDA-algorithm for the restricted and the unrestricted version. The three columns “min”, “avg”, “max” show the minimum, the average, and the maximum of the values $\Delta RS_r(BI^*)$ and $\Delta RS_u(BI^*)$. We call all reshuffles that must be performed beyond the lower bound value BI, “additional reshuffles”.

It can be seen that in the unrestricted version for every instance a solution exists which does not need any additional reshuffle. For the restricted version, two instances need one additional reshuffle in the best case. On the other hand, for 16 of the 20 instances there exist storage configurations that result in much worse solutions w.r.t. RS_r and RS_u. On average, 0.6 unrestricted and 1.9 restricted additional reshuffles must be performed if any optimal solution of the formulation IP_{BI} is used. In the worst case, a solution with an optimal number of badly placed items needs 13 restricted and 7 unrestricted additional reshuffles. If we take into account that the investigated instances are small and even the average values show that there exist a lot of solutions with additional (unnecessary) reshuffles, we may conclude that only solving the IP and taking an optimal solution for BI, is perhaps not the best approach.

Table 2.4: All optimal solutions generated with CPLEX for small instances with $n = 30, m = 5, b = 6$.

No.	# solutions	BI^*	$\Delta RS_r(BI^*)$			$\Delta RS_u(BI^*)$		
			min	avg	max	min	avg	max
1	48	4	0	0	0	0	0	0
2	6669	4	0	0.3	2	0	0.3	2
3	186020	3	0	0.2	2	0	0.1	2
4	34068	8	0	2.6	6	0	0.9	3
5	4	4	0	0	0	0	0	0
6	66282	6	1	3.6	6	0	1	3
7	852	7	0	3.1	7	0	0.6	2
8	68913	8	0	2	6	0	0.7	3
9	1342511	4	0	1.9	6	0	0.7	3
10	953	5	0	1.9	4	0	0.5	1
11	1467541	8	0	3.7	9	0	2	7
12	18	2	0	0	0	0	0	0
13	6701	6	0	1.6	6	0	0.2	2
14	9598	6	0	2.7	7	0	0.5	2
15	129172	6	0	1.3	4	0	0.3	1
16	1666	3	0	0	0	0	0	0
17	4984921	7	0	2.6	11	0	0.9	5
18	5864179	9	0	4.8	13	0	1.1	6
19	39452	7	1	3.6	8	0	1.8	4
20	944	6	0	2.3	6	0	1	3

2.6.2 Lower bounds

In the following, we compare the different lower bounds introduced in Section 2.4 as well as the optimal solutions of IP_{BI}.

In Table 2.5 we report average values for the new longest increasing subsequence lower bound LB_{LIS} on the number of unordered stackings and the iterative longest increasing subsequence lower bound LB_{ILIS} on the number of badly placed items. We also compare them with the lower bound $LB_{P\text{-}\infty}$ of Boysen and Emde [15] and an optimal solution of IP_{BI} (calculated with a larger time limit of two days), respectively. Within this time limit, only two among the 429 instances with $n \leqslant 120$ could not be verified by IP_{BI}, for these instances we use the best lower bound calculated by CPLEX. For $n = 500$, the LP relaxation does not provide any lower bound greater than zero for the instances with $n = 500$. Note that our values of $LB_{P\text{-}\infty}$ differ slightly from the values reported in Table 2 in Boysen and Emde [15] since there some other (but very similar) instances were used, which according to the authors no longer exist.

Computing the $\mathcal{O}(n \log n)$-bound LB_{LIS} took no longer than 1 millisecond, while calculating the $\mathcal{O}(n^3)$-bound $LB_{P\text{-}\infty}$ needed up to 761 milliseconds for the instances with $n = 500$. As shown in Section 2.4, we always have $LB_{P\text{-}\infty} \geqslant LB_{LIS}$. However, for the 529 tested instances, LB_{LIS} is equal to $LB_{P\text{-}\infty}$ for 483 ones. For the remaining 46 instances, the difference is between 1 and 8.

In contrast to the second instance in Example 2.3, for all our test instances the lower bound $LB_{P\text{-}\infty}$ is dominated by LB_{ILIS}. For the 260 instances with $n \leqslant 60$, the lower bound LB_{ILIS} gives a better bound than $LB_{P\text{-}\infty}$ for 79 instances, with differences between 0 and 9. For the 169 instances with $n = 120$, the new lower bound performs 84 times better, with differences up to 39. Finally, for the 100 instances with $n = 500$, the lower bound LB_{ILIS} is better for 40 instances, with differences up to 65.

Table 2.5: Comparison of lower bounds.

n	m	b	inst	LB_{LIS}	$\text{LB}_{\text{P-}\infty}$	LB_{ILIS}	IP_{BI}
30	5	6	20	3.90	3.95	5.05	5.65
30	6	5	20	2.90	2.90	3.25	3.60
30	8	4	20	1.00	1.00	1.00	1.05
30	10	3	20	0.15	0.15	0.15	0.40
40	5	8	20	4.70	4.85	8.05	9.50
40	7	6	20	2.70	2.70	3.50	3.85
40	8	5	20	1.75	1.75	2.05	2.40
40	10	4	20	0.20	0.20	0.20	0.90
60	6	10	20	6.25	6.80	11.80	14.15
60	10	6	20	2.25	2.25	2.60	3.35
60	12	5	20	0.60	0.60	0.60	1.40
60	15	4	20	0	0	0	0.60
60	20	3	20	0	0	0	0.20
120	5	24	20	13.95	18.40	50.20	57.90
120	8	15	20	10.95	11.70	28.20	33.00
120	10	12	20	8.95	9.05	18.35	21.80
120	12	10	19	7.05	7.05	11.47	13.68
120	15	8	13	4.15	4.15	4.92	6.08
120	20	6	17	0.24	0.24	0.24	1.65
120	24	5	20	0	0	0	0.80
120	30	4	20	0	0	0	0.40
120	40	3	20	0	0	0	0.10
500	20	25	20	19.55	19.80	72.15	-
500	25	20	20	14.55	14.55	37.45	-
500	50	10	20	0	0	0	-
500	100	5	20	0	0	0	-
500	125	4	20	0	0	0	-

On the other hand, the bound LB_{ILIS} is dominated by the bound IP_{BI} for all instances with $n \leqslant 120$. Among these 429 instances, IP_{BI} is better for 231 instances, with differences up to 10. Also for $n = 500$, LB_{ILIS} can still be calculated in at most 1 millisecond for every instance. This shows that the new lower bound LB_{ILIS} yields very good results taking into account the fast runtime compared with IP_{BI}. Especially, for instances with many items the new lower bound is very useful.

2.6.3 Simulated annealing

We applied our SA algorithm to the 429 instances from Table 2.1 and 100 instances with $n = 500$. We imposed the same time limit of 30 minutes as for the MIP formulations. To generate a good starting solution, each instance was first solved with the formulation IP_{BI} setting a time limit of 10 minutes, then the SA algorithm got a time limit of 20 minutes to optimize this solution w.r.t. RS_r or RS_u. Since the instances with $n = 500$ are too large to tackle them with the IP formulation, for these instances random starting solutions were generated.

In Table 2.6 average values for all instance groups are shown. The column LB contains the best lower bound on the number of badly placed items. For $n \leqslant 120$, LB is determined by IP_{BI}, for the instances with $n = 500$ we used LB_{ILIS}. Columns $\Delta\text{RS}_u^{SA}, \Delta\text{RS}_r^{SA}$ show the average

Table 2.6: Simulated annealing results.

n	m	b	inst	LB	ΔRS_u^{SA}	ΔRS_u^{IP}	ΔRS_r^{SA}	ΔRS_r^{IP}
30	5	6	20	5.65	0	0.65	0.05	1.65
30	6	5	20	3.60	0	0.50	0	1.10
30	8	4	20	1.05	0	0	0	0.05
30	10	3	20	0.40	0	0	0	0
40	5	8	20	9.50	0	1.50	0.10	3.35
40	7	6	20	3.85	0	0	0	0.20
40	8	5	20	2.40	0	0.05	0	0.05
40	10	4	20	0.90	0	0	0	0
60	6	10	20	14.15	0.15	3.65	0.50	8.30
60	10	6	20	3.35	0.10	0.25	0.10	0.45
60	12	5	20	1.40	0	0.15	0	0.20
60	15	4	20	0.60	0	0.05	0	0.05
60	20	3	20	0.20	0	0	0	0
120	5	24	20	57.90	6.35	66.20	17.15	94.15
120	8	15	20	33.00	0.75	10.50	1.35	24.50
120	10	12	20	21.80	0.50	4.55	0.30	12.20
120	12	10	19	13.68	0.63	2.58	0.53	6.58
120	15	8	13	6.08	0.77	1.23	0.54	2.23
120	20	6	17	1.65	0.12	0.47	0.12	0.53
120	24	5	20	0.80	0	0.10	0	0.15
120	30	4	20	0.40	0	0	0	0
120	40	3	20	0.10	0	0.05	0	0.05
500	20	25	20	72.15	247.90	-	298.15	-
500	25	20	20	37.45	120.45	-	115.60	-
500	50	10	20	0	15.55	-	6.35	-
500	100	5	20	0	0.70	-	0.20	-
500	125	4	20	0	0.20	-	0.05	-

differences $RS_{u/r} - LB$ for the SA solutions w.r.t. the numbers of unrestricted/restricted reshuffles, respectively. Similar to Boysen and Emde [15], we report absolute differences instead of relative ones since often the lower bounds are small (or even equal to zero). Furthermore, the columns $\Delta RS_u^{IP}, \Delta RS_r^{IP}$ contain the average differences $RS_{u/r} - LB$ for the solutions obtained by the formulation IP_{BI} (also used as a heuristic to minimize RS) in terms of the actual number of reshuffles (cf. Table 2.2).

It turns out that the SA solutions are quite good; for the small instances with $n \leqslant 60$ the average value of ΔRS_u^{SA} is 0.02 (the maximum over all instances is 1), ΔRS_r^{SA} is on average 0.06 (maximum 2). For the instances with $n = 120$ and $b \leqslant 15$ the average value of ΔRS_u^{SA} is 0.35 (maximum 2) ΔRS_r^{SA} is on average 0.46 (maximum 6). The average deviations for the instances with $n = 120$ and $b = 24$ are slightly worse (6.35/17.15), for the instances with $n = 500$ and $b \geqslant 20$ the deviations are much larger.

Finally, we compare the SA solutions with the solutions obtained by the formulation IP_{BI}. For the SA solutions, the average difference ΔRS_u^{SA} between the number of unrestricted reshuffles and the best lower bound is at most 0.15 for the smaller instances with $n \in \{30, 40, 60\}$, while IP_{BI} seen as a heuristic needed up to 3.65 reshuffles in addition to the lower bound. For the restricted BRP, the average values of ΔRS_r^{SA} are at most 0.5, while ΔRS_r^{IP} is at most 8.3. The results for the instances with $n = 120$ items are even more advantageous since the average differences for the instances with $b \leqslant 15$ are at most 1.69

for the restricted and unrestricted BRP, while the differences are at most 10.5 unrestricted respectively 24.5 restricted reshuffles for the solutions of $\mathrm{IP}_{\mathrm{BI}}$. Recall that for the instances with $n = 500$ the IP could not find any feasible solution.

Thus, we may conclude that solving the PSLP with the actual objective functions RS_r or RS_u heuristically gives much better results than optimizing w.r.t. the surrogate objective functions US_{adj} or BI.

2.7 Conclusions

In this chapter, we considered the parallel stack loading problem for the surrogate objectives counting “unordered stackings” and “badly placed items” as well as for the important and more realistic objective “total number of reshuffles”. We stated a new and strengthened $\mathcal{NP}$-completeness proof for all three objectives, analyzed the problem theoretically regarding quantitative relations between the different objectives, and developed new lower bounds for the surrogate objective “total number of badly placed items”. Furthermore, we dealt with the research question of Boysen and Emde [15] “which surrogate objective shows the greatest accuracy in approximating the true retrieval effort”. For this purpose, we compared the results with the actual numbers of (restricted and unrestricted) reshuffles. Our experiments showed a great benefit of the more accurate surrogate objective function BI counting badly placed items instead of only counting unordered stackings US_{adj} (especially, for instances with larger stack heights). Furthermore, for $b \leqslant 6$ (which is the common situation in real-world container terminals), the number of badly placed items BI is a good approximation of the actual number of reshuffles.

Additionally, we proposed a SA heuristic for the combined loading/unloading problem that tries to find a storage configuration for which the total number of reshuffles in the unloading stage is small. We obtained quite good solutions if we proceed in two stages and evaluate configurations with a fast BRP solver. It turned out that it is much better to solve the problem with the (more complex) reshuffle objective heuristically instead of calculating optimal solutions for the surrogate objective BI with an exact algorithm. On the other hand, for larger instances with $n = 500$, the deviations between our heuristic solutions and the lower bounds become quite large. Hence, for further research it would be interesting to decrease these deviations by trying to calculate more accurate bounds and perhaps also improve the behavior of the SA algorithm.

Chapter 3

Robust optimization for premarshalling with uncertain priority classes

In this chapter, we consider a robust optimization approach for premarshalling with uncertain priority classes. The research contained in this chapter has already been published in Boge et al. [9] and is organized as follows. First, we give an introduction in Section 3.1. In Section 3.2 we describe the problem setting more precisely. Theoretical results on the existence of robust solutions can be found in Section 3.3. In Section 3.4 we present *mixed-integer programming* (MIP) models for the robust problem under consideration, and in Section 3.5 we report results of a computational study. Finally, some conclusions can be found in Section 3.6.

3.1 Introduction

In *premarshalling problems*, items are sorted inside the storage area so that all items can be retrieved without any further relocations afterwards. Recall that just as in the *blocks relocation problem* (BRP) in the *premarshalling problem* (PMP), the items have priority values, but do not leave the storage area immediately. The items are moved to positions so that they can be retrieved without relocations later in the unloading stage. Often, such a sorting process is done during the night, before the unloading process is started on the next day.

In this chapter, we consider the premarshalling problem where the priority values of the items are not exactly known when the sorting is done. Our setting is motivated by the real-world situation in container terminals where containers are stored and later retrieved by ships or trucks. For each item its estimated retrieval time is derived from the expected arrival time of the corresponding ship or truck. Since in practice the arrival times of ships or trucks are affected by different uncertainties, they are usually not deterministically known in advance. In this situation, a computed solution of the deterministic PMP may cause additional relocations since items with earlier departure times may be stored below items which are now retrieved later due to delays. Thus, a terminal could benefit from solutions which are more robust with respect to changes in the retrieval sequence.

Different approaches to handle uncertain data in optimization problems have been proposed. Most prominently, these include stochastic and robust optimization. While the stochastic approach usually requires deeper problem knowledge in form of a probability distribution on the uncertain data, we only require a description of all relevant scenarios for the robust approach that we follow in this chapter. For general surveys on robust optimization, we refer to Gabrel et al. [41] as well as Goerigk and Schöbel [48].

Robust premarshalling problems have already been studied by Rendl and Prandtstetter [89] as well as Tierney and Voß [105]. In the robust PMP introduced in Rendl and Prandtstetter [89], it is assumed that each item has an associated interval of possible priority values modeling uncertainties in the retrieval times. Two items in the same stack are "conflicting" if their associated intervals overlap and a configuration is called robust if all items can be stacked without any conflicts. A *constraint programming* (CP) formulation is proposed which is able to solve only small instances in the case that a robust configuration exists. In Tierney and Voß [105] a more general concept of uncertainties is considered. There, a "blocking matrix" is introduced which indicates for each pair of items which items may be stacked on each other and which items are in conflict. Then, an adapted *iterative deepening A** (IDA) algorithm from Tierney et al. [104] is applied to instances with interval uncertainties to find strictly robust solutions. If a strictly robust solution exists, the algorithm in principle finds one (if the time limit is large enough). On the other hand, if no strictly robust solution exists, the algorithm does not terminate. Hence, if the algorithm reaches the time limit, it is not clear whether no robust solution exists or the computing time was too short. In the experiments by Tierney and Voß [105] it is shown that this approach clearly dominates the CP formulation of Rendl and Prandtstetter [89].

In this chapter, we introduce a new model for uncertainty, motivated by the occurrence of delays. We assume that a limited number of elements in the retrieval sequence may be swapped and study the impact of this number. In contrast to previous approaches, we do not only focus on strictly robust solutions, but also compute robust solutions if no strictly robust solution exists. We analyze the complexity of the resulting robust problem, and give analytical criteria when a stacking configuration exists that remains feasible in every considered scenario. We further present MIP and *integer program* (IP) formulations for the robust problem. In a computational study, we consider the computational effort to calculate robust solutions, evaluate the strength of the analytical existence criteria, and highlight the advantages (in terms of additional robustness) and disadvantages (in terms of additional relocations) of using our robust solutions.

3.2 Problem formulation

In this section, we introduce our notation and describe the problem under consideration more formally. We are given a storage area which consists of m stacks $\mathcal{Q} = \{1, \dots, m\}$, each stack contains b levels $\mathcal{L} = \{1, \dots, b\}$, i.e., at most b items can be stored in each stack. There are n items from a set $\mathcal{I} = \{1, \dots, n\}$ stored in the area, each item belonging to exactly one priority class $\gamma(i) \in \mathcal{P} = \{1, \dots, N\}$. If $\gamma(i) < \gamma(j)$, item i has to be retrieved earlier than item j. This means that in the retrieval process, all items of priority class 1 have to be retrieved first (in an arbitrary order), then all items of priority class 2, etc. We denote by $C_k \subseteq \mathcal{I}$ the set of all items belonging to priority class $k \in \mathcal{P}$. In a more special situation, we have $N = n$ and all items belong to different priority classes, i.e., $\gamma(i) \neq \gamma(j)$ for all $i \neq j$. This means that there is a unique sequence for the items in which they have to be retrieved; for example, such a situation occurs in practice if each item is retrieved by a single truck and the trucks arrive one after the other.

Our setting also includes the case that some priority classes are empty, i.e., no item is scheduled to be retrieved for such time periods. Using empty priority classes, it is possible to model that some item retrievals have more time between them and are thus less likely to exchange their positions.

Recall that for a given storage configuration, an item is called "blocked" if one or more items with later retrieval time (so-called "blocking" items) are stacked above it in the same stack. Such a situation is sometimes also called a "mis-overlay" or an "overstow". Before retrieving a blocked item, each blocking item has to be removed from this stack and relocated to another stack. Such an unproductive move is also called a "reshuffle".

The goal of the premarshalling problem is to transform a given storage configuration into a feasible configuration without any blockings. This transformation can be described by a sequence of moves (q, q') with the meaning that the topmost item in stack $q \in \mathcal{Q}$ is moved to the top of stack $q' \in \mathcal{Q}$. Such a move is feasible if the destination stack q' is not full. As objective function, usually, the total number of reshuffles during the premarshalling process is minimized. We denote this objective function by $\mathrm{RS}^{\mathrm{PM}}$.

Any configuration may be described by specifying for each item $i \in \mathcal{I}$ its location $(q, l) \in \mathcal{Q} \times \mathcal{L}$ as a pair of stack and level. In our setting, we often only need configurations of the priority classes, i.e., which priority class is stored at location (q, l).

In the literature, two different versions of storage areas are considered. While it is often assumed that all moves have to be performed inside the storage area, sometimes also an additional temporary storage area exists, cf. Wang [113], Dayama et al. [25], Wang et al. [114], and de Melo da Silva et al. [26]. In the first case, it is more difficult to find feasible movements and items may have to be reshuffled several times. However, it is mostly assumed that sufficient space exists to transform any configuration into a feasible configuration without blockings. In the second situation, in principle, all items can be removed from the area and reinserted in an arbitrary order. However, since such movements are time-consuming, they should be avoided as much as possible. In this chapter we study both versions, depending on the context.

If we assume that each priority class belongs to one specific vehicle or ship, in case of delays, the positions of the items in the retrieval sequence do not change independently, but all items of the same priority class change their priority value simultaneously. To model such a situation, we assume that we are given an initial nominal sequence $\hat{\pi} = (1, \dots, N)$ of the priority classes and the actual sequence π is uncertain. In principle, π can be any other sequence, but in practice it is more likely that π is somehow similar to $\hat{\pi}$ (i.e., not all priority values are changed simultaneously). To model this, we construct a so-called uncertainty set $\mathcal{U}^\Gamma$ containing all scenarios that are assumed to be possible. The set $\mathcal{U}^\Gamma$ contains all sequences π that can be obtained from $\hat{\pi}$ by swapping at most $\Gamma \in \{0, 1, \dots, \frac{N(N-1)}{2}\}$ adjacent elements. The "swap distance" Γ between permutations π and $\hat{\pi}$ is also known as the Kendall-Tau distance (cf. Kendall [63]) and can be calculated by counting all pairs $i < j$ with $\pi_i > \pi_j$. Furthermore, $\Gamma = 0$ corresponds to the nominal scenario and for $\Gamma = \frac{N(N-1)}{2}$ the set $\mathcal{U}^\Gamma$ contains all possible permutations.

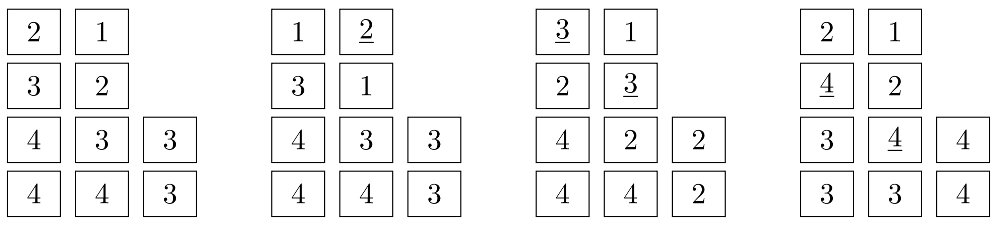
For example, let us consider the case $N = 4$ and $\hat{\pi} = (1, 2, 3, 4)$. Then, we have

$$\begin{aligned}
\mathcal{U}^0 &= \{(1,2,3,4)\}, \\
\mathcal{U}^1 &= \mathcal{U}^0 \cup \{(2,1,3,4), (1,3,2,4), (1,2,4,3)\}, \\
\mathcal{U}^2 &= \mathcal{U}^1 \cup \{(1,3,4,2), (1,4,2,3), (2,1,4,3), (2,3,1,4), (3,1,2,4)\}, \\
& \dots \\
\mathcal{U}^6 &= \mathcal{U}^5 \cup \{(4,3,2,1)\}.
\end{aligned}$$

In Knuth [67], an exact formula is presented for the number of permutations with length N having swap distance exactly $\Gamma \leqslant N$. Using this result, we see that the number of elements in $\mathcal{U}^\Gamma$ is at least in the order of magnitude of N^Γ.

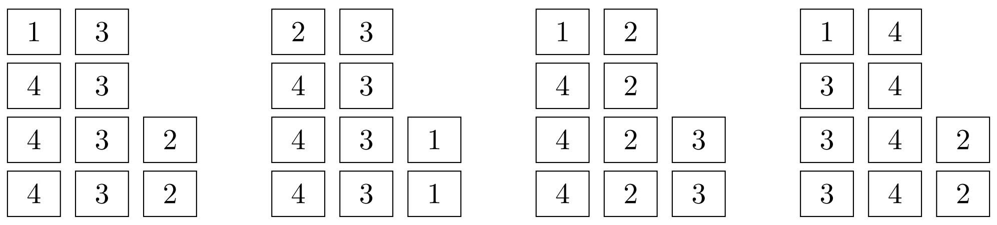
In case of uncertainties, final configurations computed for the premarshalling problem may contain blockings if the order of the priority classes is changed, i.e., they are no longer feasible solutions for the PMP. This means that later in the unloading stage additional reshuffles are necessary to retrieve all items in a correct order.

Example 3.1. *Consider an example with $N = 4$ priority classes and $n = 10$ items stored in $m = 3$ stacks of height $b = 4$. In Figure 3.1 and 3.2, two different configurations c and c' are shown (for each item i its priority class $\gamma(i)$ is depicted). Both are feasible (i.e., have no blocking items) for the nominal scenario $\hat{\pi} = (1, 2, 3, 4)$ (cf. Figure 3.1a and 3.2a). In Figure 3.1b to 3.2d the settings for the actual priority classes according to the permutations $\pi^1, \pi^2, \pi^3 \in \mathcal{U}^1$ are shown. While the configuration c' also has no blockings for all permutations π^1, π^2, π^3 (cf.*



(a) $\hat{\pi} = (1, 2, 3, 4)$. (b) $\pi^1 = (2, 1, 3, 4)$. (c) $\pi^2 = (1, 3, 2, 4)$. (d) $\pi^3 = (1, 2, 4, 3)$.

Figure 3.1: Configuration c.



(a) $\hat{\pi} = (1, 2, 3, 4)$. (b) $\pi^1 = (2, 1, 3, 4)$. (c) $\pi^2 = (1, 3, 2, 4)$. (d) $\pi^3 = (1, 2, 4, 3)$.

Figure 3.2: Configuration c'.

Figure 3.2b, 3.2c, 3.2d), the configuration c has one (cf. Figure 3.1b), two (cf. Figure 3.1c) and two (cf. Figure 3.1d) blocking items (underlined), i.e., it is not a feasible final premarshalling configuration for these scenarios. Hence, the configuration c' is more robust against uncertainties. □

We call a configuration Γ-robust if in every scenario from $\mathcal{U}^\Gamma$, there are no blocking items. A configuration is Γ-robust if and only if for all items i, j where i is stacked on top of j the condition

$$\gamma(i) = \gamma(j) \text{ or } \gamma(j) - \gamma(i) > \Gamma \tag{3.1}$$

holds. For example, configuration c from Figure 3.1 is not 1-robust since there are items i, j in the same stack with $\gamma(j) - \gamma(i) = 1$. On the other hand, configuration c' from Figure 3.2 is 2-robust, but not 3-robust since for $\pi = (4, 3, 2, 1) \in \mathcal{U}^3$ a blocking in the first stack occurs (there are items i, j with $\gamma(j) - \gamma(i) = 4 - 1 = 3$).

Obviously, not every instance admits a Γ-robust configuration. However, in this situation, we may want to find a configuration for which as few reshuffles as possible are needed in the unloading stage. Minimizing the total number of reshuffles for a fixed storage configuration corresponds to the BRP, which is known to be strongly $\mathcal{NP}$-hard and difficult to solve even for small instances (cf. Caserta et al. [19]). Thus, finding a configuration which needs a minimum number of reshuffles may be even more complex.

To evaluate a storage configuration with respect to blockings and to estimate the number of reshuffles in the unloading stage, the objective function BI (total number of badly placed items, also called "number of confirmed relocations" in Kim and Hong [64]) has been introduced. An item is "badly placed" (cf. Forster and Bortfeldt [39]) if it is blocking an item placed somewhere below in the same stack that has to be retrieved earlier. On the other hand, we denote by RS^U the actual number of reshuffles needed to unload all items from the stacks according to their priorities. We do not count the retrieval operations here (since all n items have to be retrieved in the unloading process, we always have a constant number of n retrievals). Obviously, for each configuration c the value $BI(c)$ defines a lower bound on the total number of reshuffles $RS^U(c)$ in the unloading stage since each blocking item has to be relocated at least once to retrieve all items.

In the following, we adapt the objective function BI to the robust setting. For any configuration c and a permutation $\pi \in \mathcal{U}^\Gamma$ we denote by c^π the realized configuration of priority classes for scenario π. Then let

$$\mathrm{BI}^\Gamma_{\mathrm{rob}}(c) := \max_{\pi \in \mathcal{U}^\Gamma} \{\mathrm{BI}(c^\pi)\} \tag{3.2}$$

be the maximum number of badly placed items occurring in any scenario from the uncertainty set $\mathcal{U}^\Gamma$. In a corresponding optimization problem we look for a configuration c such that $\mathrm{BI}^\Gamma_{\mathrm{rob}}(c)$ is as small as possible. Later on, we also consider the so-called “adversary problem”: For a given configuration c we want to find a worst-case scenario, i.e., a permutation $\pi \in \mathcal{U}^\Gamma$ that maximizes $\mathrm{BI}(c^\pi)$.

For any configuration c and each item $i \in \mathcal{I}$ let $B_i(c)$ be the set of all items $j \in \mathcal{I}$ where j is stacked somewhere below i in the same stack and $\gamma(i) \neq \gamma(j)$. Instead of calculating $\mathrm{BI}^\Gamma_{\mathrm{rob}}(c)$ exactly, we also use the upper bound

$$\overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c) = |\{i \in \mathcal{I} : \exists j \in B_i(c) \text{ with } |\gamma(j) - \gamma(i)| \leqslant \Gamma\}| \tag{3.3}$$

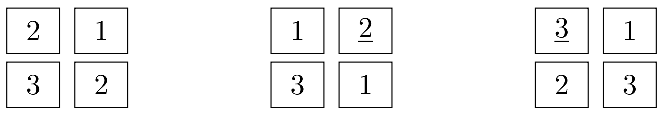
by counting all items i which may become badly placed due to an item j below that violates (3.1). Since a configuration c is Γ-robust if and only if (3.1) for all $i \in \mathcal{I}$ and all $j \in B_i(c)$ holds, we have

$$\mathrm{BI}^\Gamma_{\mathrm{rob}}(c) = 0 \text{ if and only if } \overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c) = 0. \tag{3.4}$$

Furthermore,

$$\overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c) = 1 \text{ implies } \mathrm{BI}^\Gamma_{\mathrm{rob}}(c) = 1, \tag{3.5}$$

since if only a single pair (i, j) with $j \in B_i(c)$ violates (3.1), for each scenario $\pi \in \mathcal{U}^\Gamma$ at most one badly placed item exists (and there must also be at least one scenario with one badly placed item).



(a) $\hat{\pi} = (1, 2, 3)$. (b) $\pi^1 = (2, 1, 3)$. (c) $\pi^2 = (1, 3, 2)$.

Figure 3.3: Configuration c with $\mathrm{BI}^\Gamma_{\mathrm{rob}}(c) = 1$, but $\overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c) = 2$.

However, $\mathrm{BI}^\Gamma_{\mathrm{rob}}(c) = 1$ does not imply $\overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c) = 1$ as the example in Figure 3.3 shows. There we have a configuration c with $N = 3$ priority classes, $n = 4$ items stored in $m = 2$ stacks, and $\Gamma = 1$. For the nominal scenario $\hat{\pi} = (1, 2, 3)$ we have $\mathrm{BI}(c^{\hat{\pi}}) = 0$, and for the two permutations $\pi^1 = (2, 1, 3), \pi^2 = (1, 3, 2) \in \mathcal{U}^1$ we have $\mathrm{BI}(c^{\pi^1}) = \mathrm{BI}(c^{\pi^2}) = 1$, i.e., $\mathrm{BI}^\Gamma_{\mathrm{rob}}(c) = \max\{0, 1, 1\} = 1$. On the other hand, $\overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c) = 2$ since for $\hat{\pi}$ in both stacks (3.1) is violated. However, not both swaps $2 \leftrightarrow 3$ and $1 \leftrightarrow 2$ can occur simultaneously in the case $\Gamma = 1$, i.e., $\overline{\mathrm{BI}}^\Gamma_{\mathrm{rob}}(c)$ overestimates the actual value.

Recall that a feasible final configuration for the classical deterministic premarshalling problem ($\Gamma = 0$) does not contain any blockings (i.e., $\mathrm{BI}^0_{\mathrm{rob}} = \mathrm{BI} = 0$) and that such a configuration can be reached in most realistic situations (cf. Section 3.3). Thus, most of the algorithms known for the deterministic setting concentrate on minimizing the total number of reshuffles, while a feasible final configuration without any blockings is mandatory. Since in many situations, a configuration without any blockings is not achievable for $\Gamma > 0$, we relax this traditional condition and also allow final configurations with blockings. We consider robustness as primary and the number of reshuffles as secondary goal. Our objective is to find a storage configuration minimizing lexicographically $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$ and the total number of reshuffles $\mathrm{RS}^{\mathrm{PM}}$ to achieve a corresponding solution.

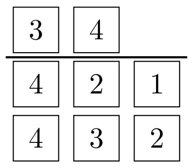
3.3 Theoretical analysis

In this section, we deal with some theoretical issues in connection with the robust premarshalling problem. In Section 3.3.1, we study whether in the deterministic setting a given configuration can be transformed into a feasible configuration without blockings. In Section 3.3.2, we consider the question whether a Γ-robust solution exists in the robust setting. Finally, in Section 3.3.3, we show that already the adversary problem is $\mathcal{NP}$-hard.

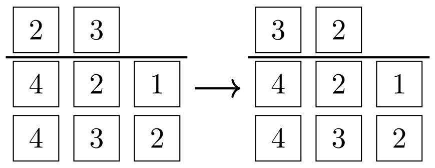
3.3.1 Feasibility

First we deal with the feasibility problem in the deterministic setting, i.e., the question whether a given configuration can be transformed into a configuration without blockings by only using feasible moves in the storage area of limited size. This has recently been stated as an open problem in Parreño-Torres et al. [84]: "On the other hand, there is the unsolved question of deciding whether a premarshalling instance has a feasible solution or not. It is a difficult question, because it depends not only on the dimensions of the bay and the number of containers, but also on the positions of the containers." Independently to our work, the feasibility problem was already studied in the dissertation of Wang [113]. Although in this thesis an involved proof is presented, some details are only sketched. In the following, we present a simpler and more thorough proof, based on another idea which may also be useful in other settings.

The case with $m = 2$ stacks only allows to move all blocking items from one to the other stack (and vice versa) so that feasibility can easily be checked. In the following, we assume that we have at least $m \geqslant 3$ stacks and denote by $f = mb - n$ the number of free slots in the storage. Obviously, in the case $f = 0$ an instance is only feasible if the storage is completely sorted, i.e., there are no blocking items. In the case $f = 1$, only the items in the topmost level can be changed.



(a) Infeasible.



(b) Feasible.

Figure 3.4: PMP instances with $f = 1$.

Example 3.2. *Consider the configuration in Figure 3.4a where again the γ-values are depicted. This instance is infeasible, since there is only a single feasible location (in the first stack) for the two items with priorities 3 and 4 in the topmost level. On the other hand, the instance in Figure 3.4b is feasible since it can be transformed into the feasible configuration shown in the right part by using the moves* $(1,3), (2,1), (3,2)$. □

For the general situation $f \geqslant 1$, we denote by $\mathcal{I}^{\mathrm{top}(f)}$ the set of all items in the top f levels. It is easy to see that only these items can change their positions and that the items in the levels $1, \dots, b - f$ are fixed. In the following we show that there is no restriction on the order of the items $\mathcal{I}^{top(f)}$ and that they can be ordered in an arbitrary way. Additionally, we show that each of these items can be moved to any other position without changing the positions of all other items.

Lemma 3.1. *In an instance of the PMP with $m \geqslant 3$ stacks and $f \geqslant 1$ free slots, all items $\mathcal{I}^{top(f)}$ stored in the top f levels can be transformed into an arbitrary configuration.*

Proof: We prove the statement in a constructive way in three steps.

(i) Each item $i \in \mathcal{I}^{top(f)}$ can be moved to the topmost level in its stack without changing the order of the other items in this stack and without changing the positions of all items in the remaining stacks.

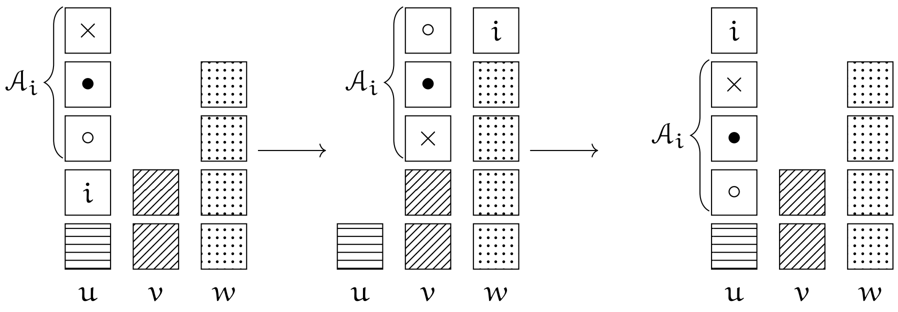


Figure 3.5: Transformation (i).

For each item i let $\mathcal{A}_i$ be the set of items above item i and $a_i := |\mathcal{A}_i|$. For each stack q let f_q be the number of free slots in q. Consider an item $i \in \mathcal{I}^{top(f)}$ and let u be its stack. Since i belongs to the top f levels, we have $a_i + f_u \leqslant f - 1$. Hence, the number of free slots not contained in stack u is at least $f - f_u \geqslant a_i + 1$. Let $v, w \neq u$ be stacks with $f_v \geqslant a_i$ and $f_w \geqslant 1$ (if such stacks do not exist, they can easily be obtained by some additional moves).

In order to move i to the topmost level in u, we proceed as follows (cf. Figure 3.5): Move all items $\mathcal{A}_i$ onto stack v and item i to stack w. Then move all items $\mathcal{A}_i$ from stack v back to u and put item i on top of them. If additional moves were necessary to get stacks v, w, reverse them also.

(ii) The topmost items $i, j \in \mathcal{I}^{top(f)}$ in two different stacks can be swapped without changing the positions of all other items.

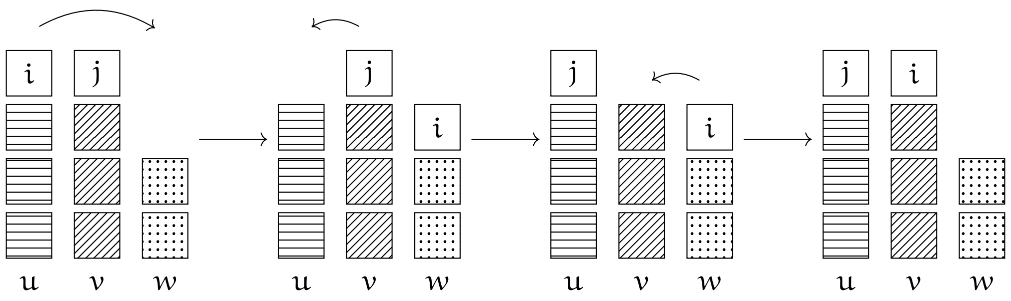


Figure 3.6: Transformation (ii), with non-full stack w.

Assume that i, j are the topmost items of stacks $u \neq v$. If there is a non-full stack w with $w \neq u$ and $w \neq v$, then move i to stack w, afterwards j to stack u, and finally i to stack v (cf. Figure 3.6). If there is no such stack w, for the number of free slots in stacks u, v we have $f_u + f_v = f$ and $f_u, f_v < f$, which implies $f_u \geqslant 1$ and $f_v \geqslant 1$. Thus, we may swap i and j using the topmost item h of another stack w as shown in Figure 3.7. Each indicated swap may be realized by a sequence of moves as in Figure 3.6.

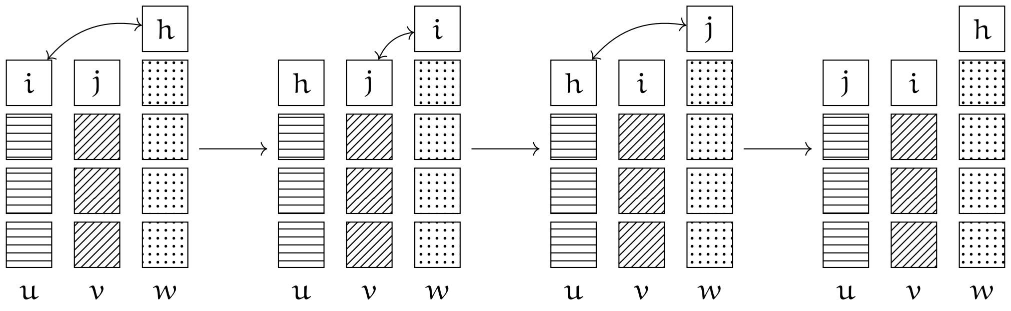


Figure 3.7: Transformation (ii), with full stack w.

(iii) The topmost item of each stack can be moved to any level above level $b - f$ in the same stack without changing the order of the other items in this stack and without changing the positions of all items in the remaining stacks.

The proof of this statement is similar to the proof of (i).

Having proved (i)-(iii), the claim of the lemma follows immediately. To achieve some arbitrary target configuration of the items in $\mathcal{I}^{top(f)}$, we first move each item to its destination stack by moving it to the topmost level of its stack as in (i) and then swapping it with a "wrong" item of its destination stack as in (ii). Afterwards, each item can be moved to the destination level in its stack as in (iii). □

For any stack $q \in \mathcal{Q}$ in a storage configuration we denote by $\gamma^{\mathcal{Q}}(q)$ the priority class of the item in level $b - f$ of stack q, i.e., of the topmost item in stack q which cannot be moved. W.l.o.g. we assume that the stacks $q_1, \ldots, q_m$ are sorted such that $\gamma^{\mathcal{Q}}(q_1) \geqslant \ldots \geqslant \gamma^{\mathcal{Q}}(q_m)$ and that the movable items from the set $\mathcal{I}^{top(f)} = \{i_1, \ldots, i_{f(m-1)}\}$ are sorted according to $\gamma(i_1) \geqslant \ldots \geqslant \gamma(i_{f(m-1)})$.

Example 3.3. *Consider the left configuration in Figure 3.8a with $f = 3$ where again the γ-values are shown. It can be transformed into the feasible configuration shown in the right by filling the stacks $q_1, \ldots, q_{m-1} = q_3$ with the items $i_1, \ldots, i_{f(m-1)} = i_9$ (in this order).*

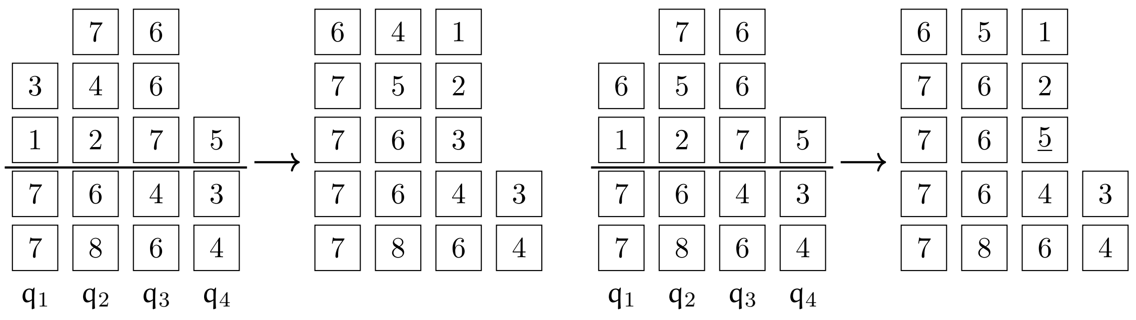
On the other hand, if we apply the same procedure to the left configuration in Figure 3.8b, we achieve the infeasible configuration shown in the right. The reason for the infeasibility is that for $k = 3$ we have $\gamma^{\mathcal{Q}}(q_3) = 4 < \gamma(i_7) = 5$. □

Theorem 3.1. *A given start configuration of the PMP with $m \geqslant 3$ stacks and $f \geqslant 1$ free slots can be transformed into a feasible configuration without blockings if and only if*

(a) there are no blocking items in the bottom levels $1, \ldots, b - f$, and

(b) $\gamma^{\mathcal{Q}}(q_k) \geqslant \gamma(i_{(k-1)f+1})$ for $k = 1, \ldots, m - 1$.

Proof: "⇒:" Assume that the conditions (*a*) and (*b*) are satisfied. Then we proceed as in Example 3.3. According to Lemma 3.1, the topmost levels $b - f + 1, \ldots, b$ of all stacks can be filled arbitrarily with items of $\mathcal{I}^{top(f)}$. Stack q_1 is filled with items $i_1, \ldots, i_f$ in levels $b - f + 1, \ldots, b$, then stack q_2 is filled with $i_{f+1}, \ldots, i_{2f}$, and so on. Finally, stack q_{m-1} is filled with $i_{(m-2)f+1}, \ldots, i_{(m-1)f}$. We claim that this configuration has no blocking items, i.e., it is a feasible premarshalling configuration.



(a) Feasible instance.

(b) Infeasible instance.

Figure 3.8: Feasible and infeasible PMP instances with $f = 3$.

Due to (*a*), there are no blockings in levels $1, \dots, b - f$. Due to (*b*), item $i_{(k-1)f+1}$ put on stack q_k in level $b - f + 1$ does not lead to a blocking since the item below does not have a smaller γ-value. Finally, in levels $b - f + 2, \dots, b$ there are no blockings since the items in $\mathcal{I}^{top(f)}$ are sorted according to non-increasing γ-values.

"$\Leftarrow$:" Conversely assume that the conditions are not satisfied. If (*a*) is violated, no configuration without blockings is possible since the corresponding items in the bottom levels cannot be moved. If (*b*) is violated, let $\bar{k}$ be the first index with $\gamma^{\mathcal{Q}}(q_{\bar{k}}) < \gamma(i_{(\bar{k}-1)f+1})$. Then there are only $(\bar{k} - 1)f$ feasible slots for the $(\bar{k} - 1)f + 1$ items $i_1, \dots, (\bar{k} - 1)f + 1$, which is not sufficient to place all of them. □

Checking feasibility with Theorem 3.1 can be done in $\mathcal{O}(n \log n)$ time. Condition (a) can be checked in $\mathcal{O}(n)$, while the $m - 1$ conditions in (b) require that all items in $\mathcal{I}^{top(f)}$ are sorted according to their priority values (which needs $\mathcal{O}(n \log n)$). In the following, we show that the effort can be reduced to $\mathcal{O}(n)$.

For this purpose, we call the value $\gamma^{\mathcal{Q}}(q)$ the "stack value" of stack q. Let m' be the number of different stack values and $\mathcal{Q}' := \{1, \dots, m'\}$ the set of corresponding indices. Furthermore, for all $\nu \in \mathcal{Q}'$ let γ^{Q}_{ν} be the νth stack value and

$$y_\nu := |\{q \in \mathcal{Q} : \gamma^{\mathcal{Q}}(q) = \gamma^{Q}_{\nu}\}| \tag{3.6}$$

be the number of stacks with stack value γ^{Q}_{ν}. Additionally, assume that these values are sorted such that $\gamma^{Q}_{\nu-1} < \gamma^{Q}_{\nu}$ for all $\nu = 2, \dots, m'$ and let

$$x_1 := |\{i \in \mathcal{I}^{top(f)} : \gamma(i) \leqslant \gamma^{Q}_{1}\}|, \tag{3.7}$$

$$x_\nu := |\{i \in \mathcal{I}^{top(f)} : \gamma^{Q}_{\nu-1} < \gamma(i) \leqslant \gamma^{Q}_{\nu}\}| \quad \text{for } \nu = 2, \dots, m'. \tag{3.8}$$

The value x_ν counts all items which may be put without blockings on the topmost item of any stack with stack value γ^{Q}_{ν}. Obviously, in a configuration without blockings, x_ν items must be put on stacks with stack values $\gamma^{Q}_{\nu}, \dots, \gamma^{Q}_{m'}$. Thus, an instance of the PMP can be transformed into a feasible configuration without blockings if and only if for each ν the accumulated stack capacities $f \sum_{\zeta=\nu}^{m'} y_\zeta$ provide enough free slots to store all items i with $\gamma(i) \leqslant \gamma^{Q}_{\nu}$, i.e., if we have

$$\sum_{\zeta=\nu}^{m'} x_\zeta \leqslant f \sum_{\zeta=\nu}^{m'} y_\zeta \quad \text{for all } \nu \in \mathcal{Q}'. \tag{3.9}$$

Calculating the x_ν- and y_ν-values as well as checking condition (a) from Theorem 3.1 and conditions (3.9) can be done in $\mathcal{O}(\max\{n, m\})$ as follows. At first we build an array A of length N to store references to tuples (x_ν, y_ν). Initially, all entries of the array are empty.

When checking condition (a) of Theorem 3.1, we know for each stack q the value $\gamma^{\mathcal{Q}}(q) = \gamma^{\mathcal{Q}}_{\nu}$ for some index $\nu \in \mathcal{Q}'$. Then we look up whether $A[\gamma^{\mathcal{Q}}(q)]$ is empty. If this is the case, we set $A[\gamma^{\mathcal{Q}}(q)] := \nu$ as a reference to the tuple (x_ν, y_ν) and set $y_\nu := 1$. Otherwise, we follow the reference and increment y_ν by one. Furthermore, we set $x_\nu := 0$ in both cases. Here, we traverse the stacks $1, \dots, m$ in an arbitrary order, which can be done in $\mathcal{O}(m)$.

After all stacks have been processed, we find the largest index $\gamma_{\mathtt{first}}$ with a non-empty reference in A. For $\gamma_{\mathtt{first}} - 1, \dots, 1$ in descending order we iteratively find the next empty entry $A[\gamma]$ and set it to $A[\gamma + 1]$. Subsequently, all entries $\gamma_{\mathtt{first}}, \dots, 1$ are filled, while all values $N, \dots, \gamma_{\mathtt{first}} + 1$ remain empty (which means that no stack exists that is able to store items of the corresponding larger priority values). This second step can be done in $\mathcal{O}(N)$.

Using the array A, we now traverse all items $i \in \mathcal{I}^{\mathtt{top}(f)}$ in an arbitrary order in $\mathcal{O}(n)$. For every item i we look up the entry of $A[\gamma(i)]$. If it is empty, the problem is infeasible since there is no stack on which item i may be placed without blockings. Otherwise, the corresponding entry x_ν is incremented by one. After all items have been processed, all x_ν- and y_ν-values have been calculated and conditions (3.9) can be checked in $\mathcal{O}(m)$.

3.3.2 Existence of Γ-robust configurations

In this section, we consider the question whether a Γ-robust configuration exists or not. Especially, we give some sufficient conditions for their existence. If we have a temporary storage area or at least $f \geqslant b$ free slots, then all items are movable and each configuration can be reached. On the other hand, if only relocations inside the storage area are allowed and $f < b$ holds, a Γ-robust configuration may exist, but may be not reachable. In the following, we ignore the latter problem and assume that sufficient space for relocations exists. This means that we do not need to consider a given start configuration.

At first, we consider storage areas with stacking height $b = 2$.

Theorem 3.2. *For $b = 2$ and any $\Gamma \geqslant 0$ the existence of a Γ-robust configuration can be decided in $\mathcal{O}(n^{2.5})$ by solving a maximum cardinality matching problem.*

Proof: We introduce a directed graph $G^\Gamma = (\mathcal{V}, \mathcal{A}^\Gamma)$ with n nodes modeling the items $\mathcal{I}$. Since the items in each priority class are interchangeable, we may fix one order for them and link all items of the same class by a single chain. Furthermore, we introduce arcs $(i, j) \in \mathcal{A}^\Gamma$ if $\gamma(j) - \gamma(i) > \Gamma$ with the meaning that i may be stacked on top of j in a Γ-robust configuration.

We calculate a maximum cardinality matching in the corresponding undirected graph (here an edge $\{i, j\}$ means that items i, j may be stacked together). Let m_1 be the number of edges in the matching and m_2 be the number of nodes not covered by the matching. It is easy to see that a Γ-robust solution exists if and only if $m_1 + m_2 \leqslant m$. A corresponding configuration can be obtained by putting the matched nodes in pairs into m_1 stacks, and the remaining m_2 nodes into separate stacks. Since the number of nodes is n, a maximum cardinality matching can be computed in $\mathcal{O}(n^{2.5})$ (cf. Even and Kariv [32]). □

In the following, we study the influence of the number of free slots on the existence of Γ-robust configurations.

Theorem 3.3. *If for any $\Gamma \geqslant 1$ there are at least $(b-1)\Gamma$ free slots, then a Γ-robust configuration always exists.*

Proof: We construct $\Gamma + 1$ chains of items, where chain σ_k contains all items i with $\gamma(i) \bmod (\Gamma + 1) \equiv k$ for $k = 1, \dots, \Gamma + 1$. All items in a chain can be robustly stacked together in a single stack, according to non-increasing priorities from bottom to top since then in each stack only items i, j with $\gamma(i) = \gamma(j)$ or $\gamma(j) - \gamma(i) > \Gamma$ are stacked on each other. We cut each chain into pieces of length b and a possible rest of length smaller than b and store each of these pieces in a separate stack. Let $m_{nf} \leqslant \Gamma + 1$ be the number of non-full stacks containing these rests. We claim that this configuration uses at most m stacks.

Since the storage contains n items and $f = mb - n \geqslant (b-1)\Gamma \geqslant (b-1)(m_{nf}-1)$ free slots, we must have

$$m = \frac{n+f}{b} = \left\lceil \frac{n+f}{b} \right\rceil \geqslant \left\lceil \frac{n+(b-1)(m_{nf}-1)}{b} \right\rceil = \left\lceil \frac{n-(m_{nf}-1)}{b} \right\rceil + (m_{nf}-1). \quad (3.10)$$

Let n_{nf} be the number of items contained in the non-full stacks and m_f be the number of full stacks. Then, $m_{nf} \leqslant n_{nf} \leqslant m_{nf}(b-1)$ and hence $n - n_{nf} = m_f b$ items are contained in the m_f full stacks of height b. It follows

$$\left\lceil \frac{n-(m_{nf}-1)}{b} \right\rceil + (m_{nf}-1) = \left\lceil \frac{(m_f b + n_{nf}) - (m_{nf}-1)}{b} \right\rceil + (m_{nf}-1) \quad (3.11)$$

$$= m_f + \left\lceil \frac{n_{nf} - (m_{nf}-1)}{b} \right\rceil + (m_{nf}-1) \quad (3.12)$$

$$\geqslant m_f + m_{nf} \quad (3.13)$$

since $n_{nf} \geqslant m_{nf}$ and therefore $\left\lceil \frac{n_{nf}-(m_{nf}-1)}{b} \right\rceil \geqslant 1$. Thus, the number of used stacks is at most

$$\left\lceil \frac{n-n_{nf}}{b} \right\rceil + m_{nf} = \left\lceil \frac{(m_f b + n_{nf}) - n_{nf}}{b} \right\rceil + m_{nf} = m_f + m_{nf} \overset{(3.10)\text{--}(3.13)}{\leqslant} m. \quad (3.14)$$

□

The following example shows that if there are less free slots, there are instances for which no Γ-robust configuration exists.

Example 3.4. *Consider a storage area with $m = \Gamma$ stacks and $n = \Gamma + 1$ items with (unique) priorities $\gamma(i) = i$ for $i = 1, \dots, n$. If the stacks have height $b \geqslant 2$, then we have $f = mb - n = (b-1)\Gamma - 1$ free slots. Since $\gamma(j) - \gamma(i) \leqslant \Gamma$ for all items $i \neq j$, in a Γ-robust solution all items must be stored in separate stacks, i.e., we need $n = \Gamma + 1 > m$ stacks.* □

Note that if for $\Gamma \geqslant 2$ the condition of Theorem 3.3 is satisfied (i.e., there are at least $(b-1)\Gamma$ free slots), then also $f \geqslant b$ holds since $(b-1)\Gamma \geqslant b \Leftrightarrow (\Gamma-1)b \geqslant \Gamma$ which is satisfied for $b \geqslant 2$ and $\Gamma \geqslant 2$. Thus, in this situation a Γ-robust solution is also always reachable. For $\Gamma = 1$ there are instances for which this is not possible.

Now we deal with the special situation of unique priorities (i.e., all priority classes contain exactly one single item), which occurs in many test instances from the literature. W.l.o.g. we assume $\gamma(i) = i$ for all $i \in \mathcal{I}$.

Theorem 3.4. *For unique priorities and any $\Gamma \geqslant 0$ the existence of a Γ-robust configuration can be decided in $\mathcal{O}(1)$.*

Proof: We distinguish three cases depending on the number of items.

- Case 1: $n \leqslant \Gamma + 1$
  Since then $\gamma(j) - \gamma(i) \leqslant \Gamma$ for all items $i \neq j$ holds, in a Γ-robust configuration all n items must be stored in separate stacks. Thus, a Γ-robust configuration exists if and only if $n \leqslant m$.

- Case 2: $\Gamma + 1 < n \leqslant (\Gamma + 1)b$
  For the first $\Gamma + 1$ items $i \neq j$ we have $\gamma(j) - \gamma(i) \leqslant \Gamma$, i.e., in a Γ-robust configuration these items must be stored in $\Gamma + 1$ different stacks. Now we arrange all items in

the chains σ_k for $k = 1, \dots, \Gamma + 1$. Recall that chain σ_k contains all items i with $\gamma(i) \bmod (\Gamma + 1) \equiv k$ and hence in a Γ-robust configuration all items in a chain can be stacked together in a single stack (according to non-increasing priorities from bottom to top). Since we consider the case of unique item priorities and $n \leqslant (\Gamma + 1)b$, all chains contain at most b items. Thus, exactly $\Gamma + 1$ stacks are required to store all items in a Γ-robust configuration and such a configuration exists if and only if $\Gamma + 1 \leqslant m$.

– Case 3: $n > (\Gamma + 1)b$
  In the following, we prove that for this case always a Γ-robust configuration exists by showing that $\lceil \frac{n}{b} \rceil$ stacks are sufficient (and $m \geqslant \lceil \frac{n}{b} \rceil$ must necessarily hold to store all items in the storage).

  Again, we consider the chains σ_k for $k = 1, \dots, \Gamma + 1$ and denote by λ_k the length of chain k. Due to $n > (\Gamma + 1)b$, we must have $\lambda_k \geqslant b$ for all chains. We claim that for any $1 \leqslant d \leqslant \lambda_k - 1$ the first d items of chain σ_k and all items except the first d items of chain σ_{k+1} can be stacked together. The priority value of the dth item of chain σ_k is $k + (\Gamma + 1)(d - 1)$ and the value of the $(d + 1)$th item of chain σ_{k+1} is $k + 1 + (\Gamma + 1)d$. Their difference satisfies

$$k + 1 + (\Gamma + 1)d - k - (\Gamma + 1)(d - 1) = \Gamma + 2 > \Gamma,$$

  i.e., in a Γ-robust configuration the corresponding items can be stacked onto each other. Moreover, if the $(d + 1)$th item of chain σ_{k+1} can be stacked onto the dth item of chain σ_k, then all previous respectively following items can also be stacked together.

  Now consider the following algorithm. We start with the first chain σ_1 and cut it into pieces of length b, starting from the end. If there is a rest of length $r_1 < b$ left, then we combine the corresponding first r_1 items of chain σ_1 with the last $b - r_1$ items of chain σ_2. Due to $\lambda_2 \geqslant b$ and since the first r_1 items of chain σ_1 can be combined with the last $\lambda_2 - r_1$ items of chain σ_2 as shown above, the length of the combination is $\lambda_2 - r_1 + r_1 = \lambda_2 \geqslant b$. Similarly, since $\lambda_k \geqslant b$ for every chain σ_k, the first items of chain σ_k can always be combined to a full stack with items of chain σ_{k+1} and this process can be iterated over all $\Gamma + 1$ chains. It results in $\lceil \frac{n}{b} \rceil - 1$ full stacks and one (possibly not completely filled) stack with $r_{\Gamma+1}$ items of the last chain. Hence, this algorithm always finds a Γ-robust configuration.

  We illustrate the algorithm at the following example with $n = 14, b = 3$ and $\Gamma = 2$. Then,

$$\begin{aligned}
\sigma_1 : &\quad 1 \to 4 \to 7 \to 10 \to 13 \\
\sigma_2 : &\quad 2 \to 5 \to 8 \to 11 \to 14 \\
\sigma_3 : &\quad 3 \to 6 \to 9 \to 12
\end{aligned}$$

  The algorithm constructs the pieces $(7, 10, 13)$, $(1, 4, 14)$, $(5, 8, 11)$, $(2, 9, 12)$, $(3, 6)$, using $\lceil \frac{n}{b} \rceil = 5$ stacks.

Obviously, the conditions of the three cases can be checked in $\mathcal{O}(1)$, since only the numbers n, m, b, Γ are involved. $\square$

Although Theorem 3.4 only applies to the case of unique priority classes, it can still be used if empty priority classes exist. If we ignore such empty priority classes, we can find a suitable configuration to the problem heuristically. Using the case distinction from Theorem 3.4, we can decide if this configuration already is a Γ-robust solution. Thus, we have a sufficient condition to check for the existence of Γ-robust solutions.

3.3.3 The adversary problem

In the following, we consider the complexity of the adversary problem. In the *decision version* (Adv) of this problem, we are given a stacking configuration c, an uncertainty set $\mathcal{U}^\Gamma$, and an integer B. The question is whether a scenario $\pi \in \mathcal{U}^\Gamma$ exists such that the number of badly placed items $BI(c^\pi)$ is at least B.

Theorem 3.5. *Problem Adv is $\mathcal{NP}$-complete, even if* $b = 2$.

Proof: First note that Adv is in $\mathcal{NP}$, since the number of badly placed items can be counted in polynomial time for any specific scenario π. A storage configuration with $b = 2$ can be fully encoded by a single matrix $Y \in \mathbb{N}^{N \times N}$, where $y_{k,k'}$ corresponds to the number of items belonging to priority class k that are stored below some other item belonging to priority class k'. Note that there is a one-to-one relationship between instances of Adv with $b = 2$ and matrices $Y \in \mathbb{N}^{N \times N}$ (ignoring the diagonal).

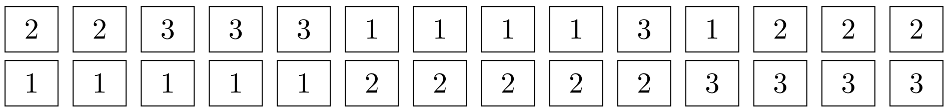


Figure 3.9: Example configuration.

For example, the stacking configuration in Figure 3.9 is encoded by the matrix

$$Y = \begin{pmatrix} 0 & 2 & 3 \\ 4 & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix}.$$

The number of badly placed items in such a configuration is given by the sum of upper diagonal entries of Y, i.e., $\sum_{k=1}^{N-1} \sum_{k'=k+1}^{N} y_{k,k'}$.

In the following, we construct an instance of Adv with Γ being sufficiently large such that $\mathcal{U}^\Gamma$ is the set of all possible permutations. For our reduction, we use the *linear ordering problem* (LOP), cf., e.g., Martí and Reinelt [81]. As input, it takes a 0-1 square matrix $A \in \{0, 1\}^{\ell \times \ell}$ and an integer V. The question is whether a permutation σ exists such that the sum of upper diagonal entries of the matrix $A(\sigma) = (a_{\sigma(i)\sigma(j)})$ is at least V.

Given a problem instance of (LOP), we build an instance of Adv by setting $Y = A$ and $B = V$. Since the input of (LOP) is a 0-1 matrix, the number of stacks in this stacking configuration is in $\mathcal{O}(\ell^2)$, i.e., polynomial in the input size. Furthermore, for a permutation π of priority classes, $Y(\pi) = (y_{\pi(k),\pi(k')})$ is the encoding of the resulting configuration. Hence, a linear ordering σ of value at least V exists if and only if there is a scenario π that results in at least $B = V$ badly placed items. $\square$

3.4 Optimization models

In this section, we introduce an IP model for a simplified robust premarshalling problem with the upper bound objective $\overline{BI}^\Gamma_{rob}(c)$ as well as an exact, iterative approach to solve the actual robust premarshalling problem with objective $BI^\Gamma_{rob}(c)$ (cf. Section 3.2). It is based on a master problem, where the uncertainty set is restricted, and the adversary subproblem, which aims at finding a worst-case scenario for the current solution. We again assume that sufficient space for relocations exists, i.e., all configurations are reachable from the given start configuration.

3.4.1 Minimizing the upper bound $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$

Recall that to calculate $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}(c)$ for a configuration c, we count all items i with at least one $j \in B_i(c)$ where $|\gamma(j) - \gamma(i)| \leqslant \Gamma$ holds, i.e., there exists a scenario in $\mathcal{U}^{\Gamma}$ in which i is badly placed. We use an IP formulation similar to one in Boge and Knust [8] where it is sufficient to assign items to stacks but not to specific levels. We claim that if an assignment of items to stacks is given, an optimal solution minimizing $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ may be obtained by sorting the items with respect to γ-values with smallest γ-value on top and largest γ-value on the bottom (i.e., this solution has no blockings in the nominal scenario). This property can easily be proved by an interchange argument as follows. Assume to the contrary that in a solution c minimizing $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ the items in a stack are not sorted. Then, in c there exists a pair of items (j, i) where j is stacked directly on top of i and $\gamma(j) > \gamma(i)$ holds, i.e., (3.1) is violated and j is badly placed in the nominal scenario. By interchanging i and j we get a new solution c' with $j \in B_i(c)$. If i satisfies $\gamma(j) - \gamma(i) \leqslant \Gamma$, in c' we again have one violation of (3.1); if i satisfies $\gamma(j) - \gamma(i) > \Gamma$, we may have one violation less (if there is no other item which causes j to be badly placed). Hence, $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}(c') \leqslant \overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}(c)$.

For each item $i \in \mathcal{I}$ let $J_i := \{j \in \mathcal{I} : \gamma(i) < \gamma(j) \text{ and } \gamma(j) - \gamma(i) \leqslant \Gamma\}$. Due to the above property we know that in an optimal solution minimizing $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ each item $j \in J_i$ may be placed below item i and hence i may become badly placed if j is stored in the same stack as i. For all $i \in \mathcal{I}$ and $q \in \mathcal{Q}$, we use two kinds of binary variables:

$$\beta_{iq} = \begin{cases} 1, & \text{if item } i \text{ is placed in stack } q \text{ as a badly placed item} \\ 0, & \text{otherwise.} \end{cases}$$

$$\alpha_{iq} = \begin{cases} 1, & \text{if item } i \text{ is placed in stack } q\text{, but is not badly placed} \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\alpha_{iq} + \beta_{iq} = 0$ is possible for an item i and stack q if item i is not placed in q at all. To model the robust premarshalling problem minimizing $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$, the following IP is used.

$$\begin{aligned}
(\mathrm{IP_{UP}}) \quad \min \sum_{i \in \mathcal{I}} \sum_{q \in \mathcal{Q}} \beta_{iq} & & & \quad (3.15)\\
\text{s.t.} \sum_{i \in \mathcal{I}} (\alpha_{iq} + \beta_{iq}) \leqslant b & & \forall q \in \mathcal{Q} & \quad (3.16)\\
\sum_{q \in \mathcal{Q}} (\alpha_{iq} + \beta_{iq}) = 1 & & \forall i \in \mathcal{I} & \quad (3.17)\\
\alpha_{iq} + \alpha_{jq} + \beta_{jq} \leqslant 1 & & \forall i \in \mathcal{I}, j \in J_i, q \in \mathcal{Q} & \quad (3.18)\\
\alpha_{iq} \in \{0, 1\} & & \forall i \in \mathcal{I}, q \in \mathcal{Q} & \quad (3.19)\\
\beta_{iq} \in \{0, 1\} & & \forall i \in \mathcal{I}, q \in \mathcal{Q} & \quad (3.20)
\end{aligned}$$

The objective function (3.15) minimizes the number of badly placed items, represented by the sum over all β-variables. Constraints (3.16) ensure that at most b items are assigned to each stack, while constraints (3.17) model that every item has to be assigned to exactly one stack. Finally, constraints (3.18) mean that if both items i and $j \in J_i$ are assigned to the same stack q, we must have $\alpha_{iq} = 0$.

3.4.2 First master problem

Instead of the full, implicitly given uncertainty set $\mathcal{U}^\Gamma$, we consider a smaller, explicitly given subset $\mathcal{U}' = \{\pi^1, \dots, \pi^T\} \subseteq \mathcal{U}^\Gamma$ of scenarios. We write $\mathcal{T} = \{1, \dots, T\}$. For each scenario $t \in \mathcal{T}$ let us denote by

$$\mathcal{B}_t = \{(k, k') \in \mathcal{P} \times \mathcal{P} : \pi_k^t > \pi_{k'}^t\}$$

all combinations of priority classes (k, k'), such that an item of priority class $k \in \mathcal{P}$ is badly placed in scenario t if it is above an item of priority class $k' \in \mathcal{P}$ in the same stack.

The following MIP generates a stacking configuration c which minimizes the number of badly placed items with respect to $\mathcal{U}'$. For all $k \in \mathcal{P}, q \in \mathcal{Q}, l \in \mathcal{L}$, we introduce variables

$$c_{kql} = \begin{cases} 1, & \text{if an item belonging to priority class } k \text{ is stored in stack } q \text{ at level } l \\ 0, & \text{otherwise} \end{cases}$$

and a variable $z \in \mathbb{N}$ counting the maximum number of badly placed items over all scenarios.

Then, we use the following MIP:

$$\begin{aligned}
&(\text{MIP}_{M^1}) \quad \min\ z && && (3.21)\\
&\text{s.t.} \sum_{q\in\mathcal{Q}} \sum_{l\in\mathcal{L}} y_{ql}^t \leqslant z && \forall t \in \mathcal{T} && (3.22)\\
&c_{kql'} + c_{k'ql} - 1 \leqslant y_{ql}^t && \forall (k,k') \in \mathcal{B}_t, q \in \mathcal{Q}, l \in \mathcal{L}, l' < l && (3.23)\\
&\sum_{k\in\mathcal{P}} c_{kql} \leqslant 1 && \forall q \in \mathcal{Q}, l \in \mathcal{L} && (3.24)\\
&\sum_{q\in\mathcal{Q}} \sum_{l\in\mathcal{L}} c_{kql} = |C_k| && \forall k \in \mathcal{P} && (3.25)\\
&\sum_{k\in\mathcal{P}} c_{kq,l+1} \leqslant \sum_{k\in\mathcal{P}} c_{kql} && \forall q \in \mathcal{Q}, l \in \mathcal{L} \setminus \{b\} && (3.26)\\
&c_{kql} \in \{0,1\} && \forall k \in \mathcal{P}, q \in \mathcal{Q}, l \in \mathcal{L} && (3.27)\\
&y_{ql}^t \in \{0,1\} && \forall q \in \mathcal{Q}, l \in \mathcal{L}, t \in \mathcal{T} && (3.28)\\
&z \geqslant 0 && && (3.29)
\end{aligned}$$

In (3.21) we minimize the maximum number of badly placed items over all scenarios, where the value of z is set through constraints (3.22). Variable y_{ql}^t (indicating a badly placed item in position (q, l) in scenario t) is set to one by constraints (3.23), using the precomputed sets $\mathcal{B}_t$. Constraints (3.24)–(3.26) model the stacking constraints on c. Recall that C_k denotes the set of items which are in the (nominal) priority class k.

3.4.3 Second master problem

We now introduce an alternative formulation for the same problem as in Section 3.4.2. We use the same constraints (3.24)–(3.26) to model the stacking configuration c. To count the number of badly placed items, we separate each position in each scenario using binary variables α_{kql}^t and β_{kql}^t with similar meanings as in Section 3.4.1. If the item in position (q, l) belongs to priority class k and is badly placed, then $\beta_{kql}^t = 1$. If the item in position (q, l) belongs to priority class k and is not badly placed, then $\alpha_{kql}^t = 1$. The resulting optimization model is as follows.

$$
\begin{aligned}
&(\text{MIP}_{\mathcal{M}^2}) \quad \min z && && (3.30)\\
\text{s.t. } &\sum_{k\in\mathcal{P}}\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{L}} \beta^t_{kql} \leqslant z && \forall t \in \mathcal{T} && (3.31)\\
&\sum_{k\in\mathcal{P}} c_{kql} \leqslant 1 && \forall q \in \mathcal{Q}, l \in \mathcal{L} && (3.32)\\
&\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{L}} c_{kql} = |C_k| && \forall k \in \mathcal{P} && (3.33)\\
&\sum_{k\in\mathcal{P}} c_{kq,l+1} \leqslant \sum_{k\in\mathcal{P}} c_{kql} && \forall q \in \mathcal{Q}, l \in \mathcal{L}\setminus\{b\} && (3.34)\\
&\alpha^t_{kql} + \beta^t_{kql} = c_{kql} && \forall k \in \mathcal{P}, q \in \mathcal{Q}, l \in \mathcal{L}, t \in \mathcal{T} && (3.35)\\
&\beta^t_{k'ql'} + \alpha^t_{k'ql'} + \sum_{l>l'} \alpha^t_{kql} \leqslant 1 && \forall (k,k') \in \mathcal{B}_t, q \in \mathcal{Q}, l' \in \mathcal{L}\setminus\{b\}, t \in \mathcal{T} && (3.36)\\
&c_{kql} \in \{0,1\} && \forall k \in \mathcal{P}, q \in \mathcal{Q}, l \in \mathcal{L} && (3.37)\\
&\alpha^t_{kql} \in \{0,1\} && \forall k \in \mathcal{P}, q \in \mathcal{Q}, l \in \mathcal{L}, t \in \mathcal{T} && (3.38)\\
&\beta^t_{kql} \in \{0,1\} && \forall k \in \mathcal{P}, q \in \mathcal{Q}, l \in \mathcal{L}, t \in \mathcal{T} && (3.39)\\
&z \geqslant 0 && && (3.40)
\end{aligned}
$$

3.4.4 The adversary subproblem

We now consider the adversary subproblem, which finds a worst-case scenario $\pi \in \mathcal{U}^\Gamma$ for a given candidate configuration c. By solving this problem, we can evaluate the objective function $BI^\Gamma_{rob}(c)$.

Note that we can measure the swap (Kendall-Tau) distance between any two permutations π and π' using

$$\Delta(\pi,\pi') = \sum_{k\in\mathcal{P}}\sum_{r>k} \delta_{kr}$$

where $\delta_{kr} = 1$ if $\pi_k < \pi_r$ and $\pi'_k > \pi'_r$, or if $\pi_k > \pi_r$ and $\pi'_k < \pi'_r$, i.e., priority classes k and r change their relative order. Thus, the set $\mathcal{U}^\Gamma = \{\pi \ : \ \Delta(\pi,\hat{\pi}) \leqslant \Gamma\}$ can also be described as

$$
\Big\{\psi \in \{0,1\}^{N\times N} : \begin{aligned}[t] &\sum_{k\in\mathcal{P}} \psi_{kr} = 1 \ \forall r \in \mathcal{P}\\
&\sum_{r\in\mathcal{P}} \psi_{kr} = 1 \ \forall k \in \mathcal{P}\\
&|\{(k,r) \in \mathcal{P}\times\mathcal{P} : k < r, \exists k' > r' \in \mathcal{P} : \psi_{kk'} = 1, \psi_{rr'} = 1\}|\\
&\leqslant \Gamma\Big\}\end{aligned}
$$

where $\psi_{kr} = 1$ if and only if $\pi_k = r$, i.e., priority class k is permuted to position r.

Using the solution of the master problem, we generate sets $\mathcal{L}_i$ for every item $i \in \mathcal{I}$ that contain all other items below i in the same stack. To model the subproblem as an IP, we introduce the following binary variables. Specifically, for all $i \in \mathcal{I}$ we introduce

$$
y_i = \begin{cases} 1, & \text{if item } i \text{ is badly placed}\\ 0, & \text{otherwise,}\end{cases}
$$

for all $i \in \mathcal{I}, j \in \mathcal{L}_i$ we introduce

$$h_{ij} = \begin{cases} 1, & \text{if item } i \text{ is badly placed above item } j \\ 0, & \text{otherwise,} \end{cases}$$

for all $k, r \in \mathcal{P}$ we introduce

$$\psi_{kr} = \begin{cases} 1, & \text{if priority class } k \text{ is permuted to position } r \\ 0, & \text{otherwise,} \end{cases}$$

and finally for all $k < r \in \mathcal{P}$ we introduce

$$\psi_{kr} = \begin{cases} 1, & \text{if priority classes } k \text{ and } r \text{ change their relative order} \\ 0, & \text{otherwise.} \end{cases}$$

Note that the resulting scenario π can be derived from the ψ-values. Recall that $\gamma(i) \in \mathcal{P}$ is the nominal priority class of item $i \in \mathcal{I}$. The adversary subproblem can then be modeled in the following way:

$$(\text{IP}_{\text{SUB}}) \quad \max \sum_{i \in \mathcal{I}} y_i \tag{3.41}$$

$$\text{s.t. } y_i \leqslant \sum_{j \in \mathcal{L}_i} h_{ij} \qquad \forall i \in \mathcal{I} \tag{3.42}$$

$$(N+1)h_{ij} - N \leqslant \sum_{k \in \mathcal{P}} k(\psi_{\gamma(i)k} - \psi_{\gamma(j)k}) \qquad \forall i \in \mathcal{I}, j \in \mathcal{L}_i \tag{3.43}$$

$$\sum_{k \in \mathcal{P}} \psi_{kr} = 1 \qquad \forall r \in \mathcal{P} \tag{3.44}$$

$$\sum_{r \in \mathcal{P}} \psi_{kr} = 1 \qquad \forall k \in \mathcal{P} \tag{3.45}$$

$$\sum_{k' \in \mathcal{P}} k'(\psi_{kk'} - \psi_{rk'}) \leqslant N\delta_{kr} \qquad \forall k, r \in \mathcal{P}, k < r \tag{3.46}$$

$$\sum_{k \in \mathcal{P}} \sum_{r > k} \delta_{kr} \leqslant \Gamma \tag{3.47}$$

$$y_i \in \{0,1\} \qquad \forall i \in \mathcal{I} \tag{3.48}$$

$$h_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{I}, j \in \mathcal{L}_i \tag{3.49}$$

$$\psi_{kr} \in \{0,1\} \qquad \forall k, r \in \mathcal{P} \tag{3.50}$$

$$\delta_{kr} \in \{0,1\} \qquad \forall k, r \in \mathcal{P}, k < r \tag{3.51}$$

The aim of the subproblem is to maximize the number of badly placed items, cf. objective (3.41). Constraints (3.42) ensure that y_i is only one if at least one of the h_{ij} variables is one. Constraints (3.43) model that h_{ij} can only be set to one if j is placed below i (using the precomputed sets $\mathcal{L}_i$) and item j must leave the stack before i. Constraints (3.44–3.47) model the uncertainty set $\mathcal{U}^\Gamma$, based on the above formulation of the Kendall-Tau distance.

3.4.5 Iterative method

Using the master and adversary problem formulations, we can solve the robust premarshalling problem with objective $\text{BI}^\Gamma_{rob}(c)$ as follows. We begin with an arbitrary scenario set $\mathcal{U}'$, e.g., by only including the nominal scenario. Solving the master problem then gives a candidate solution c, along with an underestimation of its objective value (as the master problem is a relaxation of the robust premarshalling problem). We evaluate c by solving the corresponding adversary subproblem. If both objective values are equal, we have found an optimal solution. Otherwise, we add the scenario generated by the subproblem to $\mathcal{U}'$ and repeat the process. Note that this method stops after a finite number of iterations, as $\mathcal{U}^\Gamma$ contains a finite number of scenarios, and each iteration produces a new scenario.

3.5 Computational experiments

We conducted four sets of experiments to evaluate the models proposed in this chapter. In the first experiment, we focus on the computational effort to calculate robust solutions. Then, in the second experiment, we check the effectiveness of Theorems 3.3 and 3.4 when calculating the optimal objective value of BI^Γ_{rob}. The purpose of the third experiment is to analyze the impact of buffer times in the retrieval sequence. In the last experiment, we consider the gain and price of robustness.

3.5.1 Instances

We use four test instance sets from the premarshalling literature, denoted as “Caserta”, “Forster”, “Hottung”, and “Tanaka”, respectively. The instances of Caserta et al. [22] are commonly used in the premarshalling literature, but have only unique item priorities (i.e., each priority class contains exactly one single item). On the other hand, the instances from Bortfeldt and Forster [14], Hottung et al. [49] as well as Tanaka and Tierney [97] have priority classes containing several items. Table 3.1 shows some characteristics of these instances, namely the total number of instances in each set and intervals representing minimum and maximum values for the respective parameters.

Table 3.1: Characteristics of test instances.

Set	# Inst.	m	b	n	N
Caserta	880	$[3,10]$	$[5,12]$	$[9,100]$	$[9,100]$
Forster	681	$[10,20]$	$[5,8]$	$[35,128]$	$[10,52]$
Hottung	4500	$[5,10]$	$[7,7]$	$[25,50]$	$[8,50]$
Tanaka	960	$[3,9]$	$[4,6]$	$[6,37]$	$[2,6]$

For some instances of the Tanaka set, there were empty priority classes. To control the effect of such empty priority classes, we adjusted these instances by reducing the number of priority classes N such that there is at least one item per priority class. In Experiment 3, we insert empty priority classes for all instance sets in the same way.

Note that in all benchmark instances from the literature there are $f \geqslant b$ free slots, i.e., every configuration can be reached from any other one.

3.5.2 Experiment 1: Computation times

Setup

The aim of this experiment is to compare the performance of models (MIP_{M^1}) and (MIP_{M^2}) for the master problem (cf. Sections 3.4.2 and 3.4.3) on the one hand, and to compare the performance of the IP (IP_{SUB}) for the subproblem with a brute-force full enumeration approach to solve the adversary problem on the other hand (in our experiments, we used a recursive implementation of the plain changes algorithm described in Knuth [66]).

To calculate a robust premarshalling configuration, the original storage positions of items in the problem instances are ignored. Thus, if we do not consider the number of reshuffles, instances can be treated as equivalent if they coincide in the parameters n, m, b, N, and $|C_k|$ for all $k \in \mathcal{P}$. For this experiment, we only use one representative instance per equivalence class. This reduces the test set to 21 instances for Caserta, 681 instances for Forster, 9 instances for Hottung, and 674 instances for Tanaka. We solved each instance five times, using $\Gamma \in \{1, \ldots, 5\}$.

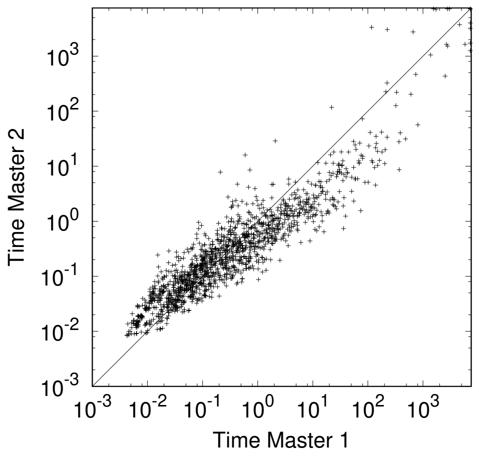
We ran our iterative algorithm for each instance using both models for the master problem and both models for the subproblem. While we record the solution times of both methods, we make use of only one of the master problem solutions, and one of the subproblem solutions. This way we avoid differences in computation times that may arise when different solutions are found in the master or in the subproblem (note that optimal solutions are not necessarily unique). To start the iterative process, the first master solution is generated using a greedy procedure. Note that in case this solution is already optimal, no other runs for the master problem are required.

Experiments were performed on a virtual Ubuntu server with ten Xeon CPU E7-2850 processors at 2.00 GHz speed and 23.5 GB RAM. MIPs were solved with CPLEX 12.8 using one thread. We used a time limit of 2 hours for the master problem.

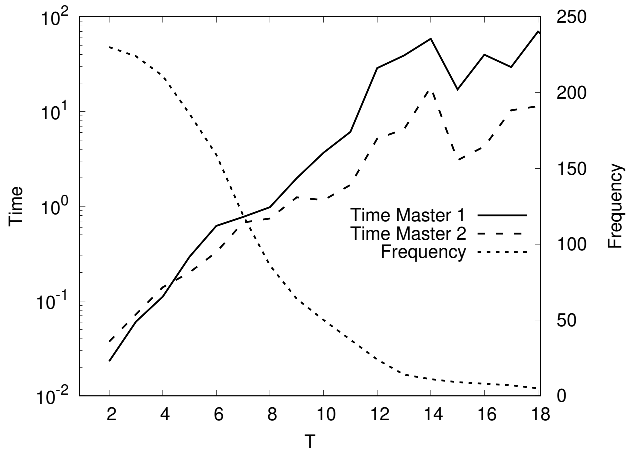
Results

We first discuss the results regarding the master problem as presented in Figure 3.10. Recall that we solve 1385 instances for five values of Γ. Each solution process may require multiple iterations. However, the first master problem is solved using a greedy procedure. Overall, our experiment resulted in 1448 data points where (MIP_{M^1}) and (MIP_{M^2}) can be compared.

Each computation time measurement (in seconds) is shown in the scatter plot in Figure 3.10a. Points on the diagonal indicate that both MIPs required the same time to solve; points below the diagonal indicate that solving (MIP_{M^2}) was faster; points above mean that solving (MIP_{M^1}) was faster. Note the logarithmic axes. Only for instances that can be solved fast anyway, the first model is more efficient, while we can observe a clear trajectory for slower instances that runs beneath the diagonal.



(a) All computation times (in seconds).



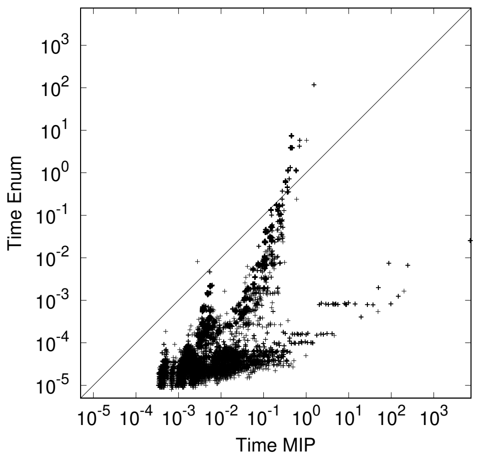
(b) Median computation time depending on the number T of scenarios.

Figure 3.10: Results for the master problem.

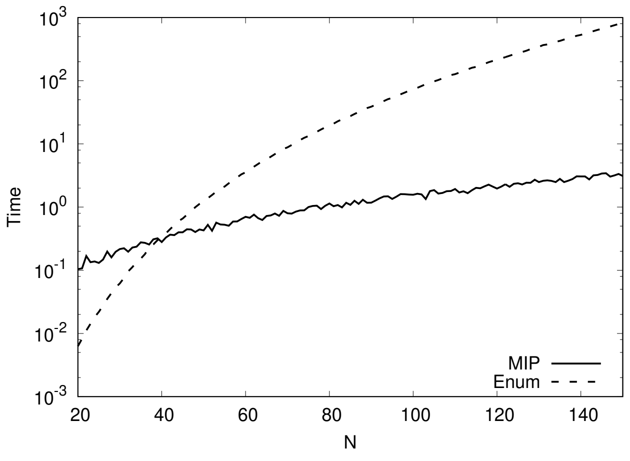
To understand these differences in more detail, we calculated the median computation times depending on the number T of scenarios included in the master problem (cf. Section 3.4.2).

This gives an indication to the problem size. Larger values of T mean that more iterations are required to solve the robust optimization problem. The results are presented in Figure 3.10b. The curve labeled "Frequency" indicates how often master problems with T scenarios were solved. As can be seen, the number of computation times over which we take the median becomes very small for large values of T, and hence numbers become less reliable. We can see that only for $T \leqslant 4$ the first master problem MIP has a slight advantage over the second.

In Figure 3.11, we present results on the comparison between using the IP (IP_{SUB}) to solve the adversary subproblem versus using a full enumeration. In total, the subproblem was solved 8289 times by each method (1385 instances with five values of Γ, where some instances required multiple iterations). Figure 3.11a shows a scatter plot of all 8289 data points. In the vast majority of cases, it is faster to enumerate all permutations instead of calculating a worst-case scenario using the IP. This is to be expected for small values of N, where the time overhead of generating the IP cannot compete with enumeration times in the order of 10^{-5} seconds.



(a) All computation times (in seconds).



(b) Computation time depending on the number N of priority classes.

Figure 3.11: Results for the subproblem.

However, we can also observe a trajectory of points that moves above the diagonal. Of the 140 points above the diagonal, only one corresponds to a Tanaka instance. At the same time, 4687 out of all 8289 points correspond to Tanaka instances. This means that the points above the diagonal correspond unproportionally to the problem sets with a large number of priority classes N. To understand this effect in more detail, we created an additional artificial set of instances with $b = 5$, $m = \lceil n/5 + 2 \rceil$, and $n = N$ for $N = 20$ to 150 (note that an instance is fully defined by these parameters). In Figure 3.11b, we show the resulting computation times for the subproblem with $\Gamma = 5$ and different values of N. For values $N \geqslant 40$, using the IP approach begins to outperform the full enumeration of scenarios. This difference becomes even more distinct when fixing N and increasing Γ. For $\Gamma = 5$ and $N = 40$, the IP approach takes 0.2 seconds and the enumeration takes 0.3 seconds. Increasing Γ to 9 results in 0.8 seconds and 445.4 seconds, respectively (recall that the number of elements in $\mathcal{U}^{\Gamma}$ grows exponentially in Γ).

To summarize these findings, both master problem formulations are competitive, with an asymptotic advantage for (MIP_{M^2}). For most subproblems that we considered here, enumerating all permutations was faster than determining a worst-case scenario by solving problem (IP_{SUB}). However, as N and Γ increase, enumeration becomes an infeasible option.

3.5.3 Experiment 2: Checking the existence of Γ-robust configurations and calculating the optimal value of $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$

Setup

While the first experiment focused on computation times, we now evaluate the usefulness of the existence criteria for Γ-robust configurations. To this end, we use the following setup for $\Gamma \in \{1, \ldots, 5\}$: For each of the representative instances from Experiment 1, we check if Theorem 3.3 or 3.4 proves the existence of a Γ-robust configuration. If this is the case, we know that $\mathrm{BI}^{\Gamma}_{\mathrm{rob}} = \overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} = 0$ for the optimal values. If none of the theorems can prove the existence of a Γ-robust configuration, we calculate the upper bound $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ using the model ($\mathrm{IP}_{\mathrm{UP}}$). If we find $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} \in \{0, 1\}$, then also $\mathrm{BI}^{\Gamma}_{\mathrm{rob}} = \overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ for the optimal values due to (3.4) and (3.5). Finally, only if $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} \geqslant 2$, we solve the robust problem using the iterative algorithm to find the optimal value of $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$.

Results

Table 3.2 shows for each value of $\Gamma \in \{1, \ldots, 5\}$, in how many cases the theorems could prove the existence of a Γ-robust configuration, the upper bound model was sufficient to compute the optimal value of $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$, or the iterative method had to be used (all values in percent). Values per row can add up to more than 100% if both Theorems 3.3 and 3.4 prove the existence.

Table 3.2: Calculating $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$, cases in %.

Γ	Th. 3.3	Th. 3.4	($\mathrm{IP}_{\mathrm{UP}}$)	Iter.
1	100.00	2.09	0.00	0.00
2	88.12	2.09	11.16	0.14
3	78.69	1.87	18.29	1.80
4	54.93	1.58	39.52	4.18
5	35.57	1.22	57.60	5.69

It can be seen that especially for small values of Γ, the application of optimization models can be avoided by using our theorems. Furthermore, even if they are not successful, in the majority of cases the upper bound model ($\mathrm{IP}_{\mathrm{UP}}$) is already sufficient to calculate the optimal value of $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$. Even for the highest value $\Gamma = 5$, in only 5.69% of cases we actually require the iterative solution method to find an optimal robust configuration, highlighting the strength of our theoretical analysis.

We show average objective values $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$ for different values of Γ in Table 3.3. For each instance set we take the average over all instances (column "All") and only over those instances where the iterative solution method was required (column "Iter."). Note that, while $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$ is non-decreasing in Γ for a single problem instance, average values in column "Iter." can decrease, as the set over which the average is taken changes. It can be seen that objective values are small overall, and most instances allow configurations with only few possible conflicts. The tradeoff in reaching these robust solutions with potentially more reshuffles is considered in the next experiment.

Table 3.3: Average and maximum values of BI^{Γ}_{rob}.

	Caserta			Forster			Hottung			Tanaka		
Γ	All	Iter.	Max	All	Iter.	Max	All	Iter.	Max	All	Iter.	Max
1	0.00	-	0	0.00	-	0	0.00	-	0	0.00	-	0
2	0.00	-	0	0.00	-	0	0.00	-	0	0.02	2.00	2
3	0.05	1.00	1	0.00	1.00	1	0.00	-	0	0.12	1.52	2
4	0.19	1.00	1	0.00	1.00	1	0.00	-	0	0.22	1.94	4
5	0.38	1.14	2	0.01	1.33	2	0.67	2.00	3	0.30	2.08	4

3.5.4 Experiment 3: Impact of empty priority classes

Setup

In the previous experiments, no empty priority classes exist, i.e., $|C_k| \geqslant 1$ for all priority classes $k \in \mathcal{P}$. This means that swapping any two neighboring priority classes creates the same costs for the adversary. The existence of empty priority classes indicates buffer times between retrievals, making a swap of non-empty priority classes more expensive for the adversary (cf. Section 3.2). In this experiment we include such buffers randomly in the data and measure the impact on the robust objective value BI^{Γ}_{rob}. We restrict our analysis to $\Gamma = 5$, which has the largest number of potential conflicts in the previous experiment.

We conduct two sets of experiments. In the first setup, we use the same instances as in the previous experiments (except for two instances from the Hottung set due to high computation times). Let E be the number of priority classes without items. For $E \in \{0, 1, \dots, 10\}$, we add E many priority classes at random uniformly into each instance. Empty priority classes cannot be the first or last priority class of an instance (as this would not change the problem). It is possible that multiple empty priority classes appear in sequence. For each instance and each value E, we repeat this 100 times and measure the minimum, average, and maximum objective value BI^{Γ}_{rob}.

In the second setup, we only consider those instances from the Tanaka set with $N = 6$. There are 201 such instances. Instead of adding the empty priority classes at random, we insert them at a specific position and analyze the effect of changing this position.

While the first setup is designed to give us insight into the effect the number of empty priority classes E has on the robust objective value, the second setup sheds more light on the effect of the position of such empty priority classes.

Results

In Table 3.4 we present a summary of results for the first setup. Recall that the minimum and maximum values are taken over the 100 repetitions per instance, respectively, while the reported values are averaged over all instances. We do not show results for the Forster set of instances, as all objective values for $E \geqslant 1$ have been equal to zero.

In the first row, $E = 0$ and no empty priority classes are inserted. This row corresponds to the row $\Gamma = 5$ of Table 3.3 (values for Hottung differ due to the two instances ignored in this experiment). It can be seen that having empty priority classes (i.e., buffer times in the retrieval sequence) reduces the number of potential conflicts in the unloading phase. The strength of this effect depends on the instance set. For Hottung instances, an effect is only recognizable for $E \geqslant 6$, while the Tanaka instances profit already from the first empty priority class. The difference can be explained with the different instance sizes. While Tanaka

Table 3.4: Average of minimum, average, and maximum values of $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$ for different numbers of empty priority classes E.

	Caserta			Hottung			Tanaka		
E	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
0	0.381	0.381	0.381	0.143	0.143	0.143	0.295	0.295	0.295
1	0.381	0.381	0.381	0.143	0.143	0.143	0.197	0.218	0.228
2	0.333	0.366	0.381	0.143	0.143	0.143	0.123	0.141	0.178
3	0.286	0.349	0.381	0.143	0.143	0.143	0.021	0.087	0.131
4	0.238	0.337	0.381	0.143	0.143	0.143	0.000	0.054	0.108
5	0.238	0.323	0.381	0.143	0.143	0.143	0.000	0.040	0.107
6	0.143	0.307	0.381	0.000	0.141	0.143	0.000	0.030	0.104
7	0.143	0.292	0.381	0.000	0.139	0.143	0.000	0.023	0.104
8	0.143	0.275	0.381	0.000	0.131	0.143	0.000	0.017	0.101
9	0.048	0.256	0.381	0.000	0.134	0.143	0.000	0.013	0.098
10	0.048	0.245	0.381	0.000	0.126	0.143	0.000	0.010	0.098

instances have comparatively small values of n and N, they are larger for the Caserta and Hottung data set.

This experiment shows that empty priority classes occurring in the data are important information for the solving process that should be taken into account in order to select the best possible robust configuration. It is also apparent that not only the number of empty priority classes, but also their placement has an effect on the robust objective value. For example, with $E = 4$ all Tanaka instances can have zero $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$ in the best-case placement, while a worst-case placement results in a value of 0.108 on average.

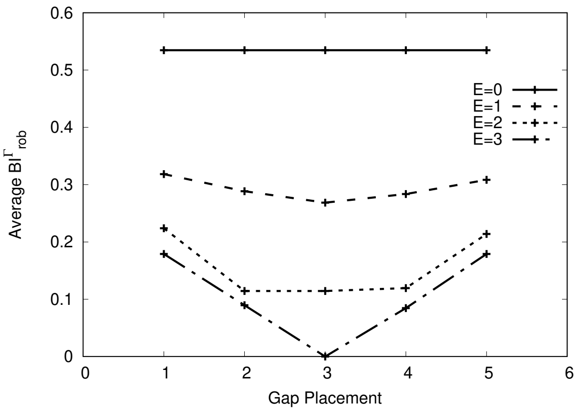


Figure 3.12: Average value $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ for different values of E and gap placement.

This effect is considered in more detail in the second setup. In Figure 3.12, we summarize these results. On the horizontal axis, we denote the position of the empty priority classes (after the kth original priority class for k from 1 to 5). Multiple empty priority classes are put into the same position. For $E = 0$, no empty priority classes are inserted and the resulting line is horizontal. For $E \geqslant 1$, there is a clear advantage if the empty priority classes are put into the middle. For $E = 3$, we can even reach an objective value of zero for all of the 201 instances considered here. We also considered $E = 4, 5$, but this did not further improve objective values. We can also see that the largest drop in objective values is between $E = 0$ and $E = 1$. The incremental effect of increasing E seems to reduce, the larger E already is.

Overall, this experiment demonstrates that buffer times should be included in the planning, if available, as even small buffers can significantly reduce the number of potential conflicts during the unloading phase. Our models can include empty priority classes without requiring any modifications. It is an interesting question for further research where to put such buffer times, if it is possible for the planner to influence the retrieval times of items.

3.5.5 Experiment 4: Gain and price of robustness

Setup

In our final experiment, we explore the quality of the robust solutions we compute. Note that there are two relevant criteria for solution quality: the level of robustness of the computed final configuration as well as the number of reshuffles required to transform the start into the final configuration. Since in the benchmark data due to $f \geqslant b$ every configuration can be transformed into any other one, the optimal values of BI^{Γ}_{rob} presented in Section 3.5.3 can always be achieved.

To calculate the required number of reshuffles, we use the branch-and-bound algorithm of Tanaka and Tierney [97] for the nominal case, and also adapted this method to search for a solution with minimum value $\overline{BI}^{\Gamma}_{rob}$ computed in Experiment 2. To distinguish the different algorithms, we refer to the original branch-and-bound algorithm as BB and our adapted algorithm as BB-Γ_{al}. The parameter $\Gamma_{al} \in \{1, \dots, 5\}$ is given to the algorithm prior to the solving process and restricts the algorithm to regard only configurations having violations of (3.1) with value at most Γ_{al}. Beside the algorithm parameter Γ_{al}, the evaluation parameter $\Gamma_{ev} \in \{1, \dots, 5\}$ is used to evaluate a final configuration c computed by an algorithm (using $\overline{BI}^{\Gamma_{ev}}_{rob}(c)$ respectively $BI^{\Gamma_{ev}}_{rob}(c)$). As an example, the algorithm BB-3 may find a configuration c with $\overline{BI}^{3}_{rob}(c) = 0$ which would be the best achievable value, but nevertheless $\overline{BI}^{3}_{rob}(c) < \overline{BI}^{4}_{rob}(c) < \overline{BI}^{5}_{rob}(c)$ could be valid evaluations for the same configuration.

For final configurations c_{BB} computed by BB and $c_{BB\text{-}\Gamma_{al}}$ computed by BB-Γ_{al}, we define the following performance metrics:

$$\Delta\overline{BI}^{\Gamma_{ev},\Gamma_{al}}_{rob} := \overline{BI}^{\Gamma_{ev}}_{rob}(c_{BB}) - \overline{BI}^{\Gamma_{ev}}_{rob}(c_{BB\text{-}\Gamma_{al}}) \tag{3.52}$$

$$\Delta BI^{\Gamma_{ev},\Gamma_{al}}_{rob} := BI^{\Gamma_{ev}}_{rob}(c_{BB}) - BI^{\Gamma_{ev}}_{rob}(c_{BB\text{-}\Gamma_{al}}) \tag{3.53}$$

$$\Delta RS^{PM,\Gamma_{al}} := RS^{PM}(c_{BB\text{-}\Gamma_{al}}) - RS^{PM}(c_{BB}) \tag{3.54}$$

Here $RS^{PM}(c)$ denotes the number of reshuffles needed to reach configuration c. If the values in (3.52) or (3.53) are greater than zero, the configurations computed by BB-Γ_{al} are more robust than the nominal configurations computed by BB. On the other hand, if the value (3.54) is positive, a solution computed by BB-Γ_{al} needs more reshuffles RS^{PM} in comparison to the nominal solution computed by BB.

These experiments were carried out on a Linux server (Ubuntu 16.04.6 LTS) with four Intel i5-3470 processors at 3.20 GHz speed and 19.5 GB RAM. We used a time limit of 2 hours per run of one algorithm.

Results for Tanaka data set

We first focus on the Tanaka data set since this is the most diverse data set in terms of different parameter settings. The instances were originally grouped by N, b, m, and the filling degree ($fd := \frac{n}{mb}$) in 48 different categories of 20 instances each. Since the number of priority classes N differs from instance to instance in these categories and the number of stacks m has not a visible effect, in the results we merge these instances and distinguish only the parameters b

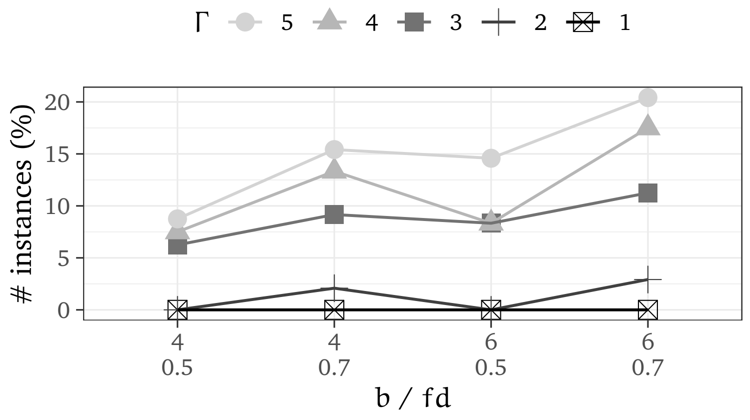
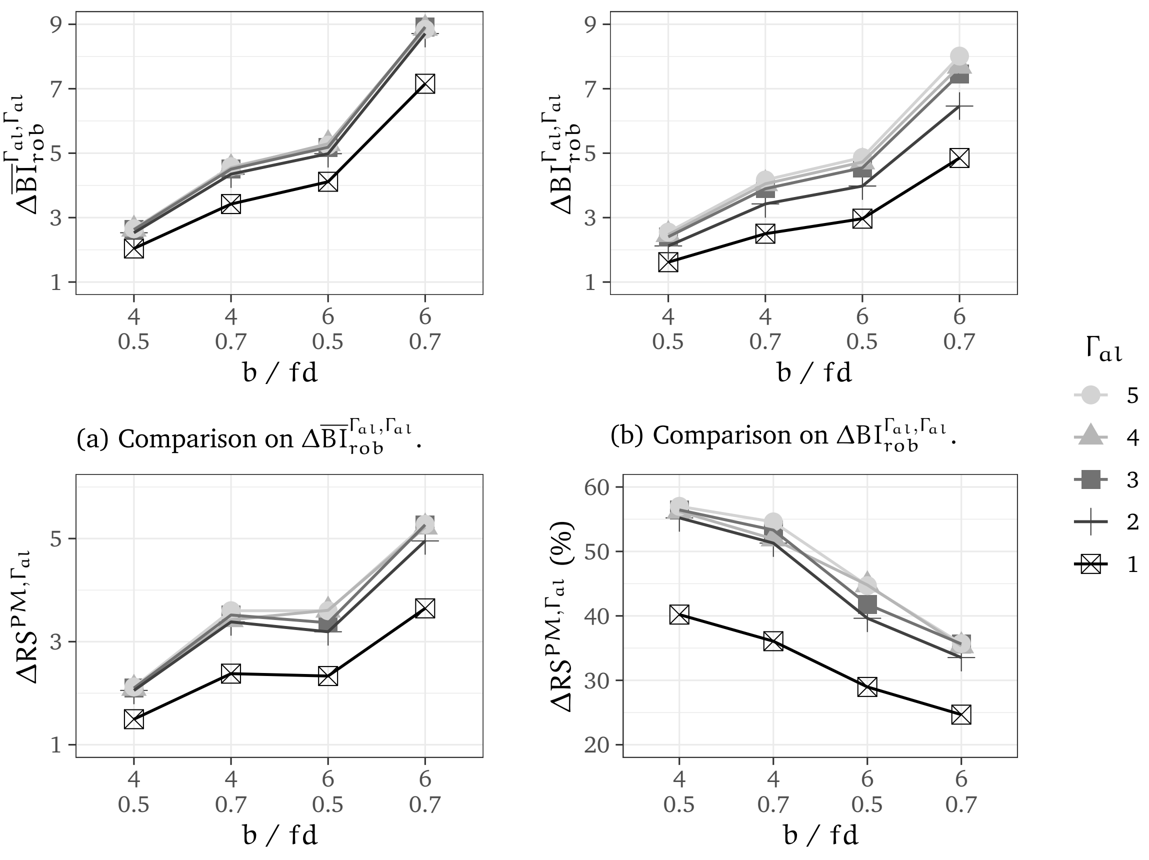


Figure 3.13: Number of instances satisfying $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} > 0$ for $\Gamma \in \{1, \dots, 5\}$ (in %) in the Tanaka data set.

and fd. We call such a category with the same parameters an "instance group". We decided to use line charts to present the results since the overall trend of the data points is the important outcome of these experiments. In the following, we refer to the lines connecting data points as "trend lines".



(a) Comparison on $\Delta\overline{\mathrm{BI}}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$.

(b) Comparison on $\Delta\mathrm{BI}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$.

(c) Comparison on absolute $\Delta\mathrm{RS}^{\mathrm{PM},\Gamma_{\mathrm{al}}}$.

(d) Comparison on percentage $\Delta\mathrm{RS}^{\mathrm{PM},\Gamma_{\mathrm{al}}}$.

Figure 3.14: Results of BB-Γ_{al} for different instance groups of the Tanaka data set.

Figure 3.13 shows the number of instances (in %) where the IP formulation described in Section 3.4.1 provides an optimal configuration with $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} > 0$ for $\Gamma \in \{1, \dots, 5\}$, i.e., a configuration which is not Γ-robust. Each line corresponds to one choice of Γ. The number of instances with a value greater than zero increases with b and fd.

Note that in BB-Γ_{al} the lower bound to estimate the number of reshuffles $\mathrm{RS}^{\mathrm{PM}}$ provides a valid lower bound value only in the case $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} = 0$. Otherwise, these values may overestimate the given configuration such that some subtrees may be be cut off too early. Hence, the computed number of reshuffles can only be guaranteed to be optimal for a Γ-robust configuration. The results of Section 3.5.3 revealed that for 350 among the the $4800 = 960 \cdot 5$ combinations of instances and values for the parameter $\Gamma_{\mathrm{al}} \in \{1, \dots, 5\}$ there is no Γ-robust configuration.

It turned out that instances differ visibly in the results if

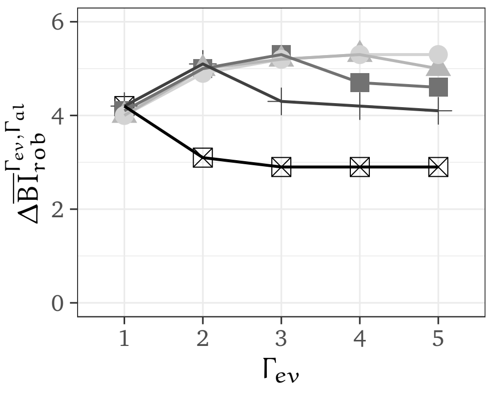
$$\text{there is some } \Gamma \in \{1, \dots, 5\} \text{ with } \overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} > 0. \tag{3.55}$$

Except for 3 among the 4450 combinations of instances and values for the parameter $\Gamma_{\mathrm{al}} \in \{1, \dots, 5\}$ with $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} = 0$, every combination could be verified to be optimally solved by BB-Γ_{al} within less than 20 minutes. For $\Gamma_{\mathrm{al}} = 5$, two times the time limit was exceeded; however, these instances could be solved by extending the time limit to 4.5 hours. For $\Gamma_{\mathrm{al}} = 3$, once the time limit was exceeded and had to be extended to 19.5 hours.

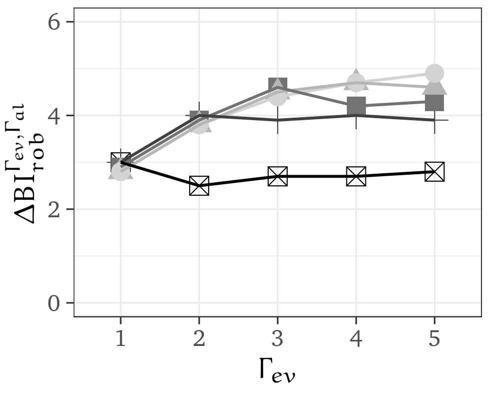
Figure 3.14 displays results for the algorithm BB-Γ_{al} with $\Gamma_{\mathrm{al}} \in \{1, \dots, 5\}$ compared to the original algorithm BB. The charts show average values of 240 instances grouped by b and fd on the horizontal axis, while the vertical axis indicates $\Delta\overline{\mathrm{BI}}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$, $\Delta\mathrm{BI}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$, and $\Delta\mathrm{RS}^{\mathrm{PM},\Gamma_{\mathrm{al}}}$ (absolute and percentage). Note that here each configuration is evaluated using the same value of Γ as used for the optimization.

Although Figure 3.14a reveals a greater reduction of $\Delta\overline{\mathrm{BI}}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$ for increasing values of Γ_{al}, the gap between two consecutive trend lines $(\Gamma_{\mathrm{al}}, \Gamma_{\mathrm{al}} + 1)$ decreases. Moreover the chart displays a higher potential for improvements for a greater stack height b and filling degree fd. As expected, Figure 3.14b shows similar results with overall smaller values since $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}}$ is an upper bound on $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$. Thus, we find that minimizing the upper bound objective leads to appropriate results also for the objective $\mathrm{BI}^{\Gamma}_{\mathrm{rob}}$. While robustness can be increased, Figure 3.14c shows that the absolute number of additional reshuffles increases moderately in a similar way like $\Delta\overline{\mathrm{BI}}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$ and $\Delta\mathrm{BI}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$. While the absolute number of reshuffles increases with b and fd, the relative number of additional reshuffles decreases, as depicted in Figure 3.14d. Note that the trend lines of $\Delta\mathrm{RS}^{\mathrm{PM},\Gamma_{\mathrm{al}}}$ lie below the trend lines of $\Delta\mathrm{BI}^{\Gamma_{\mathrm{al}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$ which means that the additional reshuffles required for robust premarshalling can be saved in a worst-case scenario of the unloading process. Hence, the added costs in the premarshalling phase when using a robust solution pay off later.

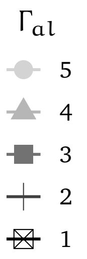
While the presented results for $\Gamma_{\mathrm{ev}} = \Gamma_{\mathrm{al}}$ highlight the advantages of the robust approach for the desired Γ-value, we also want to investigate how sensitive BB-Γ_{al} behaves on the subsequent adjustment of the evaluation parameter Γ_{ev}. For that reason, we evaluate the final configurations of all algorithms BB and BB-Γ_{al} for $\Gamma_{\mathrm{al}} \in \{1, \dots, 5\}$ with $\overline{\mathrm{BI}}^{\Gamma_{\mathrm{ev}}}_{\mathrm{rob}}$ and $\mathrm{BI}^{\Gamma_{\mathrm{ev}}}_{\mathrm{rob}}$ for $\Gamma_{\mathrm{ev}} \in \{1, \dots, 5\}$. Again the differences between the results of BB and BB-Γ_{al} as defined in equations (3.52)–(3.54) are calculated, which results in a 5×5-matrix of data points that is presented in Figures 3.15a–3.15d. Figures 3.15a–3.15b state average values of all 960 instances grouped by Γ_{ev} on the horizontal axis whereas the vertical axis indicates $\Delta\overline{\mathrm{BI}}^{\Gamma_{\mathrm{ev}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$ and $\Delta\mathrm{BI}^{\Gamma_{\mathrm{ev}},\Gamma_{\mathrm{al}}}_{\mathrm{rob}}$, respectively. Similarly, Figures 3.15c–3.15d state average values of 142 instances where (3.55) holds. Clearly, the charts show that BB-Γ_{al} behaves better for the intended parameter $\Gamma_{\mathrm{ev}} = \Gamma_{\mathrm{al}}$ than for other Γ_{ev}-values, i.e., the highest value for a fixed Γ_{ev} and among all BB-Γ_{al} can be found for the BB with $\Gamma_{\mathrm{al}} = \Gamma_{\mathrm{ev}}$. Furthermore, the trend line of BB-Γ_{al} decreases for increasing $\Gamma_{\mathrm{ev}} > \Gamma_{\mathrm{al}}$, whereas any other BB-$\Gamma'_{\mathrm{al}}$ with $\Gamma_{\mathrm{al}} < \Gamma'_{\mathrm{al}}$ reports better results. This was expected since the algorithm just does not regard violations according to higher values of Γ_{ev}. However, Figures 3.15a–3.15b also display slightly better results for BB-Γ_{al} compared to BB-Γ'_{al} with $\Gamma_{al} < \Gamma'_{al}$ for $\Gamma_{ev} \leqslant \Gamma_{al}$. This effect is even more visible in Figures 3.15c–3.15d since for one instance different optimal values $\overline{\mathrm{BI}}^{\Gamma}_{\mathrm{rob}} < \overline{\mathrm{BI}}^{\Gamma+1}_{\mathrm{rob}}$ may exist (computed by the MIP presented in Section 3.4.1) for different values of Γ, i.e., while all robust configurations display a gain in robustness in comparison to the nominal approach, this gain is sensitive to the choice of Γ_{al}, and the parameter should be chosen carefully to reflect Γ_{ev} as closely as possible.

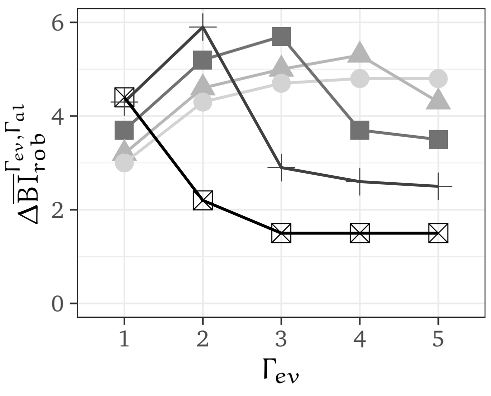


(a) Comparison on $\Delta\overline{\mathrm{BI}}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$ including all instances.

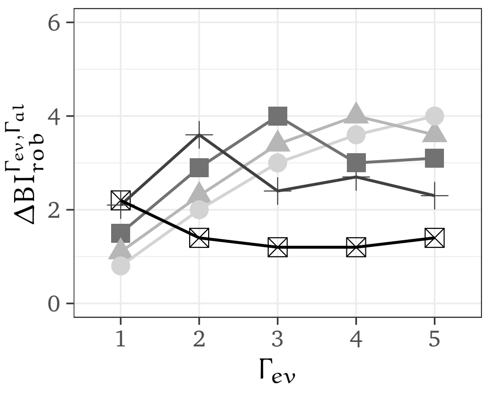


(b) Comparison on $\Delta\mathrm{BI}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$ including all instances.





(c) Comparison on $\Delta\overline{\mathrm{BI}}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$ including only instances which satisfy (3.55).



(d) Comparison on $\Delta\mathrm{BI}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$ including only instances which satisfy (3.55).

Figure 3.15: Comparing results of BB-Γ_{al} for $\Gamma_{ev} \in \{1, \ldots, 5\}$ on instances from the Tanaka data set.

Results for Forster data set

Secondarily, we present results of the Forster data set which include much larger instances. For the Forster data set, we solved all instances with up to $n = 64$ items. Since the data set contains very different instance groups, we include a break in the trend lines of Figure 3.16 and 3.17a–3.17c dividing the results in instance groups with a smaller number of priority classes ($N \in \{10, 12, 13\}$) on the left side of the charts and instance groups with a larger number of priority classes ($N \in \{20, 24, 26\}$) on the right.

Figure 3.18a and 3.18b present results for a smaller, while Figure 3.18c and 3.18d present results for a larger number of priority classes. Since not all instances could be solved in

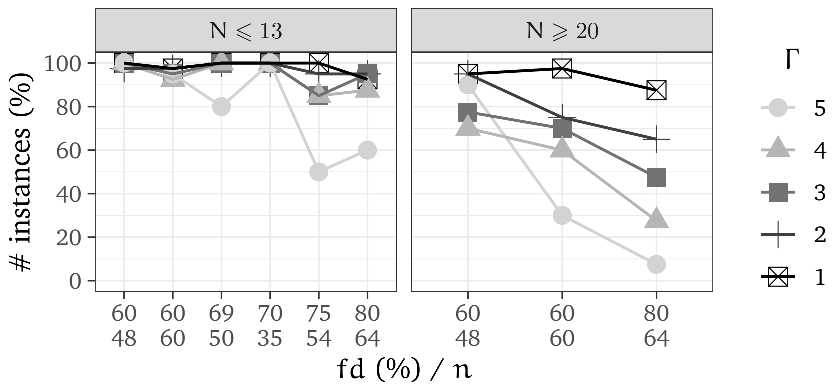
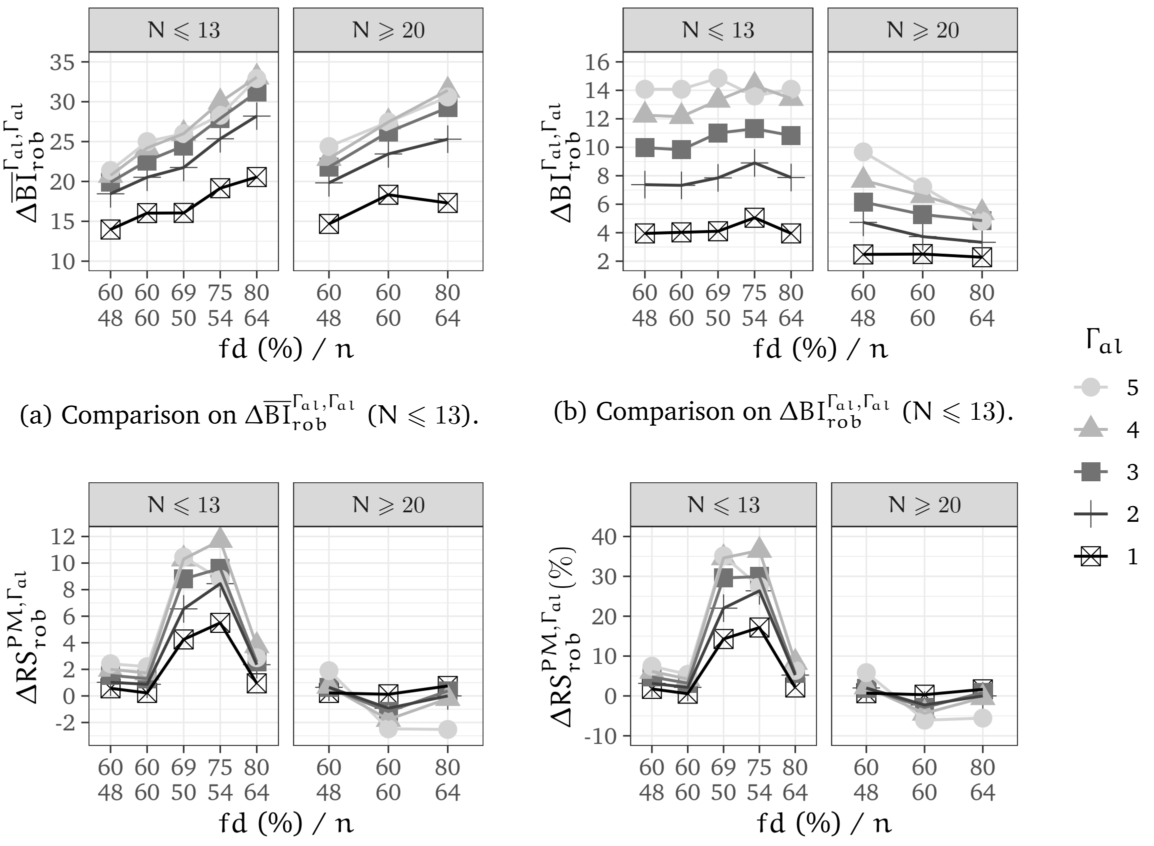


Figure 3.16: Number of instances verified to be optimally solved (%) for the Forster data set.



(a) Comparison on $\Delta\overline{\mathrm{BI}}^{\Gamma_{al},\Gamma_{al}}_{rob}$ ($N \leqslant 13$).

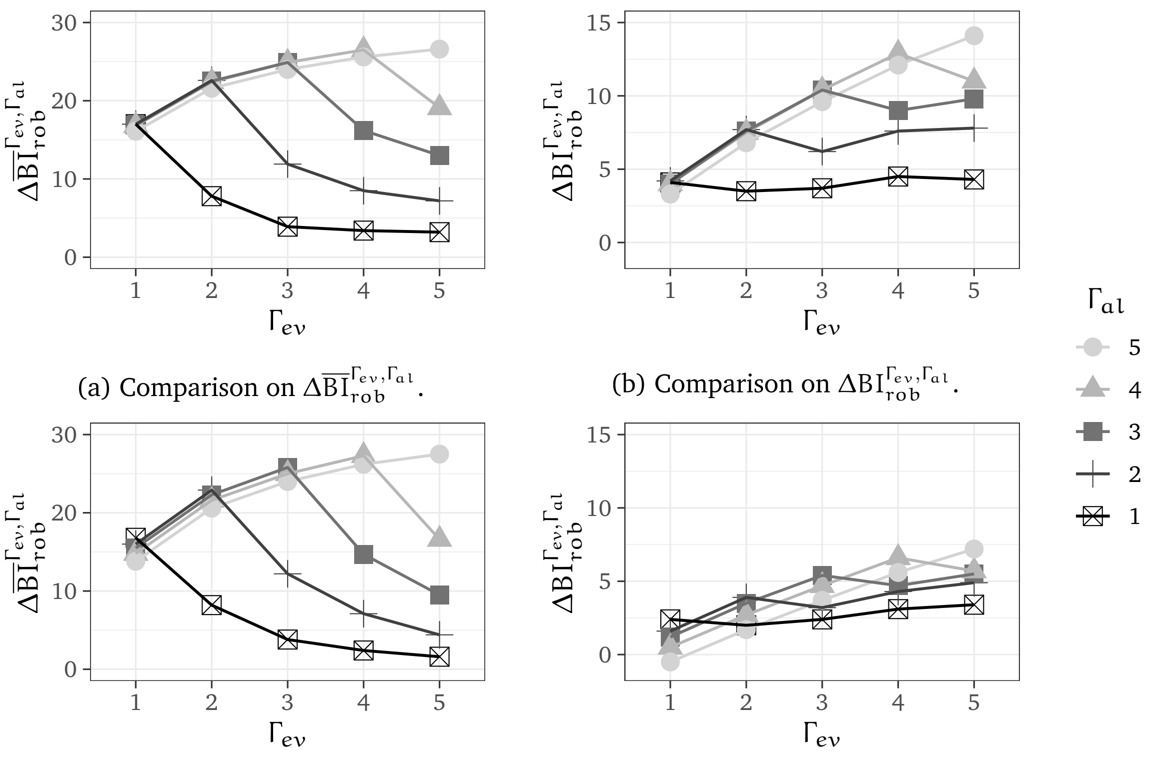
(b) Comparison on $\Delta\mathrm{BI}^{\Gamma_{al},\Gamma_{al}}_{rob}$ ($N \leqslant 13$).

(c) Comparison on $\Delta\mathrm{RS}^{PM,\Gamma_{al}}$ ($N \geqslant 20$).

(d) Comparison on $\Delta\mathrm{RS}^{PM,\Gamma_{al}}$ (%) ($N \geqslant 20$).

Figure 3.17: Results of BB-Γ_{al} for different n, b, fd on instances from the Forster data set.

the given time limit, Figure 3.16 state the percentage number of instances verified to be optimally solved. Figure 3.17a–3.17d are similarly organized as Figure 3.14a–3.14d related to the Tanaka data set. Contrary to the Tanaka data set, all instances have the optimal value $\mathrm{BI}^{\Gamma}_{rob} = 0$ for all $\Gamma \in \{1, \dots, 5\}$. However, the results appear qualitatively similar compared to the results of the Tanaka data set when taking into account that not all instances could be optimally solved in the given time limit.



(a) Comparison on $\Delta\overline{\mathrm{BI}}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$.

(b) Comparison on $\Delta\mathrm{BI}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$.

(c) Comparison on $\Delta\overline{\mathrm{BI}}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$.

(d) Comparison on $\Delta\mathrm{BI}_{\mathrm{rob}}^{\Gamma_{ev},\Gamma_{al}}$.

Figure 3.18: Results of BB-Γ_{al} for $\Gamma_{ev} \in \{1, \ldots, 5\}$ on instances from the Forster data set.

Results for Caserta data set

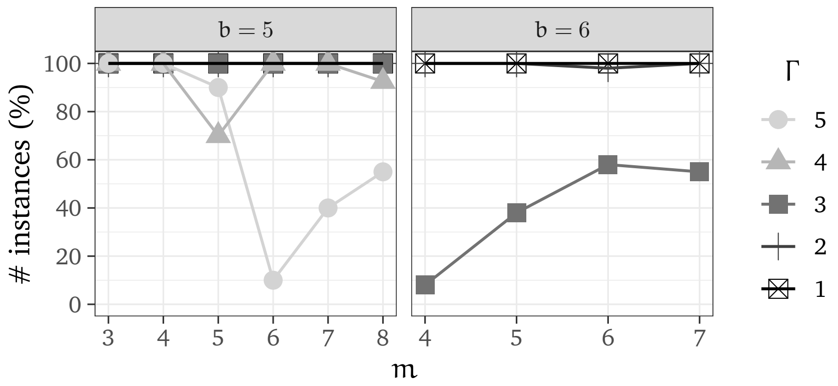
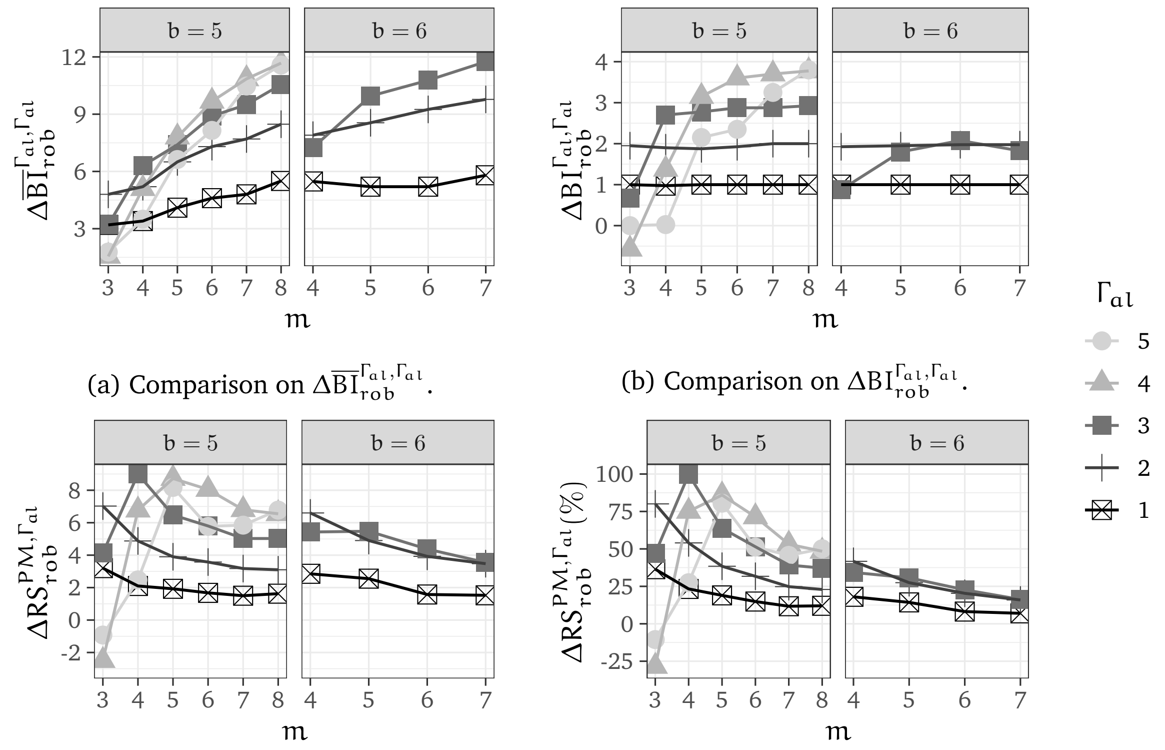


Figure 3.19: Number of instances verified to be optimally solved (%) for the Caserta data set.

Finally, we state the results for the Caserta data set where all stacks are filled up to level $b - 2$ and have only unique priority classes. Moreover, the instances of the Hottung data set have the same structure as the Caserta instances. Since the smallest instances have a stack height of $b = 7$ and we decided to include only instances with $b \leqslant 6$, we did not use these instances for the third experiment.

For the Caserta data set, we solved all smaller instances with stack heights $b \in \{5, 6\}$. We present the results together by separating $b = 5$ and $b = 6$ with a break in the trend lines similarly to the Forster results. Figures 3.19 and 3.20a–3.20d state the results for $b = 5$ and



(a) Comparison on $\Delta\overline{BI}_{rob}^{\Gamma_{al},\Gamma_{al}}$.

(b) Comparison on $\Delta BI_{rob}^{\Gamma_{al},\Gamma_{al}}$.

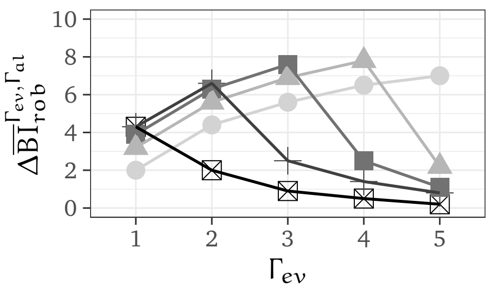
(c) Comparison on absolute $\Delta RS^{PM,\Gamma_{al}}$.

(d) Comparison on percentage $\Delta RS^{PM,\Gamma_{al}}$.

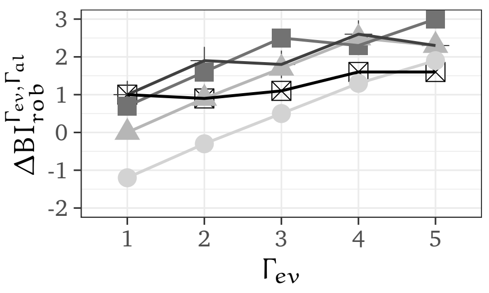
Figure 3.20: Results of BB-Γ_{al} for different m and instances from the Caserta data set.

$\Gamma_{al} \in \{1,\dots,5\}$ on the left side and for $b = 6$ and $\Gamma_{al} \in \{1,\dots,3\}$ on the right side of the charts. These figures are similarly presented as the results of the Tanaka data set, but instead of grouping by b and fd, the number of stacks m is varied. However, the results are quite comparable when taking into account that not all instances could be solved to optimality and all instances for $\Gamma_{al} = 4$ and $m \leqslant 4$ respectively $\Gamma_{al} = 5$ and $m \leqslant 5$ fulfill $\overline{BI}_{rob}^{\Gamma} > 0$ which reduces the potential for improvements. As with the results for the Tanaka data set, all mentioned instances that fulfill $\overline{BI}_{rob}^{\Gamma} > 0$ may not be solved to optimality in terms of RS^{PM}.

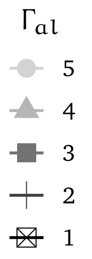
Figure 3.19 displays the number of instances verified to be optimally solved in terms of $\overline{BI}_{rob}$. Moreover, it must be noted that all instances of the Caserta data set have unique priority classes which restricts the improvement to $\Delta BI_{rob}^{\Gamma_{al},\Gamma_{al}} \leqslant \Gamma_{al}$. The results of the sensitivity analysis in Figures 3.21a–3.21d for $b = 5$ and in Figures 3.22a–3.22d for $b = 6$ are similarly presented as the results of the Tanaka data set. Altogether, the described results of this section are qualitatively very similar (with slightly larger gaps between the trend lines) to the results of the Tanaka data set when considering that some instances fulfill $\overline{BI}_{rob}^{\Gamma} > 0$ and not all instances could be optimally solved.

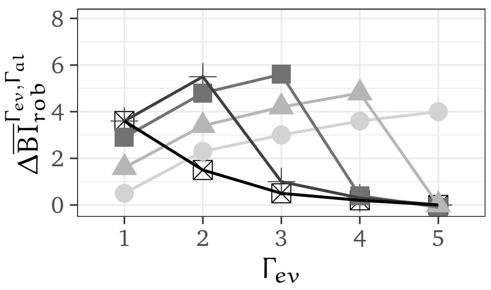


(a) Comparison on $\Delta\overline{BI}_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including all instances.

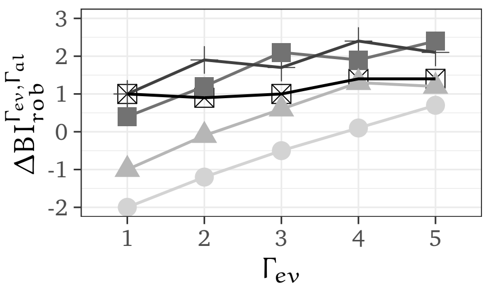


(b) Comparison on $\Delta BI_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including all instances.



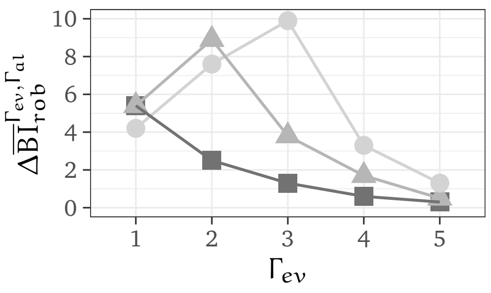


(c) Comparison on $\Delta\overline{BI}_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including instances which satisfy (3.55).

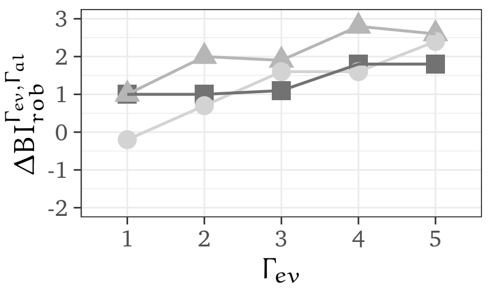


(d) Comparison on $\Delta BI_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including instances which satisfy (3.55).

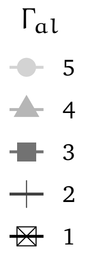
Figure 3.21: Comparing results of BB-Γ_{al} for $\Gamma_{ev} \in \{1, \ldots, 5\}$ on instances with stack height $b = 5$ from the Caserta data set.

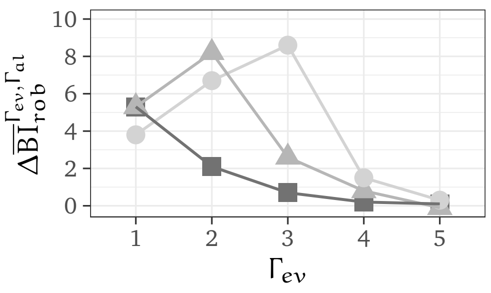


(a) Comparison on $\Delta\overline{BI}_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including all instances.

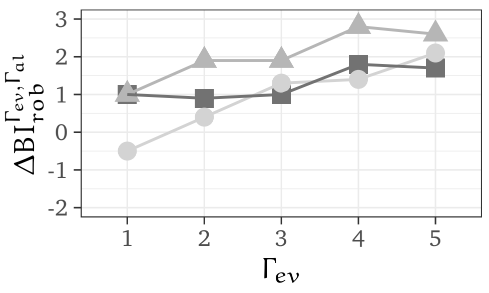


(b) Comparison on $\Delta BI_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including all instances.





(c) Comparison on $\Delta\overline{BI}_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including instances which satisfy (3.55).



(d) Comparison on $\Delta BI_{rob}^{\Gamma_{ev},\Gamma_{al}}$ including instances which satisfy (3.55).

Figure 3.22: Comparing results of BB-Γ_{al} for $\Gamma_{ev} \in \{1, \ldots, 5\}$ on instances with stack height $b = 6$ from the Caserta data set.

3.6 Conclusions

In this chapter, we considered a premarshalling problem under uncertainty, where we assume that the order of retrievals can be different than expected. By choosing a parameter Γ, the decision maker can determine the degree of robustness, which corresponds to the number of adjacent swaps that are expected in the retrieval sequence.

We provided a theoretical analysis with regards to existence of robust configurations and problem complexity, and provided (mixed) integer programming models to find and evaluate robust configurations. We performed three computational experiments using four benchmark sets from the literature. In the first experiment, we compared solution methods for the master and subproblems. The second experiment highlighted the value of the theoretical analysis, which makes it possible to find optimal robustness objective values without using the relatively time-consuming iterative solution procedure. In our third experiment we discussed the impact of the number and position of empty priority classes. Finally, the fourth experiment considered the tradeoff between additional reshuffles during premarshalling for reaching a robust configuration and the reduced number of reshuffles that result during unloading. We found that few reshuffles during premarshalling (which is usually done overnight, where time is not critical) can save a larger number of reshuffles during unloading (where time is most important).

In further research, one could reduce the sensitivity of robust configurations to the parameter Γ by considering globalized robust counterparts (cf., e.g., Ben-Tal et al. [5]) or mixed uncertainty approaches (cf., e.g., Dokka et al. [29]), where different uncertainty sets are taken into account simultaneously.

Chapter 4

The blocks relocation problem with item families minimizing the number of reshuffles

In this chapter, we take the *blocks relocation problem with item families* (BRPIF) into consideration and investigate it theoretically as well as examine suitable solving strategies. This chapter is based on Boge and Knust [10] and is organized as follows. At first, we describe the problem setting more precisely in Section 4.2, while we prove some new complexity results in Section 4.3. Section 4.4 is devoted to solution approaches, including integer programming formulations, a simple heuristic, and a two-stage simulated annealing algorithm. In Section 4.5 we report results of a computational study. Finally, some concluding remarks can be found in Section 4.6.

4.1 Introduction

Often items must be stored in a storage area of limited capacity, e.g., in container terminals, container ships, storage yards, warehouses, steel plants or tram depots (cf. the survey in Lehnfeld and Knust [73]). Such storage systems are organized in stacks of limited height which are filled from bottom to top and unloaded in reverse order. The items arrive in a *loading stage* and have to be stored at appropriate locations in the stacks until they are demanded for further transport or processing. In container terminals, usually every item is unique since contents of containers correspond to individual shipments, which implies that containers cannot be exchanged. On the other hand, in warehouses where commodities like wood or steel plates are stored, the items are exchangeable to a certain degree. Another example for storage areas are tram depots where trams stay during the night. Each rail siding may be interpreted as a (horizontal) stack with only one exit. In all these settings, certain data are associated with each item (e.g., its weight, size, destination, expected retrieval time). On the one hand, this data plays a role for different stacking policies (cf. Dekker et al. [27]) and on the other hand, it determines the *family*, i.e., the main type that is used to distinguish different items.

After the loading process is completed, all items stay in the storage until items belonging to specific families are demanded in the *unloading stage*. In this stage, a sequence of demanded families is given and at first appropriate items must be selected, i.e., for each demanded family an individual item must be chosen in the storage. Second, the selected items have to be retrieved according to the given sequence for further processing or transportation. Since only the topmost item in each stack can be directly accessed, often *reshuffles* (*relocations*) of other so-called *blocking items* are necessary before the next demanded item can be retrieved. Such unproductive movements should be avoided as much as possible to ensure efficient operation of the storage. Hence, an important indicator for a selection of items in the storage is the total number of blocking items (which is a lower bound on the total number of reshuffles). However, usually blocking items cannot be completely avoided since often the exact retrieval sequence is not known during the loading stage.

We encountered the described setting in a German company that maintains a warehouse of wooden boards stored in stacks. These boards have specific attributes (e.g., color, length, height, etc.) and families are defined based on these attributes (e.g., a family consists of boards with the same color and the same size). All boards are stacked in a storage area of limited size. An automatic crane picks up the boards and relocates them between the stacks or transports them to the exit of the storage where a stationary saw cuts the boards into smaller pieces for further processing. Based on the demands for pieces with certain attributes and a saw schedule, a retrieval sequence demanding certain families is specified a day or a week in advance. Since the automatic crane of the company is linked with an automatically updated storage database, all movements and item positions can always be tracked and no restrictions on the movements are imposed (in contrast, in the literature often so-called "push-backs" are considered, see the discussion below). The objective is to select appropriate items for the demanded families and to retrieve them from the storage minimizing the total number of reshuffles.

In the literature, related settings were considered as *slab stack shuffling* (SSS) (cf. Fernandes et al. [38], Li et al. [75], Ren and Tang [88], Singh et al. [95], Tang and Ren [103], and Tang et al. [101, 102]) in the context of steel production and as *scheduling trams in the morning* (STIM) or *minimum shunting problem at departure* (MSDP) in the context of tram scheduling (cf. Blasum et al. [7] and Winter and Zimmermann [116]). All these publications have in common that the items are assigned to specific families, all items are already in the storage, (i.e., the loading stage is completed before) and the objective aims at optimizing the retrieval effort. On the other hand, they differ in restrictions on the movements and the considered objective function (e.g., counting only the number of blocking items or more accurately the number of actual reshuffles). The families are often restricted to be disjoint sets (cf. Ren and Tang [88], Singh et al. [95], Tang et al. [102], Blasum et al. [7], and Winter and Zimmermann [116]), but are sometimes also introduced without any restrictions (cf. Fernandes et al. [38], Tang and Ren [103], and Tang et al. [101]).

In the context of the steel industry, Tang et al. [101] tackled the SSS problem where steel plates of different types must be selected in a certain retrieval sequence. Here it is assumed that every blocking item which is relocated to another stack must immediately be pushed back to its initial stack after the retrieval of the current target item. The authors developed a greedy heuristic in combination with a neighborhood search and showed improvements in comparison to a greedy heuristic used by a company on real-world and randomly generated test data. Tang et al. [102] worked on the same problem and proposed an integer programming formulation (but without solving it) as well as a *genetic algorithm* (GA). They compared their heuristic to the greedy heuristic on randomly generated test data. Singh et al. [95] presented the same *integer program* (IP) and another GA for the SSS problem and claimed an improvement over the previous GA results. Fernandes et al. [38] made further progress on the SSS problem by developing a quadratic integer programming formulation that was linearized and solved with an IP solver. They compared it with a heuristic used in practice on randomly generated instances. Moreover, Tang and Ren [103] considered the SSS problem without the assumption that items must be pushed back. Instead they included the assumption that items can be moved to "nearby" target stacks where these items stay until they are finally taken out of the storage. Furthermore, other constraints regarding deadlines for the items in the storage and a simplified model for the crane working time are considered. An adapted variant of the IP of Tang et al. [102] and a dynamic programming approach are stated and randomly generated instances of large size are solved using a partitioning strategy. Ren and Tang [88] treat the SSS problem as a subproblem of a superior crane scheduling problem. They developed an IP formulation for the SSS subproblem and a heuristic for the crane scheduling using the heuristic in an iterative process to add cuts to the IP until a feasible solution is found. Their algorithm was tested on ten randomly generated instances.

A similar setting can also be found in train depots with dead-end sidings which may be interpreted as stacks. Blasum et al. [7] considered a problem tackling the shunting of trams in such train depots. Each tram belongs to a specific type and all trams leave the depot in a certain sequence of types to serve the daily schedule. Although it is not explicitly stated, the shunting moves seem to be push-back moves as explained above. Only shunting-free schedules (schedules with no shunting moves) are considered and $\mathcal{NP}$-completeness of the STIM problem was shown. Furthermore a dynamic program was developed to solve several real-world and randomly generated instances. Winter and Zimmermann [116] continued working on the mentioned problem, also considering an optimization variant minimizing the number of shuntings instead of searching for a shunting-free schedule. They developed a quadratic IP formulation, a branch-and-bound algorithm, and heuristics.

In this chapter, we consider the basic problem for the unloading stage with item families where a family retrieval sequence is given. We assume that the space in the storage area is limited and all movements of items have to be carried out within this limited area (i.e., no temporary storage exists). Moreover, we do not consider simplifying assumptions (like push-backs) and concentrate on the more realistic, but also more difficult objective "total number of reshuffles" which according to Fernandes et al. [38] "increases considerably the problem complexity and should be carefully addressed".

Recall the well-known *blocks relocation problem* (BRP) (cf., e.g., Caserta et al. [19] and Forster and Bortfeldt [39]) which we generalize to the BRPIF. The BRP is an important and well-studied problem in the literature for which various heuristics and exact algorithms have been developed. It can be seen as a special case of the BRPIF where all items belong to different families (i.e., each family contains only a single item) and the retrieval sequence includes all items that are in the storage.

We propose a two-stage *simulated annealing* (SA) algorithm for the BRPIF which selects appropriate items for the demanded families in the first (selection) stage and evaluates the chosen selection by applying a fast and effective BRP heuristic in the second (retrieval) stage. We present IP formulations for the objectives (i) total number of reshuffles, and (ii) total number of blocking items where the second is used to compute lower bounds. Furthermore, we analyze the complexity of the problem in different versions. In some computational experiments based on real-world data from a company, benchmark instances from the literature, and randomly generated instances with different characteristics reflecting real-world settings, we evaluate both the IP formulations and the SA approach with respect to solution quality and computation times.

Additionally, we address a new variant of the problem where the family retrieval sequence may be relaxed. In a complete relaxation this means that a multiset of families is given and again appropriate items have to be selected for the demanded families. However, their retrieval order is not fixed and may also be optimized. On the one hand, this relaxation may be useful for lower bound calculations. On the other hand, the company mentioned above imposed the question how much they can gain w.r.t. the total retrieval time if their subsequent sawing stage can be organized in a more flexible way. In another version of the problem, not the whole retrieval sequence may be relaxed, but some subsequences are flexible. We model this by a sequence of multisets where the order of the multisets has to be respected, but inside each multiset the retrieval order is flexible.

4.2 Problem formulation

In this section, we give a formal description of the considered problem and introduce the used notations. We are given a storage area which consists of m stacks $\mathcal{Q} = \{1, \dots, m\}$, each stack contains b levels $\mathcal{B} = \{1, \dots, b\}$, and at every level exactly one item can be stored (i.e., at most b items can be placed in each stack). In a generalized version of the problem, the stacks $q \in \mathcal{Q}$ may have different stack heights b_q. In the stacks, n items $\mathcal{I} = \{1, \dots, n\}$ are stored. Each item $i \in \mathcal{I}$ belongs to a certain family $f_i \in \mathcal{F} = \{1, \dots, F\}$. We denote by $\mathcal{I}_f$ the set of all items belonging to family f and assume that all families are disjoint, i.e., $\mathcal{I}_f \cap \mathcal{I}_{f'} = \emptyset$ for $f \neq f'$.

Furthermore, a "*family retrieval sequence*" $\Phi = (\phi_1, \dots, \phi_L)$ containing $L \leqslant n$ families $\phi_k \in \mathcal{F}$ is given. This sequence determines which families are demanded for retrieval meaning that at position k of the sequence an item from family ϕ_k must be selected. Note that $L < n$ is possible, i.e., in contrast to the classical BRP not all items have to be unloaded. We assume that in the storage for each demanded family enough items belonging to these families are available and denote by $\mathcal{L} := \{1, \dots, L\}$ the set of all indices in Φ. Furthermore, let $\mathcal{L}_i := \{k \in \mathcal{L} \mid \phi_k = f_i\}$ be the set of all indices k of the family retrieval sequence where the demanded family ϕ_k equals the family of item i (i.e., i can be assigned to index k).

A selection of appropriate items may be represented by a mapping from each family retrieval sequence index k to exactly one item i with family $f_i = \phi_k$. This leads to a sequence of selected items $S = (i_1, \dots, i_L)$ where item i_k at index k corresponds to family ϕ_k of the family retrieval sequence Φ. We call such a sequence S an *item retrieval sequence* and denote the set of selected items by $\mathcal{I}_{sel}$. After selecting appropriate items, these items have to be retrieved according to the order of S in the unloading stage. Most times it is necessary to relocate items between different stacks since other items are blocking the selected items. Thus, beneath an item retrieval sequence S, a solution also involves a *sequence of relocations* that describes which items are relocated in which order. A single relocation is described by a (non-empty) departure stack and a (non-full) destination stack. Since relocations are time consuming, they should be avoided as much as possible.

In the remainder of this chapter we propose a two-stage approach where in the first stage appropriate items are selected (*selection stage*) which are unloaded in the second stage (*retrieval stage*). In this context, the retrieval problem can be considered as BRP as follows. In the BRP a storage of fixed dimensions (m stacks of height b) filled with n items is given and all items must be retrieved from the storage. The items have specific priorities that determine the retrieval order. For two items i, j with priorities $p_i < p_j$, item i must leave the storage before item j. In our problem setting, the order of the items in the item retrieval sequence induces priority values for all items $i \in \mathcal{I}_{sel}$. If item i is assigned to index $k \in \mathcal{L}$, then we set the priority value of item i to $p_i := k$. Furthermore, all remaining non-selected items $i \in \mathcal{I} \setminus \mathcal{I}_{sel}$ get the artificial priority value $p_i := L + 1$ indicating that these items remain in the storage after the selected items have been retrieved. Each solution of such a BRP instance can be transformed into a feasible solution of our retrieval problem by ignoring the retrieval of all items with priority $L + 1$.

Example 4.1. *Figure 4.1a shows an example with $m = 3$ stacks of height $b = 3$ and $n = 9$ items belonging to families A, B, C, D. We have to select appropriate items for the family retrieval sequence $\Phi = (A, B, C, A)$ of length $L = 4$. In Figure 4.1b, four items are selected and assigned to the priority values $1, 2, 3, 4$. All remaining items get the artificial priority value $L + 1 = 5$. In the corresponding BRP instance two blocking items (with priorities $3, 4$, underlined) exist which must be relocated in any case during the unloading process. Figure 4.1c shows another selection with no blocking items. Since here no relocations are necessary, this selection is optimal.*

Considering the problem of the retrieval stage as BRP allows to reuse measures and solution algorithms developed for the BRP. As described above, the objective is to retrieve all items from the storage using a minimum *total number of reshuffles*, which is $\mathcal{NP}$-hard (cf. Caserta et al. [19]). To simplify the problem, we may also evaluate a fixed selection of items by lower

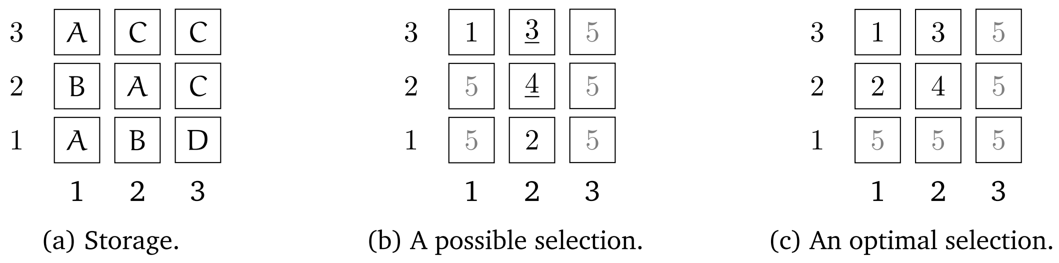


Figure 4.1: Family retrieval sequence $\Phi = (A, B, C, A)$.

bounds. For this purpose, we count the *total number of badly placed items* (BI). An item i is called *badly placed* (cf. Forster and Bortfeldt [39]) if it is blocking an item placed somewhere (not necessarily directly) below in the same stack which has to be retrieved earlier (i.e., there is an item j placed below with priority $p_j < p_i$). We call an item *well placed* if it is not badly placed. Obviously, the value BI defines a lower bound on the total number of reshuffles since each badly placed item has to be reshuffled at least once. However, as shown in Blasum et al. [7], finding a solution for the BRPIF with decision value $BI = 0$ is already strongly $\mathcal{NP}$-complete.

In the literature, the objective BI is considered if relocations do not play any role (cf. Li et al. [75], Ren and Tang [88], and Tang and Ren [103]). This situation occurs in practice if a temporary storage area exists where blocking items can be stored. This implies that each blocking item must only be reshuffled once before its retrieval, i.e., the total number of reshuffles equals the number of badly placed items. Sometimes, also another constraint is considered (cf. Blasum et al. [7], Tang et al. [101, 102], Singh et al. [95], and Winter and Zimmermann [116]). There it is assumed that each relocation must be reverted immediately after retrieving the next item (push-back). This situation occurs for example in practice if in the database no dynamically changing storage positions of items can be handled (i.e., each item is fixed to one specific stack).

Example 4.2. *We highlight the difference between the objective functions* BI *and* RS *in Figure 4.2 by considering a storage area with* $n = 9$ *items in* $m = 3$ *stacks of height* $b = 5$ *where the items have to be retrieved in the order* $1, \dots, 9$. *For this storage we have* $BI = 4$ *(items underlined). On the other hand,* $RS = 5$ *since one of the badly placed items must be relocated twice. To achieve a solution with 5 reshuffles, we retrieve item 1, move item 7 to stack 2, retrieve items 2 and 3, move item 8 to stack 1, retrieve item 4, move item 7 to stack 1 (second relocation of item 7), move item 6 to stack 1, move item 9 to stack 3, and then retrieve items 5,6,7,8,9 without further reshuffles.*

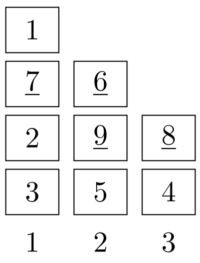


Figure 4.2: Example with $BI = 4$, but $RS = 5$.

Note that in the two-stage approach as described above, all selected items get different priority values $1, \dots, L$. However, in the literature sometimes also a more general BRP variant is studied where different items may have the same priority (cf., for example Forster and

Bortfeldt [39] and Jin et al. [57]), also called BRP with “duplicate priorities”. In this situation, all items with priority value p have to be retrieved (in an arbitrary order) before all items of priority value $p + 1$, and so on. We will also use this variant in our two-stage approach to reduce the search space in the first stage (see Section 4.4.3). Furthermore, note that the BRP with duplicate priorities is also a special case of the BRPIF where $L = n$ (i.e., all items have to be retrieved) and in the family retrieval sequence at first all items with priority 1 have to be retrieved, then all items with priority 2, and so on.

In the literature, the BRP appears in two different variants (cf., e.g., Caserta et al. [19]): in the *restricted BRP*, only blocking items above the next item to be retrieved may be relocated (so-called “forced moves”), in the *unrestricted BRP*, also so-called “voluntary moves” are allowed which means that any item may be relocated at any time. Often, only the restricted variant of the BRP is used since then the solution space is much smaller. However, since in the unrestricted variant in general solutions with fewer moves are possible and our solution method is able to deal with it, we solve the unrestricted BRP in the second stage.

4.3 Complexity results

In this section, we study the complexity of the BRPIF in different variants. As described above, in the literature often either push-backs are considered or it is assumed that sufficient storage locations for relocated items exist (i.e., blocking items have to be moved only once and can stay at temporary positions until their retrieval). On the other hand, if the storage area is limited and all relocations must be carried out inside this limited area, the reshuffling process is much more complicated. Then, it may even happen that the problem is infeasible since not enough empty storage locations exist. At first we deal with this feasibility issue.

Lemma 4.1. *For the BRPIF, it can be decided in $\mathcal{O}(n)$ whether for a given family retrieval sequence a selection of items exists in such a way that these items can be retrieved using only relocations within the limited storage area.*

Proof. We generalize the feasibility test of Zhu et al. [124] for the BRP to the BRPIF. In Zhu et al. [124] the BRP is considered where all items $i \in \mathcal{I}$ have pairwise different priorities. Assuming that the items have to be retrieved in the sequence $1, \dots, n$ and l_i denotes the level at which item i is stored, we must have

$$b - l_i \leqslant mb - n + (i - 1) \tag{4.1}$$

to ensure that all reshuffles can be carried out inside the limited storage area. In inequality (4.1), the term $mb - n$ equals the initial number of free slots in the storage. When item i is to be retrieved, $(i-1)$ additional slots are free, since $i-1$ items have already been retrieved before. Thus, item i must have at least the level b minus the current number of free slots.

If we adapt this to the context of the BRPIF, for index k of the family retrieval sequence Φ we must select an appropriate item i_k with $f_{i_k} = \phi_k$ that has not been selected before and that is located at a level l_{i_k} satisfying

$$b - l_{i_k} \leqslant mb - n + (k - 1). \tag{4.2}$$

To check this condition for the relevant indices k (we can stop as soon as $b - 1$ free slots in the storage exist, since then all remaining items in the storage are accessible), we iterate over all n items in the storage and create a list for each demanded family in which the levels of all its items are stored in descending order. Afterwards, we check inequality (4.2) for each relevant index k using the list of family ϕ_k. □

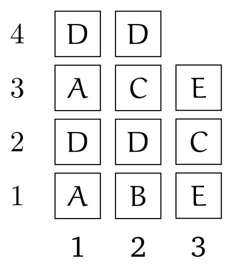


Figure 4.3: Feasibility check.

Example 4.3. *Figure 4.3 shows a storage with $b = 4$, $m = 3$, $n = 11$, and families A, B, C, D, E. If we want to check feasibility for the family retrieval sequence $\Phi = (A, A, B, C, D)$, we create four lists for the demanded families:*

$$A : (3, 1), \quad B : (1), \quad C : (3, 2), \quad D : (4, 4, 2, 2)$$

Checking inequality (4.2) for index $k = 1$ leads to $4 - l_{i_1} \leqslant 3 \cdot 4 - 11 + (1 - 1)$, i.e., an item i_1 belonging to the corresponding family A must be stored at level $l_{i_1} \geqslant 3$. Since the first entry in the list of family A satisfies this, we proceed with $k = 2$ resulting in the condition $l_{i_2} \geqslant 2$. However, since the second entry in the list of family A is equal to 1, we state infeasibility.

On the other hand, if we consider the retrieval sequence $\Phi' = (A, D, A, B, C)$, we have a feasible solution since the first entry in the list of family D satisfies $l_{i_2} \geqslant 2$ and afterwards all items are accessible.

In the following, we consider the objective functions BI and RS. Blasum et al. [7] showed that for the BRPIF with $b = 3$ it is strongly $\mathcal{NP}$-complete to decide whether a solution with $BI = 0$ (and hence also with $RS = 0$) exists. We strengthen their result by showing that the problem is already hard for $b = 2$.

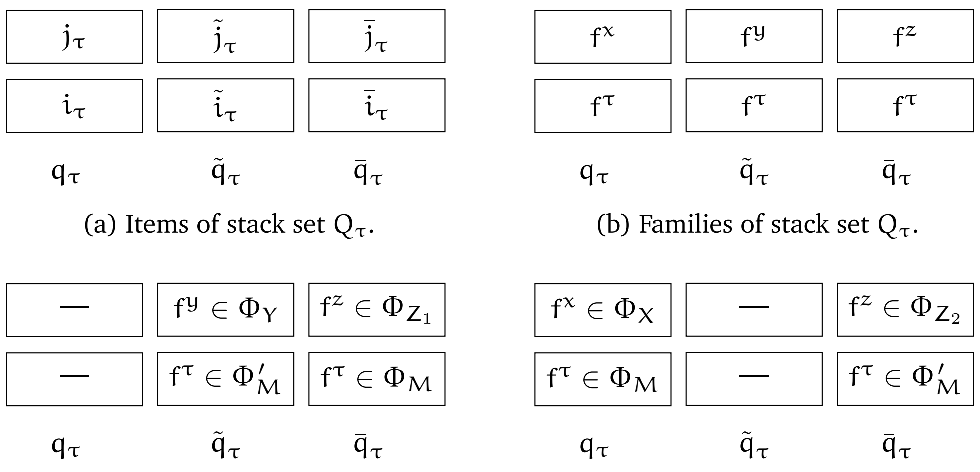
Theorem 4.1. *For the BRPIF with $b = 2$ it is strongly $\mathcal{NP}$-complete to decide whether a solution without any badly placed items exists.*

Proof. Similar to the proof of Blasum et al. [7] for the case $b = 3$, we give a reduction from *3-dimensional matching* (3DM).

3DM: Given are three sets X, Y, Z such that $|X| = |Y| = |Z| = a$ and a subset $M \subset X \times Y \times Z$ with $|M| \geqslant a$. We want to decide if a subset $U \subset M$ with $|U| = a$ exists such that for any two triples $(x, y, z), (x', y', z') \in U$ the conditions $x \neq x'$, $y \neq y'$, and $z \neq z'$ hold.

Given a 3DM instance, we construct an instance of the BRPIF by identifying each element $x \in X, y \in Y, z \in Z$ with one family f^x, f^y, f^z, resulting in $3a$ different families. Let $\mathcal{F}_X, \mathcal{F}_Y, \mathcal{F}_Z$ be the corresponding sets of families. Furthermore, we identify each triple $\tau = (x, y, z) \in M$ with a “triple family” $f^\tau \in \mathcal{F}_M$. For each triple $\tau \in M$ in 3DM, we create three stacks $q_\tau, \tilde{q}_\tau, \bar{q}_\tau$ containing items $i_\tau, j_\tau, \tilde{i}_\tau, \tilde{j}_\tau, \bar{i}_\tau, \bar{j}_\tau$ resulting in $m = 3|M|$ stacks with $b = 2$. We call these three related stacks a “stack set” and denote it by Q_τ. Items $i_\tau, \tilde{i}_\tau, \bar{i}_\tau$ are placed at the bottom level, items $j_\tau, \tilde{j}_\tau, \bar{j}_\tau$ at the top level (cf. Figure 4.4a). For $\tau = (x, y, z) \in M$, the families are set to $f_{j_\tau} = f^x, f_{\tilde{j}_\tau} = f^y, f_{\bar{j}_\tau} = f^z$, and $f_{i_\tau} = f_{\tilde{i}_\tau} = f_{\bar{i}_\tau} = f^\tau$ (cf. Figure 4.4b).

For each element $x \in X$ and $z \in Z$ let n_x, n_z be the numbers of triples in M that include x and z, respectively. The family retrieval sequence Φ is set to $(\Phi_{Z_1}, \Phi_X, \Phi_M, \Phi_Y, \Phi_{Z_2}, \Phi'_M)$ where $\Phi_{Z_1} = (f^{z_1}, \ldots, f^{z_a})$ and $\Phi_Y = (f^{y_1}, \ldots, f^{y_a})$ include all a families from $\mathcal{F}_Z$ and $\mathcal{F}_Y$ once. The subsequences $\Phi_M = \Phi'_M = (f^{\tau_1}, \ldots, f^{\tau_{|M|}})$ contain all $|M|$ families associated with the triples $\tau \in M$. The subsequences $\Phi_X = (f^{x_1}, \ldots, f^{x_1} | \ldots | f^{x_a}, \ldots, f^{x_a})$ and $\Phi_{Z_2} = (f^{z_1}, \ldots, f^{z_1} | \ldots | f^{z_a}, \ldots, f^{z_a})$ contain the families $f^{x_\lambda} \in \mathcal{F}_X$ and $f^{z_\lambda} \in \mathcal{F}_Z$ exactly $n_{x_\lambda} - 1$ and



(a) Items of stack set Q_τ.

(b) Families of stack set Q_τ.

(c) Stack set Q_τ selected.

(d) Stack set Q_τ not selected.

Figure 4.4: Stack set Q_τ related to triple τ.

$n_{z_\lambda} - 1$ times for $\lambda = 1, \ldots, a$ and hence have length $\sum_{\lambda=1}^{a}(n_{x_\lambda} - 1) = \sum_{\lambda=1}^{a}(n_{z_\lambda} - 1) = |M| - a$.

To illustrate the construction, we consider a small example with $a = 2$, sets $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, $Z = \{z_1, z_2\}$ and five tuples

$$M = \{(x_1, y_1, z_1), (x_2, y_1, z_2), (x_2, y_1, z_1), (x_2, y_2, z_2), (x_2, y_2, z_1)\}.$$

Then we have $m = 3|M| = 15$ stacks and $2m = 30$ items belonging to $3a = 6$ families. These items are stored as shown in Fig. 4.5a. The family retrieval sequence is

$$\Phi = (f^{z_1}, f^{z_2} | f^{x_2}, f^{x_2}, f^{x_2} | f^{\tau_1}, f^{\tau_2}, f^{\tau_3}, f^{\tau_4}, f^{\tau_5} | f^{y_1}, f^{y_2} | f^{z_1}, f^{z_1}, f^{z_2} | f^{\tau_1}, f^{\tau_2}, f^{\tau_3}, f^{\tau_4}, f^{\tau_5})$$

with related sequence indices

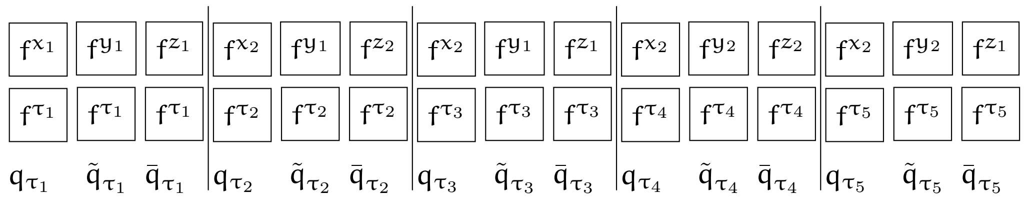
$$\mathcal{L} = (1, 2 | 3, 4, 5 | 6, 7, 8, 9, 10 | 11, 12 | 13, 14, 15 | 16, 17, 18, 19, 20).$$

Φ has the length $4|M|$ and contains two parts of length $2|M|$: The first part includes all a families from $\mathcal{F}_Z$ in Φ_{Z_1}, then $|M| - a$ families from $\mathcal{F}_X$ in Φ_X, and all $|M|$ triple families in Φ_M. The second part includes all a families from $\mathcal{F}_Y$ in Φ_Y, then $|M| - a$ families from $\mathcal{F}_Z$ in Φ_{Z_2}, and again all $|M|$ triple families in Φ'_M.

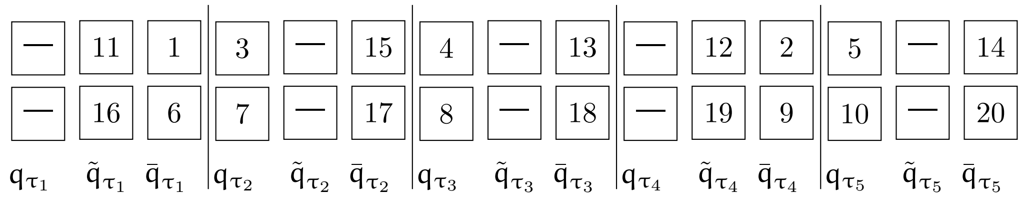
For each stack set, two items from the top level must be selected: one belonging to family $\mathcal{F}_X$ or $\mathcal{F}_Z$ for the first part of Φ, one from $\mathcal{F}_Y$ or $\mathcal{F}_Z$ for the second part. Furthermore, in each part, $|M|$ items from the bottom level have to be assigned to the $|M|$ triple families in $\mathcal{F}_M$. Overall, for family $\mathcal{F}_X$ exactly $|M| - a$ items must be selected, for $\mathcal{F}_Y$ we need exactly a items, and for $\mathcal{F}_Z$ all $|M|$ items must be selected.

We claim that a solution for 3DM exists if and only if a solution for the corresponding instance of the BRPIF with $BI = 0$ exists. In the example above, we may choose (x_1, y_1, z_1) and (x_2, y_2, z_2) as a feasible 3DM solution and assign the sequence indices as in Fig. 4.5b without any badly placed items. On the other hand, if we consider $M' = M \setminus \{(x_2, y_2, z_2)\}$, then no feasible 3DM solution exists and also no assignment without badly placed items.

"$\Rightarrow$": A solution for 3DM means that a subset $U \subset M$ exists which contains a triples covering all elements from X, Y, Z. To get a solution for the BRPIF, we choose the a stack sets corresponding to the triples $\tau \in U$ for Φ_Y, Φ_{Z_1} as in Figure 4.4c, and the remaining $|M| - a$ stack sets for Φ_X, Φ_{Z_2} as in Figure 4.4d (also cf. Figure 4.5). With this selection we are able


f^{x_1} f^{y_1} f^{z_1} f^{x_2} f^{y_1} f^{z_2} f^{x_2} f^{y_1} f^{z_1} f^{x_2} f^{y_2} f^{z_2} f^{x_2} f^{y_2} f^{z_1}
f^{τ_1} f^{τ_1} f^{τ_1} f^{τ_2} f^{τ_2} f^{τ_2} f^{τ_3} f^{τ_3} f^{τ_3} f^{τ_4} f^{τ_4} f^{τ_4} f^{τ_5} f^{τ_5} f^{τ_5}
q_{τ_1} $\tilde{q}_{\tau_1}$ $\bar{q}_{\tau_1}$ q_{τ_2} $\tilde{q}_{\tau_2}$ $\bar{q}_{\tau_2}$ q_{τ_3} $\tilde{q}_{\tau_3}$ $\bar{q}_{\tau_3}$ q_{τ_4} $\tilde{q}_{\tau_4}$ $\bar{q}_{\tau_4}$ q_{τ_5} $\tilde{q}_{\tau_5}$ $\bar{q}_{\tau_5}$

(a) Items depicted by their families.


— 11 1 3 — 15 4 — 13 — 12 2 5 — 14
— 16 6 7 — 17 8 — 18 — 19 9 10 — 20
q_{τ_1} $\tilde{q}_{\tau_1}$ $\bar{q}_{\tau_1}$ q_{τ_2} $\tilde{q}_{\tau_2}$ $\bar{q}_{\tau_2}$ q_{τ_3} $\tilde{q}_{\tau_3}$ $\bar{q}_{\tau_3}$ q_{τ_4} $\tilde{q}_{\tau_4}$ $\bar{q}_{\tau_4}$ q_{τ_5} $\tilde{q}_{\tau_5}$ $\bar{q}_{\tau_5}$

(b) Items assigned to sequence indices.

Figure 4.5: Example.

to assign the families of Φ_M, Φ'_M without causing any badly placed items and hence have a solution with $BI = 0$.

"$\Leftarrow$": We show that for a solution of the BRPIF with $BI = 0$, all demanded item families of Φ must be selected from a stack sets according to the pattern shown in Figure 4.4c and from $|M| - a$ stack sets according to the pattern in Figure 4.4d.

To guarantee $BI = 0$, the $2|M|$ families of the second part in Φ have to be combined in exactly $|M|$ stacks. A combination of top items for the second part with bottom items for the first part leads to $BI > 0$ since the families from Φ_M have to be retrieved earlier than Φ_Y and Φ_{Z_2}. In every stack set, one family of Φ_M and one family of Φ'_M must be assigned to one of the three bottom items $i_\tau, \tilde{i}_\tau, \bar{i}_\tau$, which implies that also one family from $\Phi_X \cup \Phi_{Z_1}$ and one family from $\Phi_Y \cup \Phi_{Z_2}$ must be assigned to each stack set.

Together with Φ_{Z_1} or Φ_{Z_2}, we need to assign either one family of Φ_Y or Φ_X. Since families of Φ_Y, Φ_{Z_2} may only be combined with families of Φ'_M in one stack, we assign either families of Φ_Y together with Φ_{Z_1} in a stack set Q_τ as in Fig. 4.4c or families of Φ_X together with Φ_{Z_2} as in Fig. 4.4d.

Since $|\Phi_Y| = |\Phi_{Z_1}| = a$ and $|\Phi_X| = |\Phi_{Z_2}| = |M| - a$, we must choose a stack sets as in Fig. 4.4c, and $|M| - a$ stack sets as in Fig. 4.4d. Let U be set of all triples τ for which the corresponding stack set Q_τ is chosen as in Fig. 4.4c. Then U builds a solution for 3DM. $\square$

Note that the construction in the proof also works if there is no limit on the stack height. Thus, the problem is also hard in the case $b = \infty$.

Now, we consider the variant BRPIF$_{set}$ where the family retrieval sequence is relaxed to a set. Instead of a family retrieval sequence Φ, we are given a multiset $\Phi_{set} = \{\phi_1, \dots, \phi_L\}$ of families. Again, for each index k an item i with $f_i = \phi_k$ must be selected, but the order is not fixed. For example, the relaxation of the retrieval sequence $\Phi = (A, A, B, D, A, E, F, F, A, C)$ is $\Phi_{set} = \{A, A, A, A, B, C, D, E, F, F\}$ which means that we need four items belonging to family A, one item for families B, C, D, E each, and two items for family F. However, their retrieval order is completely flexible.

Theorem 4.2. *For the BRPIF$_{set}$ with $b \geqslant 3$ it is strongly $\mathcal{NP}$-complete to decide whether a solution without any badly placed items exists.*

Proof. We again give a reduction from 3DM. For a given 3DM instance, we construct an instance of the BRPIF$_{set}$ by identifying each element $x \in X, y \in Y, z \in Z$ with one item family f^x, f^y, f^z resulting in $3a$ different families. Let $\mathcal{F}_X, \mathcal{F}_Y, \mathcal{F}_Z$ be the corresponding sets of families. For each triple $(x_q, y_q, z_q) \in M$ in 3DM, we create a stack q containing items i_q, j_q, h_q with families $f_{i_q} = f^{x_q}, f_{j_q} = f^{y_q}, f_{h_q} = f^{z_q}$ resulting in $m = |M|$ stacks with $b = 3$. In stack q, item i_q is placed at the top level, item j_q in the middle, and item h_q at the bottom level. The set of demanded families is $\Phi_{set} = \mathcal{F}_X \cup \mathcal{F}_Y \cup \mathcal{F}_Z$.

We claim that a solution for 3DM exists if and only if a solution for the corresponding instance of the BRPIF$_{set}$ with $BI = 0$ exists.

"$\Rightarrow$": A solution for 3DM means that a triples exist covering all elements from X, Y, Z. To get a solution for the BRPIF$_{set}$, we choose the a stacks related to these triples, assign the items in these stacks to the $3a$ demanded families of Φ_{set}, and have a solution with $BI = 0$.

"$\Leftarrow$": We show that for a solution of the BRPIF$_{set}$ with $BI = 0$, all demanded families of Φ_{set} must be chosen from exactly a stacks, i.e., all items of these a stacks are selected and consequently all $|M| - a$ other stacks contain only items which are not selected.

The demanded families f^x, f^y, f^z can only be assigned to items placed at the top level, the middle, and the bottom level, respectively. In a solution for the BRPIF$_{set}$ we must have selected $a = |\mathcal{F}_Z|$ items on the bottom level from a stacks for the demanded families in $\mathcal{F}_Z$. Similarly, for the a families in $\mathcal{F}_Y$ and $\mathcal{F}_X$ we must have selected a items from the middle and the top level, respectively. Furthermore, for all $f^y \in \mathcal{F}_Y$ and all $f^x \in \mathcal{F}_X$ we must have chosen a stack where an item on the bottom level has been selected for family $f^z \in \mathcal{F}_z$, since otherwise at least one badly placed item exists. Thus, the selected a stacks for the $3a$ demanded families of Φ_{set} correspond to a triples from M and hence build a solution for 3DM. □

On the other hand, this problem variant becomes easy for stack height $b = 2$.

Theorem 4.3. *The BRPIF$_{set}$ with $b = 2$ and objective RS can be solved as maximum flow problem in polynomial time.*

Proof. To solve this problem, we proceed in two phases. In the first phase we decide how many items are selected from the top level $l = 2$ and the bottom level $l = 1$, respectively. Afterwards, we assign specific items from these levels to the demanded families. Let n_f be the number of occurrences of family f in the multiset Φ_{set} and η_f^l be the number of items which belong to family f and are stored at level $l \in \{1, 2\}$. Items located at the bottom level of a stack containing only one item count as top items since these items are never badly placed.

We partition n_f into two numbers $\alpha_f^l \leqslant \eta_f^l$ for $l = 1, 2$ such that $n_f = \alpha_f^1 + \alpha_f^2$ where α_f^l denotes how many items are selected from level l. We choose $\alpha_f^2 \leqslant \eta_f^2$ as large as possible, i.e., we select as many items from the top level as possible (which ensures that these items are not badly placed).

After all family entries from Φ_{set} have been assigned to either the top or the bottom level, the families must be assigned to specific items in the stacks. For all families f with $\eta_f^2 \leqslant n_f$ we choose all the η_f^2 items in the top level. If afterwards an item i on the bottom level exists for which the item above has been selected, $\alpha_{f_i}^2 = \eta_{f_i}^2$ (i.e., there are no more items of family f_i at the top level which may be used to get access to other items at the bottom level), and $\alpha_{f_i}^1 > 0$, we choose i also.

In the following we assume that the numbers α_f^l are adjusted so that they correspond to the remaining demands. To fulfill them, we solve a maximum flow problem. The node set V consists of a source node s, a sink node t, a set of "bottom nodes" V^b, a set of "top nodes" V^t and a set of "stack nodes" V^s:

- For all families f for which an item has to be selected from the bottom level (i.e., $\alpha_f^1 > 0$) we introduce α_f^1 bottom nodes $u_f^1, \dots, u_f^{\alpha_f^1}$.
- For all families f for which an item has to be selected from the top level (i.e., $\alpha_f^2 > 0$) we introduce α_f^2 top nodes $v_f^1, \dots, v_f^{\alpha_f^2}$.
- For all stacks q where both items may be selected (i.e., for both corresponding families f_{ql} in levels $l = 1, 2$ we have $\alpha_{f_{ql}}^l > 0$), we introduce two stack nodes q, q'.

The source node s is connected to every bottom node $u_f \in V^b$, every top node $v_f \in V^t$ is connected to the sink t, and every stack node q is linked to its partner node q' ensuring that every stack can be used at most once. Furthermore, we have an arc from a bottom node $u_f \in V^b$ to a stack node q if an item belonging to family f is stored at the bottom level of q. Analogously, we have an arc from stack node q' to a top node $v_f \in V^t$ if an item belonging to family f is stored at the top level of stack q. All arcs get the upper capacity one.

We then calculate a maximum flow from the source s to the sink t. It decomposes into s-t-paths carrying one unit of flow where exactly one node from V^b and one node from V^t are matched to a stack q which means that the corresponding items are selected for the demanded families. All nodes in V^b that cannot be matched lead to a badly placed item.

A	A	A	E	F	A	B	F	D	D
B	C	D	B	C	E	A	B	C	B
1	2	3	4	5	6	7	8	9	10

(a) Initial storage.

$$\begin{tikzcd}
 & V^b & & V^s & V^t & \\
 & & q_1 \arrow[r] & q_1' \arrow[rd] \arrow[rddd] & & \\
 & u_B^1 \arrow[ru] \arrow[rddddd] & & & v_A^1 \arrow[rdd] & \\
 & & q_2 \arrow[r] & q_2' \arrow[ru] \arrow[rd] & & \\
s \arrow[ruu] \arrow[r] \arrow[rdd] & u_B^2 \arrow[ruuu] \arrow[rddd] & & & v_A^2 \arrow[r] & t \\
 & & q_5 \arrow[r] & q_5' \arrow[rd] & & \\
 & u_C^1 \arrow[ruuu] \arrow[ru] & & & v_F^1 \arrow[ruu] & \\
 & & q_8 \arrow[r] & q_8' \arrow[ru] & &
\end{tikzcd}$$

(b) Corresponding maximum flow problem.

$\underline{A}$	$\underline{A}$	A	$\underline{E}$	F	A	$\underline{B}$	$\underline{F}$	D	D
$\underline{B}$	$\underline{C}$	D	$\underline{B}$	C	E	A	$\underline{B}$	C	B
1	2	3	4	5	6	7	8	9	10

(c) Selected items.

Figure 4.6: Example for $\Phi_{set} = \{A, A, B, B, B, B, C, E, F\}$.

Example 4.4. *In Figure 4.6 we illustrate the procedure using an example with $m = 9$ stacks, and 6 families A, B, C, D, E, F. We assume that the families $\Phi_{set} = \{A, A, B, B, B, B, C, E, F\}$ have to be retrieved from the storage depicted in Figure 4.6a.*

We have

$$n_A = 2, n_B = 4, n_C = 1, n_D = 0, n_E = 1, n_F = 1,$$
$$\eta_A^2 = 4, \eta_B^2 = 1, \eta_C^2 = 0, \eta_D^2 = 2, \eta_E^2 = 1, \eta_F^2 = 2,$$
$$\eta_A^1 = 1, \eta_B^1 = 4, \eta_C^1 = 3, \eta_D^1 = 1, \eta_E^1 = 1, \eta_F^1 = 0,$$

which leads to

$$\alpha_A^2 = 2, \alpha_B^2 = 1, \alpha_C^2 = 0, \alpha_D^2 = 0, \alpha_E^2 = 1, \alpha_F^2 = 1,$$
$$\alpha_A^1 = 0, \alpha_B^1 = 3, \alpha_C^1 = 1, \alpha_D^1 = 0, \alpha_E^1 = 0, \alpha_F^1 = 0.$$

For families B and E, the two top items in stacks 4 and 7 are immediately chosen due to $\eta_B^2 = 1 < 4 = n_B$ and $\eta_E^2 = 1 = n_E$. Then for family B also the bottom item in stack 4 is selected, since $\alpha_B^2 = \eta_B^2$ and $\alpha_B^1 > 0$.

For the remaining items to be selected we consider the maximum flow problem shown in Figure 4.6b. For this problem, a maximum flow exists in which all bottom nodes are covered if we send one unit of flow through the three paths $u_B^1 \to q_1 \to q_1' \to v_A^1$, $u_B^2 \to q_8 \to q_8' \to v_F^1$, and $u_C^1 \to q_2 \to q_2' \to v_A^2$. Thus, for the solution depicted in Figure 4.6c no badly placed items exist and hence no reshuffles are necessary.

On the other hand, if we have $\Phi'_{set} = \{A, B, B, B, B, C, E, F\}$, then only one A-node in V^t exists and the maximum flow value is $2 < |V^b|$. We can send flow through the paths $u_B^1 \to q_1 \to q_1' \to v_A^1$ and $u_B^2 \to q_8 \to q_8' \to v_F^1$ (or $u_C^1 \to q_5 \to q_5' \to v_F^1$), which implies that one item (belonging to family B or C) has to be selected that leads to a badly placed item above it.

Up to now, we only considered the total number of badly placed items and not the total number of reshuffles. We claim that if we have to retrieve a multiset Φ_{set} from a storage with $b = 2$, always $RS = BI$ holds, i.e., each badly placed item must only be reshuffled once. If $BI = 0$, then also $RS = 0$ since no item must be reshuffled. On the other hand, if $BI > 0$, then at least one non-selected item at the top level blocks a selected item below. All selected items which are not blocked can immediately be retrieved. If then an empty slot above a non-selected item or a completely free stack exists, we simply relocate the blocking item to this location, retrieve the blocking item and have a completely empty stack afterwards. The only problematic case is that no empty slot exists and all top level items are badly placed. But then no feasible solution exists. □

Finally, we consider a variant "between" the BRPIF and the BRPIF$_{set}$ where not the whole retrieval sequence may be relaxed, but only some subsequences are flexible. We assume that the retrieval sequence $\Phi = (\mathcal{M}_1, \dots, \mathcal{M}_z)$ consists of a sequence of multisets $\mathcal{M}_\lambda \subset \mathcal{F}$ where the order of the multisets has to be respected, but inside each multiset the retrieval order is flexible.

For example, we may have $\Phi = (\{A, A, B\}, \{D, A\}, \{E, F, F\}, \{A, C\})$ with $\mathcal{M}_1 = \{A, A, B\}$, $\mathcal{M}_2 = \{D, A\}$, $\mathcal{M}_3 = \{E, F, F\}$, $\mathcal{M}_4 = \{A, C\}$. According to Φ, at first items for the set $\mathcal{M}_1$ have to be retrieved (i.e., two items from family A and one item from family B). These three items may be retrieved in the order (A, A, B), (A, B, A) or (B, A, A). Afterwards, the items from $\mathcal{M}_2 = \{D, A\}$ may be retrieved in the sequence (A, D) or (D, A), and so on.

Having a closer look into the proof of Theorem 4.1, we see that for $b = 2$ this problem is already hard for two multisets:

Corollary 4.1. *For the BRPIF with* $b = 2$ *where the family retrieval sequence* $\Phi = (\mathcal{M}_1, \mathcal{M}_2)$ *consists of two multisets, it is strongly* $\mathcal{NP}$*-complete to decide whether a solution without any badly placed items exists.*

Proof. Recall the proof of Theorem 4.1 where the sequence Φ was defined by the subsequences $\Phi_{Z_1}, \Phi_Y, \Phi_M, \Phi_X, \Phi_{Z_2}, \Phi'_M$. It is easy to check that the proof is also valid if each subsequence is relaxed to a set from which the families can be retrieved in an arbitrary order. Furthermore, if we set $\mathcal{M}_1 = \Phi_{Z_1} \cup \Phi_Y \cup \Phi_M$ and $\mathcal{M}_2 = \Phi_X \cup \Phi_{Z_2} \cup \Phi'_M$, the proof remains valid without any changes. □

4.4 Solution approaches for the BRPIF

This section is devoted to solution approaches for the BRPIF with two different objective functions. At first, we concentrate on minimizing the RS. We present an IP formulation in Section 4.4.1, a simple heuristic in Section 4.4.2, and a two-stage simulated annealing algorithm in Section 4.4.3. Finally, IP formulations and some dominance properties for the objective BI are introduced in Section 4.4.4, which give lower bounds for the main objective RS.

4.4.1 An IP formulation minimizing RS

In this subsection, we adapt the IP formulation for the unrestricted BRP of Caserta et al. [19] which uses discrete time steps to model the movements in the storage. In each time step, either a relocation move from one stack to another or a retrieval move may be performed and the objective is to minimize the number of time steps to retrieve all selected items from the storage. Since the variables of this IP formulation are defined for all time steps, we need an upper bound T on their number, which may be estimated by a heuristic algorithm. In our experiments, we used the heuristic described in Section 4.4.2 to compute this value. We denote by $\mathcal{T} := \{1, \dots, T\}$ the set of all time steps and describe the initial configuration for all $i \in \mathcal{I}$, $q \in \mathcal{Q}$, and $L \in \mathcal{B}$ by

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

For all items $i \in \mathcal{I}$, stacks $q, u \in \mathcal{Q}$, levels $l, r \in \mathcal{B}$, and time steps $t \in \mathcal{T}$ we have binary variables

$$x_{iqlurt} = \begin{cases} 1, & \text{if item } i \text{ is moved from stack } q \text{ at level } l \text{ to stack } u \text{ at level } r \\ & \text{in time step } t \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, for all $i \in \mathcal{I}$, $q \in \mathcal{Q}$, $l \in \mathcal{B}$, and $t \in \mathcal{T}$ we have binary variables

$$y_{iqlt} = \begin{cases} 1, & \text{if item } i \text{ is retrieved from stack } q \text{ at level } l \text{ in time step } t \\ 0, & \text{otherwise} \end{cases}$$

$$z_{iqlt} = \begin{cases} 1, & \text{if item } i \text{ is stored in stack } q \text{ at level } l \text{ at the beginning of time step } t \\ 0, & \text{otherwise.} \end{cases}$$

For all $i \in \mathcal{I}$, $k \in \mathcal{L}_i$, and $t \in \mathcal{T}$ we introduce binary variables

$$\xi_{ikt} = \begin{cases} 1, & \text{if item } i \text{ is assigned to index } k \text{ of } \Phi \text{ and retrieved in time step } t \\ 0, & \text{otherwise.} \end{cases}$$

Then, the binary IP reads as follows. In comparison to the IP formulation of Caserta et al. [19], we added the variables ξ_{ikt}, the constraints (4.9), (4.10), (4.11), and adapted constraints (4.4), (4.12) as well as the objective (4.3).

$$
\begin{aligned}
&(\mathrm{IP_{RS}}) \quad \min \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} t\cdot\xi_{iLt} && && (4.3)\\
\text{s.t.}\quad &\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{B}} z_{iqlt} + \sum_{k\in\mathcal{L}_i}\sum_{t'=1}^{t-1}\xi_{ikt'} = 1 && (i\in\mathcal{I}; t\in\mathcal{T}\setminus\{1\}) && (4.4)\\
&\sum_{i\in\mathcal{I}} z_{iqlt} \leqslant 1 && (q\in\mathcal{Q}; l\in\mathcal{B}; t\in\mathcal{T}) && (4.5)\\
&\sum_{i\in\mathcal{I}} z_{iqlt} - \sum_{i\in\mathcal{I}} z_{iq,l+1,t} \geqslant 0 && (q\in\mathcal{Q}; l\in\mathcal{B}\setminus\{b\}; t\in\mathcal{T}) && (4.6)\\
&\sum_{i\in\mathcal{I}}\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{B}}\sum_{u\in\mathcal{Q}}\sum_{r\in\mathcal{B}} x_{iqlurt} + \sum_{i\in\mathcal{I}}\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{B}} y_{iqlt} \leqslant 1 && (t\in\mathcal{T}) && (4.7)\\
&z_{iqlt} - z_{iql,t-1} \\
&-\sum_{q\in\mathcal{Q}}\sum_{r\in\mathcal{B}} x_{iqlur,t-1} + \sum_{q\in\mathcal{Q}}\sum_{r\in\mathcal{B}} x_{iurql,t-1} + y_{iql,t-1} = 0 && (i\in\mathcal{I}; u\in\mathcal{Q}; l\in\mathcal{B}; t\in\mathcal{T}\setminus\{1\}) && (4.8)\\
&\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{B}} y_{iqlt} - \sum_{k\in\mathcal{L}_i}\xi_{ikt} = 0 && (i\in\mathcal{I}; t\in\mathcal{T}) && (4.9)\\
&\sum_{i\in\mathcal{I}_{\phi_k}}\sum_{t\in\mathcal{T}}\xi_{ikt} = 1 && (k\in\mathcal{L}) && (4.10)\\
&\sum_{k\in\mathcal{L}_i}\sum_{t\in\mathcal{T}}\xi_{ikt} \leqslant 1 && (i\in\mathcal{I}) && (4.11)\\
&\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} t\cdot\xi_{ikt} - \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} t\cdot\xi_{ik't} \geqslant 1 && (k,k'\in\mathcal{L}, k>k') && (4.12)\\
&z_{iql1} = \mathrm{Init}_{iql} && (i\in\mathcal{I}; q\in\mathcal{Q}; l\in\mathcal{B}) && (4.13)\\
&x_{iqlurt} \in\{0,1\} && (i\in\mathcal{I}; q,u\in\mathcal{Q}; l,r\in\mathcal{B}; t\in\mathcal{T}) && (4.14)\\
&y_{iqlt}, z_{iqlt} \in\{0,1\} && (i\in\mathcal{I}; q\in\mathcal{Q}; l\in\mathcal{B}; t\in\mathcal{T}) && (4.15)\\
&\xi_{ikt} \in\{0,1\} && (i\in\mathcal{I}; k\in\mathcal{L}_i; t\in\mathcal{T}) && (4.16)
\end{aligned}
$$

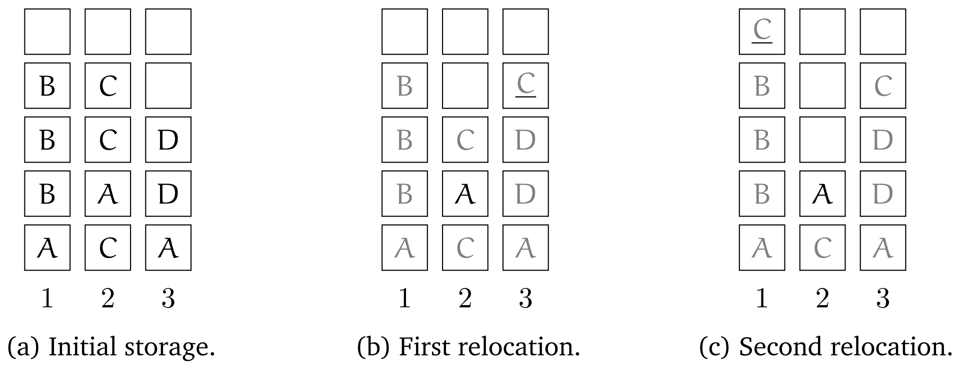
The objective in (4.3) minimizes the time step in which the item chosen for the last index L of the family retrieval sequence Φ is retrieved. The first constraints (4.4) ensure that an item is either in the storage or has been retrieved. Constraints (4.5)–(4.8), (4.13) are taken from the original formulation and guarantee correct movements in the storage. The connection between the variables y and ξ is established by constraints (4.9), while constraints (4.10) assure that to every index of the family retrieval sequence an item with the demanded family is assigned. Furthermore, constraints (4.11) guarantee that every item i is assigned to at most one index k from the family retrieval sequence with $f_i = \phi_k$. Finally, constraints (4.12) ensure the correct priority order implied by the family retrieval sequence, while constraints (4.13) establish the initial positions Init_{iql} of all items in the storage.

The stated IP formulation is very complex and tests showed that only very small instances with at most 12 items could be solved to optimality in less than one hour.

4.4.2 A simple heuristic

In this subsection, we present a simple constructive heuristic which is used by the German company mentioned above. A similar procedure has also been described in Tang et al. [102].

The algorithm for retrieving appropriate items considers the family retrieval sequence $\Phi = (\phi_1, \dots, \phi_L)$ by processing one entry ϕ_k after the other without any look-ahead to further elements of the sequence. At iteration k among all items in the storage that are not selected before, an item i with family $f_i = \phi_k$ is selected which has a minimal number of blocking items above. If more than one item meets this criterion, then the item in the stack with the smallest index is chosen. After selecting a specific item i, the blocking items above i are relocated. Iteratively, we move each blocking item to a stack containing currently the smallest number of items. In case of ties, the stack with smallest index is preferred.



(a) Initial storage. (b) First relocation. (c) Second relocation.

Figure 4.7: Selecting an item belonging to family A.

Example 4.5. *An example of the heuristic procedure is depicted in Figure 4.7. The initial storage is shown in Figure 4.7a, all items are represented by their families* A, B, C, D*, and the stacks are indexed from 1 to 3. We assume that an item belonging to family* A *has to be chosen next. Among the three candidate items the one in stack 2 is selected since it has the smallest number of blocking items (two items) above and the smallest stack index 2. Then, the topmost blocking item is relocated to the third stack since this stack currently contains the smallest number of items (cf. Figure 4.7b). Finally, in Figure 4.7c the second blocking item is relocated from stack 2 to stack 1, since now among the two candidate stacks 1,3 (containing the same number of items), the first stack has the smallest index.*

4.4.3 A simulated annealing algorithm

As stated in Section 4.4.1, the IP formulation can only be used for very small instances. On the other hand, the simple heuristic from Section 4.4.2 usually does not provide good solutions.

In the following, we propose a heuristic approach based on SA that aims at finding good solutions with a small number of reshuffles. The SA algorithm was implemented in a standard way (cf. Eglese [30]) using a standard acceptance criterion and geometric cooling. During the cooling process, the number of probably realizable iterations is periodically estimated and, based on this information, the (usually fixed) cooling factor is adjusted so that a predefined final temperature is reached in the end. Moreover, the algorithm operates in two altering phases. While in the first phase (selection stage) the current selection of items is changed, in the second phase (retrieval stage) the corresponding unrestricted BRP instance is evaluated w.r.t. the total number of reshuffles needed to retrieve the selected items. For the evaluation, we use the (re-implemented) bottom level heuristic of Jin et al. [57] as BRP solver. This heuristic combines a fast runtime with a relatively good solution quality which is beneficial to evaluate a large number of selections in the first phase.

One may start with a randomly generated solution by traversing the family retrieval sequence and randomly selecting any appropriate item that has not been selected before. Alternatively, each item selection obtained by a heuristic for the BRPIF (e.g., by the heuristic from Section 4.4.2 or the selection computed by the IP from Section 4.4.4) may be a feasible starting point.

As described in Section 4.2, a solution of the selection stage is represented by an item retrieval sequence $S = (i_1, \dots, i_L)$. For generating neighbors in the first phase, we use a swap neighborhood. Two items i, j with $f_i = f_j$ are chosen where at least one of the items is assigned to an index $k \in \mathcal{L}$ of the family retrieval sequence. Then, the assigned entries (or the status of being not assigned) of i, j are exchanged. Note that this swap neighborhood is connected, i.e., a finite number of swaps allows to reach every solution $S' = (i'_1, \dots, i'_L)$ starting from any other solution $S = (i_1, \dots, i_L)$. This can be seen as follows. At first, all items $i \in S \setminus S'$ are swapped with items $j \in S' \setminus S$ such that the families of the items are respected. This results in a solution $\bar{S}$ which contains the same items as S' but possibly in a different order. Second, the items in the solution $\bar{S}$ are swapped family-wise until the target order from S' is reached.

In order to accelerate the search process, we tried to reduce the search space and to avoid the evaluation of some neighbors by the BRP solver. Recall that the family retrieval sequence Φ induces priority values for the retrieval of the selected items as described in Section 4.2. More specifically, if item i is assigned to index $k \in \mathcal{L}$, then we set the priority value of item i to $p_i := k$. This mapping leads to pairwise different priority values $1, \dots, L$ for all selected items. However, this restriction can be relaxed in the following situation:

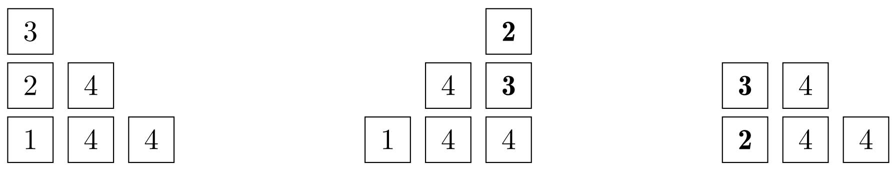
Proposition 4.1. *Assume that in the family retrieval sequence Φ for two consecutive indices $k, k+1$ the same family is demanded (i.e., $\phi_k = \phi_{k+1}$). Then instead of setting the priority values of the selected items i, j to $p_i = k, p_j = k+1$, we may also set $p_i = p_j = k$.*

Proof. If two items of the same family are demanded at positions k and $k+1$ in the retrieval sequence and two different items i, j with $f_i = f_j$ are selected, then it is feasible to retrieve item i or item j first. $\square$

The advantage of this property is that less neighbors may be generated in the first stage since two items get the same priority value. Otherwise, solutions with $p_i = k, p_j = k+1$ (i.e., $i_k = i, i_{k+1} = j$) and with $p_j = k, p_i = k+1$ (i.e., $i_k = j, i_{k+1} = i$) would be considered as two different solutions. Furthermore, it may be advantageous to postpone the decision which item is retrieved first to the evaluation stage where the BRP solver may have more information to make a good decision.

4.4.4 Lower bounds

Finally, this subsection is devoted to lower bound calculations to evaluate the quality of heuristic solutions. For this purpose, we consider the objective function BI, which is a lower bound on the total number of reshuffles. Especially, if $BI = 0$, we know that no reshuffles are necessary. We propose two IP formulations.



(a) Initial storage. (b) Items 3, 2 relocated. (c) Items 2, 3 pushed back.

Figure 4.8: Example for push-backs.

Winter and Zimmermann [116] considered a variant of the BRPIF where all relocated items are directly pushed back after each retrieval, i.e., blocking items above the current target item may be relocated to other stacks, but must be relocated back to their initial stack (resulting in the same order as before) after the target item is removed from the storage. In Figure 4.8 an example is given. Item 1 is the current target item and shall be removed from the storage (cf. Figure 4.8a). Items $3, 2$ are blocking item 1 and must be relocated (cf. Figure 4.8b). After item 1 is removed from the storage, items $2, 3$ are pushed back to their initial stack so that they appear in the initial order afterwards (cf. Figure 4.8c). In the objective function "reshuffles with push-back" $\mathrm{RS}_{\mathrm{PB}}$ the relocation and the push-back are counted as one reshuffle.

In the following, we state the linearized quadratic *mixed-integer programming* (MIP) formulation from Winter and Zimmermann [116] after applying the linearization technique of Kaufman and Broeckx [62] as suggested in Winter and Zimmermann [116]. Afterwards we adapt this MIP with small changes to the objective BI. Let $\mathcal{L}^+ := \mathcal{L} \cup \{L+1\}$ and $\mathcal{L}_i^+ := \mathcal{L}_i \cup \{L+1\}$. Furthermore, let $\mathcal{I}_k := \{i \in \mathcal{I} \mid f_i = \phi_k\}$ be the set of all items which may be assigned to sequence index k. For $i, j \in \mathcal{I}$ and $k \in \mathcal{L}_i^+, \lambda \in \mathcal{L}_j^+$ we define the parameter

$$u_{ijk\lambda} = \begin{cases} 1, & \text{if item } i \text{ blocks item } j \text{ in the case that } i \text{ is assigned to index } k \\ & \text{and } j \text{ is assigned to the earlier index } \lambda < k \\ 0, & \text{otherwise} \end{cases}$$

i.e., $u_{ijk\lambda} = 1$ if item i is stored somewhere above j in the same stack, but has to be retrieved later. Moreover, for $i \in \mathcal{I}$ and $k \in \mathcal{L}_i^+$ let

$$v_{ik} = \sum_{j \in \mathcal{I}} \sum_{\lambda \in \mathcal{L}_j^+} u_{ijk\lambda}$$

be an upper bound on the number of items blocked by item i if i is assigned to index k.

For all $i \in \mathcal{I}$, $k \in \mathcal{L}_i^+$ we have variables

$$y_{ik} = \begin{cases} 1, & \text{if item } i \text{ is assigned to sequence index } k \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, for all $i \in \mathcal{I}$ variables z_i indicate the number of items blocked by item i. Then, the problem to minimize $\mathrm{RS}_{\mathrm{PB}}$ can be formulated as follows.

$$(\mathrm{IP}_1^{\mathrm{BI}}) \quad \min \sum_{i \in \mathcal{I}} z_i \tag{4.17}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{L}_i^+} y_{ik} = 1 \qquad \forall i \in \mathcal{I} \tag{4.18}$$

$$\sum_{i \in \mathcal{I}_k} y_{ik} = 1 \qquad \forall k \in \mathcal{L} \tag{4.19}$$

$$\sum_{j \in \mathcal{I}} \sum_{\lambda \in \mathcal{L}_j} u_{ijk\lambda} \cdot y_{j\lambda} - z_i \leqslant v_{ik} \cdot (1 - y_{ik}) \qquad \forall i \in \mathcal{I}, k \in \mathcal{L}_i^+ \tag{4.20}$$

$$y_{ik} \in \{0, 1\} \qquad \forall i \in \mathcal{I}, k \in \mathcal{L}_i^+ \tag{4.21}$$

$$z_i \geqslant 0 \qquad \forall i \in \mathcal{I} \tag{4.22}$$

In the objective (4.17), the number of reshuffles with push-back $\mathrm{RS}_{\mathrm{PB}}$ is minimized. Due to (4.18), each item is assigned exactly once to an index of the family retrieval sequence or to index $L+1$. Similarly, constraints (4.19) ensure that to every index of the family retrieval sequence exactly one appropriate item is assigned. Finally, constraints (4.20) connect the variables y and z such that z_i counts the number of items blocked by item i.

To adjust this MIP to the objective BI, at first we simply make variable z_i binary such that it indicates whether item i is blocking any item or not. Together with this adjustment, we change constraints (4.20) slightly by multiplying parameter v_{ik} to variable z_i which results in

$$\sum_{j\in\mathcal{I}}\sum_{\lambda\in\mathcal{L}_j} u_{ijk\lambda}\cdot y_{j\lambda} - v_{ik}\cdot z_i \leqslant v_{ik}\cdot(1-y_{ik}) \qquad \forall i\in\mathcal{I}, k\in\mathcal{L}_i^+ \tag{4.23}$$

The resulting integer program is called IP_1^{BI}. Since in some preliminary tests IP_1^{BI} showed a relatively poor performance, we decided to develop a further IP formulation IP_2^{BI} based on ideas from Boge and Knust [8] for a loading problem.

While up to now we identified each item by a unique number $i \in \mathcal{I}$, in the following, we use the stack index q and the level index l to refer to an item. The item which is stored in a specific stack q at a certain level l is denoted by i_{ql}, its corresponding family by f_{ql}. The set $\mathcal{L}_{ql} := \{k \in \mathcal{L} \mid \phi_k = f_{ql}\} \cup \{L+1\}$ contains all indices k in the family retrieval sequence Φ to which item i_{ql} can be assigned as well as the artificial value $L+1$ indicating that item i_{ql} is not selected. Furthermore, let $\mathcal{B}_q$ be the set of all occupied levels in stack q.

For all $q \in \mathcal{Q}$, $l \in \mathcal{B}_q$ and $k \in \mathcal{L}_{ql}$ we introduce binary variables

$$\beta_{qlk} = \begin{cases} 1, & \text{if item } i_{ql} \text{ is assigned to index } k \text{ and is badly placed} \\ 0, & \text{otherwise} \end{cases}$$

$$\gamma_{qlk} = \begin{cases} 1, & \text{if item } i_{ql} \text{ is assigned to index } k \text{ and is well placed} \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let $\mathcal{B}_q^l := \{r \in \mathcal{B}_q \mid r > l\}$ be the set of all occupied levels above level l in stack q, let $\mathcal{L}_{qr}^k := \{\lambda \in \mathcal{L}_{qr} \mid \lambda > k\}$ be the set of all indices in $\mathcal{L}_{qr}$ after index k, and let

$$a_{qlk} = \begin{cases} 1, & \text{if item } i_{ql} \text{ satisfies } f_{ql} = \phi_k \\ 0, & \text{otherwise.} \end{cases}$$

Then, the binary IP reads as follows.

$$(\text{IP}_2^{BI}) \quad \min \sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{B}_q}\sum_{k\in\mathcal{L}_{ql}} \beta_{qlk} \tag{4.24}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{L}_{ql}} (\beta_{qlk} + \gamma_{qlk}) = 1 \qquad \forall q\in\mathcal{Q}, l\in\mathcal{B}_q \tag{4.25}$$

$$\sum_{q\in\mathcal{Q}}\sum_{l\in\mathcal{B}_q} a_{qlk}\cdot(\beta_{qlk} + \gamma_{qlk}) = 1 \qquad \forall k\in\mathcal{L} \tag{4.26}$$

$$\beta_{qlk} + \gamma_{qlk} + \sum_{\lambda\in\mathcal{L}_{qr}^k} \gamma_{qr\lambda} \leqslant 1 \qquad \forall q\in\mathcal{Q}, l\in\mathcal{B}_q, r\in\mathcal{B}_q^l, k\in\mathcal{L}_{ql} \tag{4.27}$$

$$\beta_{qlk}, \gamma_{qlk} \in \{0,1\} \qquad \forall q\in\mathcal{Q}, l\in\mathcal{B}_q, k\in\mathcal{L}_{ql} \tag{4.28}$$

In the objective (4.24), the total number of badly placed items is minimized. Due to constraints (4.25) every item i_{ql} is assigned to exactly one index of the family retrieval sequence or to the "not selected" value. Similarly, constraints (4.26) ensure that to every index of the family retrieval sequence exactly one item belonging to the demanded family is assigned. Finally, constraints (4.27) guarantee that if item i_{ql} is assigned to sequence index k (either as badly or well placed item, i.e., $\beta_{qlk} + \gamma_{qlk} = 1$), then any item at a higher level $r > l$ in stack q assigned to a later index $\lambda > k$ cannot be well placed (i.e., we must have $\sum_{\lambda\in\mathcal{L}_{qr}^k} \gamma_{qr\lambda} = 0$). Note that in contrast to the IP stated in Boge and Knust [8], we use a stronger version with aggregated constraints (4.27). We call this formulation IP_2^{BI}. Similarly to Boge and Knust [8], we also use this formulation to compute heuristic solutions. For this purpose, we first calculate a solution for IP_2^{BI} w.r.t. the objective function BI and evaluate the resulting configuration with a BRP solver to minimize the total number of reshuffles afterwards. We call this approach $IP_2^{BI \to RS}$. Moreover, we use the solution calculated by IP_2^{BI} as a start solution for our SA algorithm and call this combination $IP_2^{BI} + SA$. These more complex heuristics are compared to the simple heuristic of the company in Section 4.5.3.

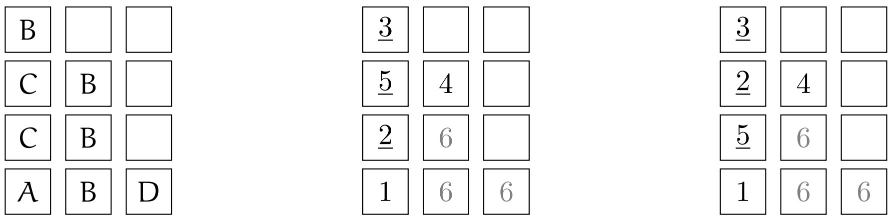
In the following, we propose some techniques to reduce the number of variables and constraints. Tang and Ren [103] presented two properties where one solution S is dominating another solution S' with respect to the crane workload, which means that it is not necessary to consider S' if S is considered. Since in Tang and Ren [103] it is assumed that the storage area is sufficiently large and each blocking item can be shuffled to a stack where it can stay until it is retrieved, these properties are also valid for the objective BI. The first property considers two items i, j with $f_i = f_j$ where item i is stored on top of item j in the same stack. It is shown that the solution $S = (i_1, \dots, i_k = i, i_{k+1} = j, \dots, i_L)$ is not worse than $S' = (i_1, \dots, i'_k = j, i'_{k+1} = i, \dots, i_L)$. The second property considers two items i, j with $f_i = f_j$ where items i, j are stored in two different stacks. It is shown that the solutions $S = (i_1, \dots, i_k = i, i_{k+1} = j, \dots, i_L)$ and $S' = (i_1, \dots, i'_k = j, i'_{k+1} = i, \dots, i_L)$ have the same objective value.

Note that the first property from Tang and Ren [103] considers the case that items i, j are assigned to adjacent indices $k, k+1$ in the sequences S, S' and item i is stored somewhere above item j in the same stack. If we have the situation that i is stored directly on top of item j, we also have dominance in the case of non-adjacent indices $k < \lambda$ when swapping items i, j in the selection sequence:

Proposition 4.2. *Consider two items i, j with $f_i = f_j$ where item i is stored directly on top of item j in the same stack. Then the solution $S = (i_1, \dots, i_k = i, \dots, i_\lambda = j, \dots, i_L)$ is not worse than $S' = (i_1, \dots, i'_k = j, \dots, i'_\lambda = i, \dots, i_L)$.*

Proof. As described in Tang and Ren [103], if item i is stored above item j, then in solution S' item i blocks item j while this is not true for solution S. However, this also holds for two items i, j that are assigned to non-adjacent indices in the retrieval sequence if item i and j are stored directly on top of each other, since then the status of blocking other items is not affected by the swap. □

Example 4.6. *Consider the example in Figure 4.9 with families A, B, C, D and the family retrieval sequence $\Phi = (A, C, B, B, C)$.*



(a) Storage. (b) Selection 1. (c) Selection 2.

Figure 4.9: $\Phi = (A, C, B, B, C)$.

In Figure 4.9b and 4.9c two possible selections are shown where items with priorities $1, 2, 3, 4, 5$ are selected, while items with priority 6 are not. Both selections have three badly placed items (those with priorities $2, 3, 5$, underlined). Let us now consider the family retrieval sequence $\Phi' = (C, B, B, C)$. In this case, the bottom element in the first stack is not selected (i.e., gets the priority value 6). Then, in Fig. 4.9c the number of badly placed items decreases from 2 to 1, which shows that also a strict improvement may be possible.

In a preprocessing step, by applying Properties 4.1 and 4.2 to $\mathrm{IP}_2^{\mathrm{BI}}$, several constraints and some variables in (4.27) can be eliminated. Every constraint of this type refers to the relationship of two items i_{ql}, i_{qr} in the same stack q where i_{qr} is stored on a higher level $r > l$, item i_{ql} is assigned to index k and item i_{qr} is assigned to a later index $\lambda > k$. We must have $f_{i_{ql}} = f_{i_{qr}}$ to apply the two properties.

At first we consider the case $r = l + 1$ where Property 4.2 can be applied. We claim that we can omit the constraint

$$\beta_{qlk} + \gamma_{qlk} + \sum_{\lambda \in \mathcal{L}^k_{q,l+1}} \gamma_{qr\lambda} \leqslant 1$$

without changing the set of optimal solutions. If we omit the constraint and set $\beta_{qlk} = 1$ or $\gamma_{qlk} = 1$, we may also set $\gamma_{q,l+1,\lambda'} = 1$ for an index $\lambda' \in \mathcal{L}^k_{q,l+1}$ (and $\beta_{q,l+1,\lambda'} = 0$) unless item $i_{q,l+1}$ is classified as badly placed due to another blocked item. Then, in the corresponding solution the meaning of $\gamma_{q,l+1,\lambda'} = 1$ is incorrect since item $i_{q,l+1}$ is not recognized as a badly placed item and hence not correctly counted in the objective function. However, we may "repair" this solution due to Property 4.2. Instead of assigning item i_{ql} to index k and item $i_{q,l+1}$ to index $\lambda' > k$, we assign item i_{ql} to index λ' and item $i_{q,l+1}$ to index k. Since this resolves the neglected blockage, we get an equivalent feasible solution with the same (correct) objective value where the changed variables have the correct meaning.

Now we consider the case $\lambda = k + 1$ where Property 4.1 can be applied. Let $k' \geqslant k + 1$ be the largest index with $\phi_k = \phi_{k+1} = \ldots = \phi_{k'}$ and define the set $\mathcal{L}^{k,k'}_{qr} = \{\lambda \in \mathcal{L}^k_{qr} | \lambda \leqslant k'\}$. Then we may eliminate any variable $\gamma_{qr\lambda}$ with $\lambda \in \mathcal{L}^{k,k'}_{qr}$ in inequality (4.27) or eliminate even the whole constraint if $\mathcal{L}^k_{qr} \setminus \mathcal{L}^{k,k'}_{qr} = \emptyset$. If we omit variable $\gamma_{qr\lambda'}$ with $\lambda' \in \mathcal{L}^{k,k'}_{qr}$, we may assign item i_{ql} to index k and item i_{qr} to index λ', without recognizing the blocking. However, by swapping the assigned indices for these two items, a correct solution with the same objective value is obtained. Note that Property 4.1 also covers both properties of Tang and Ren [103].

In the following, we present a result which can be used to exclude some items from being selected for the item retrieval sequence. For this purpose, let $n_f = |\{k \in \mathcal{L} \mid \phi_k = f\}|$ be the number of all indices in Φ requiring family f. For all families f with $n_f > 0$ let $\mathcal{I}^q_f = \{i_{ql} \in \mathcal{I} \mid f_{i_{ql}} = f\}$ be the set of all items in stack q belonging to family f. Furthermore, let $i_{ql_1}, i_{ql_2}, \ldots, i_{ql_{|\mathcal{I}^q_f|}}$ be the sequence of all items in the set $\mathcal{I}^q_f$ ordered by descending levels, i.e., $l_1 > l_2 > \ldots > l_{|\mathcal{I}^q_f|}$. We define a "critical level" l^q_f for stack q and family f as follows. If a family f is not required in Φ (i.e., $n_f = 0$), we set $l^q_f = b + 1$. Otherwise, if $n_f > 0$, we set

$$l^q_f = \begin{cases} l_{|\mathcal{I}^q_f|}, & \text{if } |\mathcal{I}^q_f| \leqslant n_f \\ l_{n_f}, & \text{otherwise.} \end{cases}$$

If stack q contains at most as many items of family f as are required in Φ (i.e., $|\mathcal{I}^q_f| \leqslant n_f$), then l^q_f corresponds to the lowest level of an item in $\mathcal{I}^q_f$ (meaning that all items in $\mathcal{I}^q_f$ may be selected). Otherwise, only the highest n_f items in $\mathcal{I}^q_f$ are relevant and l^q_f corresponds to the lowest level of these items. Let l^q be the lowest critical level in stack q among all families, i.e.,

$$l^q := \min_{f \in \mathcal{F}} \{l^q_f\}$$

and $\mathcal{A}_q = \{i_{ql} \mid l < l^q\}$ be the set of all items that are located below level l^q in stack q (if $l^q = 1$, the set $\mathcal{A}_q$ is empty, if $l^q = b + 1$, then $\mathcal{A}_q$ contains all items in stack q).

Example 4.7. *Consider the stack q in Figure 4.10 with families A, B, C and the family retrieval sequence $\Phi = (A, C, C)$. We have $n_A = 1, n_B = 0, n_C = 2$, $l^q_A = 5, l^q_B = 7, l^q_C = 3$ and hence $l^q = 3$.*

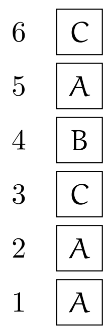


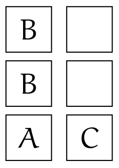
Figure 4.10: Stack with six levels.

Proposition 4.3. *For all stacks $q \in \mathcal{Q}$, all items $i \in \mathcal{A}_q$ can be excluded from being selected without changing the minimal* BI*-value.*

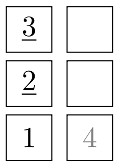
Proof. Consider an optimal solution not satisfying Property 4.3, i.e., in some stack q at least one item from $\mathcal{A}_q$ is selected. Among these items let i be the item with lowest level. In this case we must have another non-selected item $j \notin \mathcal{A}_q$ in stack q above i with $f_j = f_i$. Since item j blocks all selected items below, j must be badly placed. If we select j instead of i, the number of badly placed items does not increase. Repeating this procedure for every item $i \in \mathcal{A}_q$ for all stacks q leads to a solution satisfying Property 4.3 without increasing the number of badly placed items. □

The result of Property 4.3 allows us to exclude items from the selection stage completely. This may also be used in a preprocessing step for IP_2^{BI} by eliminating variables related to these items. More specifically, for each stack q the variables $\beta_{qlk}, \gamma_{qlk}$ for all $l < l^q$ and all $k \in \mathcal{L}_{ql}$ can be removed from the IP. Moreover, also all constraints (4.25) for all $l < l^q$ and all $k \in \mathcal{L}_{ql}$ can be eliminated. Constraints (4.26) as well as the objective function (4.24) do not include the removed variables any more. Furthermore, all constraints (4.27) involving at least one of the removed variables can be omitted.

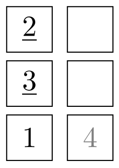
Finally, we would like to note that Property 4.2 (even for adjacent indices $k, k+1$ as considered in Tang and Ren [103]) and Property 4.3 do not hold for the objective RS.



(a) Storage.



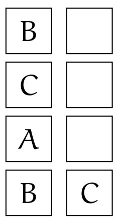
(b) Selection 1.

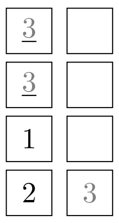


(c) Selection 2.

Figure 4.11: $\Phi = (A, B, B)$.

Example 4.8. *Consider the example in Figure 4.11 with families* A, B, C *and the family retrieval sequence* $\Phi = (A, B, B)$*. In Figure 4.11b and 4.11c two possible selections are shown where items with priorities* $1, 2, 3$ *are selected, while items with priority* 4 *are not. Both selections have two badly placed items (those with priorities 2,3, underlined). However, if we consider the total number of reshuffles, the selection in Figure 4.11b needs only two reshuffles to retrieve all items, while the selection in Figure 4.11c needs three. Thus, by Property 4.2, the better selection would be excluded.*


(a) Storage.


(b) Selection 1.

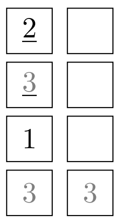

(c) Selection 2.

Figure 4.12: $\Phi = (A, B)$.

Example 4.9. *Consider the example in Figure 4.12 with families A, B, C and the family retrieval sequence $\Phi = (A, B)$. In Figure 4.12b and 4.12c two possible selections are shown where items with priorities $1, 2$ are selected. Both selections have two badly placed items in the first stack above item 1. By Property 4.3 the bottom item in the first stack would be excluded due to $l^1 = 2$ and hence the first selection would not be considered. However, if we consider RS, the selection in Figure 4.12b needs only two reshuffles, while the selection in Figure 4.12c needs three. Thus, by Property 4.3, the better selection would be excluded.*

4.5 Computational study

In this section, we report results of our computational study. We used an Intel(R) Core(TM) i5-3470 CPU with 3.2 GHz and 32 GB RAM. The IP formulations were solved with CPLEX 12.9, all heuristics were implemented in Java 11. In all computational results, the BRPIF without push-backs is considered, and the unloading stage is solved as unrestricted BRP. After describing the used test data in Section 4.5.1, we compare the two IP formulations in Section 4.5.2. In Section 4.5.3, we present results of the heuristics, in Section 4.5.4 we consider the impact of relaxing the family retrieval sequence. Finally, in Section 4.5.5, we study which parameters are most important for the difficulty of an instance.

4.5.1 Test data

Table 4.1: Varied parameters in instances from the literature.

	publications							
	[7]	[116]	[101]	[102]	[95]	[88]	[103]	[38]
# instances	1300	300	810	810	512	10	300	5
L	×	×	×	×	×	×	×	×
$\frac{F}{n}$	×	×	×	×	×		×	×
b	×	×	×	×	×		×	×
m	×	×	×					
Ψ				×	×		×	×

In the literature, some variants of the BRPIF have already been considered in several publications and in most of them new test data was randomly generated (only Winter and Zimmermann [116] used the same instances as Blasum et al. [7]). All publications with relevant instances and their varied parameters are listed in Table 4.1, where the columns are ordered by the year of the publication. In the literature it is claimed that all these parameters influence the problem complexity visibly. The parameters L, F, b, m, n have already been introduced in Section 4.2. Furthermore, Ψ is a parameter called “slab distribution factor” in Fernandes et al. [38], “complexity factor” in Singh et al. [95], and “stack number factor” in Tang et al. [102]. In Tang et al. [102] it was verbally described as “the ratio of the number of stacks that host the candidate slabs to the minimum number of stacks that these candidate slabs might be stored in”. We interpret this description of Ψ as

$$\Psi = \frac{m}{\left\lceil \frac{|\mathcal{I}_{can}|}{b} \right\rceil}$$

where $\mathcal{I}_{can} = \{i \in \mathcal{I} | \exists k \in \mathcal{L}, \phi_k = f_i\}$ is the set of possible candidate items that may be assigned to a sequence index. The value of Ψ is at least equal to one and in the literature it is stated that smaller values of Ψ indicate a larger problem complexity. The number $|\mathcal{I}_{can}|$ of items that may be selected for retrieval is an important parameter since the corresponding items have to be considered, while all other items merely lead to blockages. If $|\mathcal{I}_{can}|$ equals the maximum value n, then Ψ is approximately $\frac{m \cdot b}{n}$. This value is the reciprocal of the parameter “occupancy” $occ = \frac{n}{m \cdot b}$, a relevant parameter for the BRP (cf., e.g. Expósito-Izquierdo et al. [35]).

When inspecting real-world data we received from the German company mentioned above, we identified an additional important feature that we call *item distribution* and which describes the “mixture” of items belonging to certain families. We identified mainly four important item distributions: *constant*, *linear*, *quadratic*, and *exponential*. A constant item distribution means that every family contains nearly the same number of items. A linear distribution means that after sorting the families by their cardinalities it is possible to find a linear function that describes the numbers of items belonging to the families. For example, if family f_1 contains $n_{f_1} = 2$ items, family f_2 contains $n_{f_2} = 4$ items, family f_3 contains $n_{f_3} = 6$ items, and so on, then the function $n_{f_i} = 2i$ for $i = 1, 2, 3, \ldots$ is an appropriate linear function. Quadratic and exponential distributions are defined analogously.

Among all publications included in Table 4.1, we were only able to obtain the STIM data from Blasum et al. [7]. These instances have $F \in \{2, 3, 4, 5, 6\}$, $n \in \{25, 50, 75, 100, 125, 150\}$, $L = n$, $m = \frac{n}{5}$, $\Psi = 1$, and constant item distributions. The stack height is always limited by $b \leqslant 7$. However, these instances are a bit more general since they include different stack-dependent heights $b_q \in \{1, \ldots, 7\}$. These instances were randomly generated and per parameter configuration 10 instances exist, resulting in 300 instances in total.

Table 4.2: Parameters of data set C.

	week 1		week 2	
segmentation	complete	day	complete	day
# instances	24	91	23	86
m	$[1, 14]$	$[1, 14]$	$[1, 22]$	$[1, 22]$
n	$[4, 408]$	$[2, 408]$	$[11, 470]$	$[8, 470]$
occ	$[0.06, 0.6]$	$[0.03, 0.6]$	$[0.18, 0.51]$	$[0.06, 0.51]$
L	$[1, 33]$	$[1, 9]$	$[2, 128]$	$[1, 30]$
$\frac{L}{n}$	$[0.01, 0.75]$	$[0.0, 0.5]$	$[0.0, 0.88]$	$[0.0, 0.65]$
F	$[1, 65]$	$[1, 65]$	$[1, 49]$	$[1, 49]$
$\frac{F}{n}$	$[0.07, 0.25]$	$[0.07, 0.5]$	$[0.03, 0.51]$	$[0.03, 0.87]$
Ψ	$[1, 7]$	$[1, 14]$	$[1, 9]$	$[1, 22]$

Furthermore, we used a real-world data set from the company mentioned above, which we call *Company* (C). We analyzed the parameters of these instances and summarize their characteristics in Table 4.2. This data set contains instances belonging to two different weeks (week 1, week 2). Each week is considered in two different scenarios related to the knowledge of the family retrieval sequence Φ: either the complete sequence of the week is known or the sequence is split into five smaller subsequences, each representing a day. All instances have the same stack height $b = 60$, but are really heterogeneous according to other parameters. There exist small and large instances (according to the parameters n and L) with constant, linear, quadratic and exponential item distributions. However, mostly a linear distribution can be found in the data.

Table 4.3: Parameters of data set AC.

no	parameter	range	#
1	m	$\{3, 6, 9, \dots, 36, 39\}$	$13 \cdot 40$
2	b	$\{10, 20, 30, \dots, 80, 90\}$	$9 \cdot 40$
3	$\frac{L}{n}$	$\{0.1, 0.2, \dots, 0.9, 1.0\}$	$10 \cdot 40$
4	occ	$\{0.1, 0.2, \dots, 0.9, 1.0\}$	$10 \cdot 40$
5	$\frac{F}{n}$	$\{0.05, 0.1, \dots, 0.45, 0.5\}$	$10 \cdot 40$
6	distribution	{const., lin., quad., exp.}	$4 \cdot 40$
7	Ψ	$\{1, 1.5, 2, 2.5, 3, 4, 5\}$	$7 \cdot 40$

Additionally, we generated further instances to take a wider range of scenarios into account. The generation process was based on the knowledge gathered from the real-world data and inspired by the parameters investigated in the literature. Starting from a typical setting with the parameters

$$m = 20, b = 45, \mathrm{occ} = 0.4, \frac{L}{n} = 0.2, \frac{F}{n} = 0.2, \Psi \approx 3, \text{ and a linear item distribution} \tag{4.29}$$

we varied each parameter in seven different data series. In this context, a data series is a set of instances where only one parameter is changed according to a certain increment, while all other parameters are fixed. With these data series we want to evaluate which parameters are most important for the difficulty of an instance. We investigate this question in an experiment later on in Section 4.5.5.

We call this data set *Artificial Company* (AC). For each data series, the number of instances and parameter ranges are summarized in Table 4.3. For example, the parameter b is varied in the data series with the number 2. Nine different stack heights $b \in \{10, 20, \dots, 80, 90\}$ are used to generate 40 instances per parameter assignment. For the instance generation, we first created an empty storage of appropriate dimensions (m, b). Then items are generated fulfilling the parameters F, occ and the desired item distribution by fixing the number of families F and randomly drawing for each item a family with a roulette-wheel selection based on the item distribution until occ is reached. Then, to place the items into the storage, we used the loading algorithm of the company. This simple algorithm stores each newly arriving item in a stack containing currently the smallest number of items, preferring the stack with smallest index in case of ties. With this procedure, all stacks are almost equally filled (± 1) in the end. Finally, the family retrieval sequence is created by shuffling the whole item list and using the first L items of that list. The generated test instances of data set AC can be found at `http://www2.informatik.uos.de/kombopt/data/brpif/`.

4.5.2 IP results

In this section, we present some results concerning the two integer linear programming formulations $\mathrm{IP}_1^{\mathrm{BI}}$ and $\mathrm{IP}_2^{\mathrm{BI}}$ with the objective BI proposed in Section 4.4.4. Furthermore, we study the impact of Properties 4.1–4.3.

Comparison of $\mathrm{IP}_1^{\mathrm{BI}}$ and $\mathrm{IP}_2^{\mathrm{BI}}$

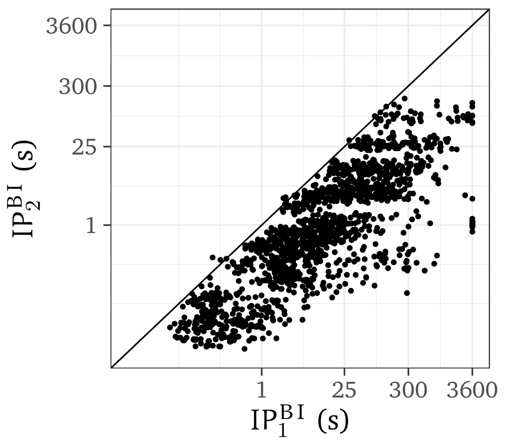
At first we compare the performance of $\mathrm{IP}_1^{\mathrm{BI}}$ and $\mathrm{IP}_2^{\mathrm{BI}}$. Since some preliminary tests revealed that $\mathrm{IP}_1^{\mathrm{BI}}$ was not able to solve most of the instances in the data set AC within one hour, we only used the 300 instances of the data set STIM as well as the 224 instances of the data set C. We solved each instance with $\mathrm{IP}_1^{\mathrm{BI}}$ and $\mathrm{IP}_2^{\mathrm{BI}}$ (without applying Properties 4.1–4.3) with a time limit of one hour five times using five different seeds.

In Figure 4.13, two scatter plots are depicted, including 1500 and 1120 data points for the two different data sets. Each point indicates the computational runtimes (in seconds) for $\mathrm{IP}_1^{\mathrm{BI}}$ on the horizontal axis and for $\mathrm{IP}_2^{\mathrm{BI}}$ on the vertical axis (both with a logarithmic scale) for one specific instance with a specific seed. The diagonal line indicates that both IPs need the same computational runtime, points above or below this line indicate that $\mathrm{IP}_1^{\mathrm{BI}}$ or $\mathrm{IP}_2^{\mathrm{BI}}$ is faster in solving the corresponding problem instance.

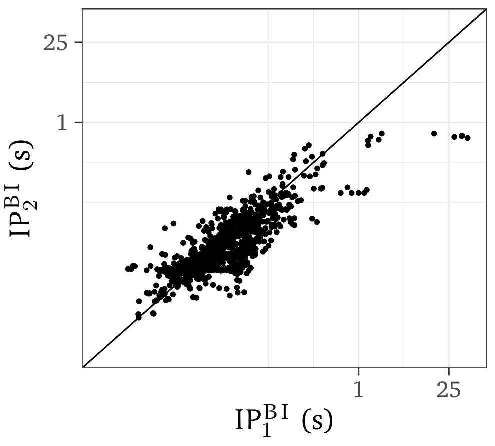
For the data set STIM, all instances could be solved to optimality by $\mathrm{IP}_2^{\mathrm{BI}}$, 22 instances could not be verified to be optimally solved by $\mathrm{IP}_1^{\mathrm{BI}}$ within the time limit. The plot in Figure 4.13a shows that most instances could be solved faster by $\mathrm{IP}_2^{\mathrm{BI}}$, only 9 among the 1500 points are above the diagonal. Moreover, the points above the diagonal are relatively close to it, which means that the differences in the computational time are rather small.

For the data set C, all instances could be solved very fast by CPLEX, which means that the initialization process and other CPLEX-related routines have a larger impact on the runtime (as the relative amount of runtime for the initialization is larger in this case). The plot in Fig. 4.13b has 623 points below and 484 points above the diagonal for a computational time of less than one second. The 13 points with a runtime > 1s could be clearly solved faster with $\mathrm{IP}_2^{\mathrm{BI}}$, confirming the results of the STIM instances.

Overall, $\mathrm{IP}_2^{\mathrm{BI}}$ clearly outperforms $\mathrm{IP}_1^{\mathrm{BI}}$ in the direct comparison of the runtime as well as in the number of verified optimal solutions. Hence, we use $\mathrm{IP}_2^{\mathrm{BI}}$ for all further experiments.



(a) Data set STIM.



(b) Data set C.

Figure 4.13: Comparison of $\mathrm{IP}_1^{\mathrm{BI}}$ and $\mathrm{IP}_2^{\mathrm{BI}}$.

Impact of Properties 4.1–4.3 on IP_2^{BI}

To evaluate the impact of the dominance properties, we solved IP_2^{BI} using no properties (PN), Property 4.1 (P1), Property 4.2 (P2), Property 4.3 (P3) and all three properties (PA) with a time limit of one hour. Each instance of the three data sets was solved five times using five distinct seeds.

For the data set C, the model without applying any property contains 221 variables and 595 constraints on average. The application of P1 reduces the number of variables in constraints (4.27) by 2.7% on average, the application of P2 causes a reduction of 14.4% in the number of constraints in the whole model. Moreover, on average the application of P3 reduces the number of variables by 63.8% and the number of constraints by 62.8%. Obviously, all single properties reduce the size of the model, but P3 has the highest impact among the non-combined rules. This effect is probably caused by the large values of Ψ in this data set. However, the combination of all properties gives the largest reduction. All instances of the data set C could be solved to optimality within one second by all IP-variants.

Since for all instances of the data set STIM, we have $L = n$ (i.e., all items must be retrieved), Property 4.3 cannot be applied. The model without applying any property contains 2 899 variables and 6 285 constraints on average. P1 reduces both the number of constraints in the model as well as the number of variables in constraints (4.27) on average by 0.7%. P2 reduces the number of constraints on average by approx. 14.5%. The combination of both properties is slightly stronger with a reduction of 14.9% in the number of constraints and 0.7% in the number of eliminated variables in constraints (4.27) (exactly the same as for P1).

Table 4.4: Average/maximum runtimes of IP_2^{BI} using different properties on data set STIM.

	n			
Prop.	$\leqslant 75$	100	125	150
PN	< 1.0 / 5.1	3.2 / 13.7	10.0 / 45.5	28.3 / 179.6
P1	< 1.0 / 6.1	2.4 / 8.9	7.3 / 27.4	18.7 / 79.6
P2	< 1.0 / 1.5	1.2 / 3.7	2.6 / 9.6	7.2 / 32.0
PA	< 1.0 / 1.2	1.2 / 3.7	2.4 / 9.3	6.7 / 30.2

The average/maximum runtimes (in seconds) applying the different properties to IP_2^{BI} are shown in Table 4.4. As it can be seen, both properties lead to reductions in runtimes, especially for larger instances. All instances of the data set STIM could be solved to optimality in at most 3 minutes by all IP-variants.

Table 4.5: Comparison of IP_2^{BI} using different properties on data set AC.

(a) All instances except those with exponential item distribution.

	runtime (s)	
Prop.	avg	max
PN	48	3600
P1	48	3600
P2	52	3600
P3	49	3600
PA	50	3600

(b) All instances with exponential item distribution.

	runtime (s)	
Prop.	avg	max
PN	2445	3600
P1	49	174
P2	81	369
P3	2358	3600
PA	18	46

Finally, the results for the data set AC are presented in Table 4.5. Here, we distinguish the instances with exponential item distribution from the remaining instances. Overall, 200 combinations of individual instances and seeds with exponential item distribution as well as 12 200 combinations of the other instances are considered. In the tables average and maximum computational runtimes (in seconds) are presented. For the instances without exponential item distribution, the model without applying any property contains 1192 variables and 8910 constraints on average. The application of P1 reduces the number of variables in constraints (4.27) by 0.5% and P2 reduces the number of constraints by 0.4%. The effect of P3 is slightly stronger with a reduction of 1.7% of all variables, which causes a reduction in the number of constraints by 1.6%. Nonetheless, the impact of the properties is relatively small and it is hence not surprising that the application of the properties has no visible effect, neither on the average nor the maximum computational runtime. Although, every IP-variant hit the time limit of one hour for some seed, every single instance could be verified to be optimally solved with every variant for at least one seed.

For the instances with exponential item distribution, the model without applying any property contains many more constraints and variables, namely 18 950 variables and 158 673 constraints on average. Again, the application of Property 4.3 has no visible effect, but Property 4.1, Property 4.2 and the combination of all properties (PA) improve the computational runtime drastically. Not only the average runtime could be reduced by 96.7% (P2), 98.0% (P1), 99.3% (PA), in addition all instances could also be verified to be optimally solved in less than one minute by the IP in combination with all properties. Presumably, this effect is related to the reduction in the number of constraints and the number of variables in (4.27) compared to PN. In this context, on average the number of constraints could be reduced by 0.0%, 39.1%, 39.1% for P1, P2, and PA, respectively. Although P1 is not able to reduce the number of constraints, the number of variables could be reduced by 12.8% on average.

In summary, the properties perform differently on the data sets, but on average they are able to reduce the size of the model and hence also the computational runtimes. Property 4.3 shows the smallest effects for the considered instances, but reduces the number of constraints on the heterogeneous data set C the most. Hence, we decided to apply all properties in $\mathrm{IP}_2^{\mathrm{BI}}$ in further experiments.

4.5.3 Heuristic results

This section is devoted to the comparison of the different heuristics minimizing the total number of reshuffles introduced in Section 4.4.2–4.4.4. These approaches include the simple heuristic used by the company, $\mathrm{IP}_2^{\mathrm{BI}}$ used as a heuristic by evaluating the corresponding solution according to RS, and the simulated annealing algorithm. All algorithms were started with a time limit of 300 seconds which seems an acceptable runtime limit in practice and each instance was solved five times with five different seeds. We used all instances of data sets C and STIM as well as all instances of data set AC with $\mathrm{occ} \leqslant 0.9$ since the 40 instances of data series 4 and all instances of data series 7 with $\mathrm{occ} = 1.0$ are infeasible w.r.t. RS due to Lemma 4.1 (i.e., there is not enough space in the storage area to apply the necessary movements). The stated values of the best known solutions (BKS) in the first row of each table are collected during all tests presented in this work (taking into account all single solutions computed with different seeds).

In Section 4.5.2 we compared the impact of the different dominance properties in the context of $\mathrm{IP}_2^{\mathrm{BI}}$. We repeated a similar experiment with the SA algorithm. Mostly, the properties show small advantages in the computational runtime and the quality of the solutions. Nonetheless, the differences are small and we decided to include only Property 4.2 for further experiments (which is also the only one that is valid for RS in general, cf. the

Table 4.7: Comparison of different heuristics on all data sets.

(a) Data set C, 224 instances.

	RS − LB			
solver	min	avg	# ver	time
BKS	148		215	–
Comp.	1195		140	< 0.1
$\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$	412	438	212	< 0.1
Comp.+SA	151	165	213	30
$\mathrm{IP}_2^{\mathrm{BI}}$+*SA*	148	155	215	15

(b) Data set STIM, 300 instances.

	RS − LB			
solver	min	avg	# ver	time
BKS	19		283	–
Comp.	2781		9	< 0.1
$\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$	58	91	264	2
Comp.+SA	21	26	282	235
$\mathrm{IP}_2^{\mathrm{BI}}$+*SA*	20	20	283	42

(c) Data set AC without exponential item distribution, 2120 instances.

	RS − LB			
solver	min	avg	# ver	time
BKS	93275		1577	–
Comp.	1752195		15	< 0.1
$\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$	202560	244020	1428	19
Comp.+SA	122089	160776	906	300
$\mathrm{IP}_2^{\mathrm{BI}}$+*SA*	113155	142217	1500	110

(d) Data set AC with exponential item distribution, 40 instances.

	RS − LB			
solver	min	avg	# ver	time
BKS	0		40	–
Comp.	184		17	< 0.1
$\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$	0	0	40	17
Comp.+SA	18	46	28	171
$\mathrm{IP}_2^{\mathrm{BI}}$+*SA*	0	0	40	16

discussion at the end of Section 4.4.4).

The results for the different data sets are summarized in Tables 4.8a–4.8d. In the first column, *Comp.* denotes the simple company heuristic. The strategy $\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$ uses CPLEX on $\mathrm{IP}_2^{\mathrm{BI}}$ in a first step to compute a single item retrieval sequence with minimum BI-value. Afterwards, in a second step this retrieval sequence is fixed and the resulting BRP with objective RS is solved with the BRP heuristic of Jin et al. [57]. Finally, *Comp.+SA* and $\mathrm{IP}_2^{\mathrm{BI}}$+*SA* utilize the SA algorithm of Section 4.4.3 with the solution of *Comp.* respectively $\mathrm{IP}_2^{\mathrm{BI}}$ (with a time limit of 60 seconds) as starting point. Let Ω be a specific set of instances, ω one certain instance of this set, and recall that every instance ω is solved five times using five different seeds s. Furthermore let RS_ω^s be the computed objective function value related to instance ω with seed s and LB_ω the lower bound value derived from an optimal solution calculated by $\mathrm{IP}_2^{\mathrm{BI}}$. Regarding this, in the major column "RS − LB" the sum $\sum_{\omega\in\Omega}(\mathrm{RS}_\omega - \mathrm{LB}_\omega)$ is shown. Particularly, in column "min", the best value $\mathrm{RS}_\omega = \min_{s=1}^{5}\{\mathrm{RS}_\omega^s\}$ is used. In column "avg" the value $\mathrm{RS}_\omega = \sum_{s=1}^{5}\mathrm{RS}_\omega^s/5$ is used to show the spread of the objective between the different runs. Note that this column is empty for the heuristic *Comp.* since this heuristic is deterministic and does not include any random components based on a seed. The column "# ver" states the number of instances verified to be optimally solved by the lower bound from $\mathrm{IP}_2^{\mathrm{BI}}$. In the last column, the average runtime (in seconds) is reported.

The results are quite similar for the data sets C, STIM, and AC without exponential item distribution. All algorithms are able to improve the solutions of the company heuristic drastically. Moreover, the solver $\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$ that uses $\mathrm{IP}_2^{\mathrm{BI}}$ and aims at minimizing the number of badly placed items BI only, performs worse than the SA algorithms that tackle the desired objective function number of reshuffles RS directly. By comparing the SA algorithms in more detail, it is obviously worth to compute a starting solution for the SA algorithm with $\mathrm{IP}_2^{\mathrm{BI}}$. Although, much more time is invested in computing a good starting solution by $\mathrm{IP}_2^{\mathrm{BI}}$ instead of *Comp.*, the overall runtime decreases on average. With the solver $\mathrm{IP}_2^{\mathrm{BI}}$+*SA* for the data sets STIM and C in total 95% of the instances could be solved to optimality.

For the data set AC with exponential item distribution, the situation is different. In this case, the approaches $\mathrm{IP}_2^{\mathrm{BI}\to\mathrm{RS}}$ and $\mathrm{IP}_2^{\mathrm{BI}}+SA$ perform best and are able to find a solution with $\mathrm{BI} = \mathrm{RS} = 0$ for every instance. Obviously, the starting solution of $\mathrm{IP}_2^{\mathrm{BI}}$ is responsible for this good performance, which highlights the strength of the IP finding solutions with zero reshuffles if possible.

4.5.4 Evaluating the impact of relaxing the family retrieval sequence

As already mentioned in the introduction, companies may be interested in the question how much they can gain w.r.t. the total retrieval time if the family retrieval sequence may be changed somehow. In practice, it may be feasible to swap adjacent entries in the retrieval sequence, for example, if they belong to the same customer order. If all entries according to one shift are requested without requiring a certain order, even the whole sequence may be flexible. Recall that according to Theorem 4.2 and Corollary 4.1 relaxed versions of the BRPIF with multisets are still $\mathcal{NP}$-complete problems. Nonetheless, for the objective BI, we are able to compute optimal solutions for several instances with $\mathrm{IP}_2^{\mathrm{BI}}$. In our experiment, we consider retrieval sequences which are relaxed to different degrees.

To solve the relaxed problems, we used $\mathrm{IP}_2^{\mathrm{BI}}$ without a time limit. We explain our experiment using the family retrieval sequence $\Phi = (A, A, B, D, A, E, F, F, A, C)$ as example. At first, every instance is solved without any relaxation. In the example, the sequence Φ is converted into $\Phi^1_{rel} = (\{A, A\}, \{B\}, \{D\}, \{A\}, \{E\}, \{F, F\}, \{A\}, \{C\})$, where consecutive entries belonging to the same family are assigned to one multiset according to Property 4.1. Afterwards, we relax the family retrieval sequence by joining every two adjacent smaller multisets of Φ^1_{rel} resulting in Φ^2_{rel}. Then we continue with Φ^2_{rel} by joining again every two adjacent multisets, and so on. In the last iteration, the whole sequence is relaxed to a single multiset. In the example, we obtain

$$\Phi^1_{rel} = (\{A, A\}, \{B\}, \{D\}, \{A\}, \{E\}, \{F, F\}, \{A\}, \{C\}) \tag{4.30}$$

$$\Phi^2_{rel} = (\{A, A, B\}, \{D, A\}, \{E, F, F\}, \{A, C\}) \tag{4.31}$$

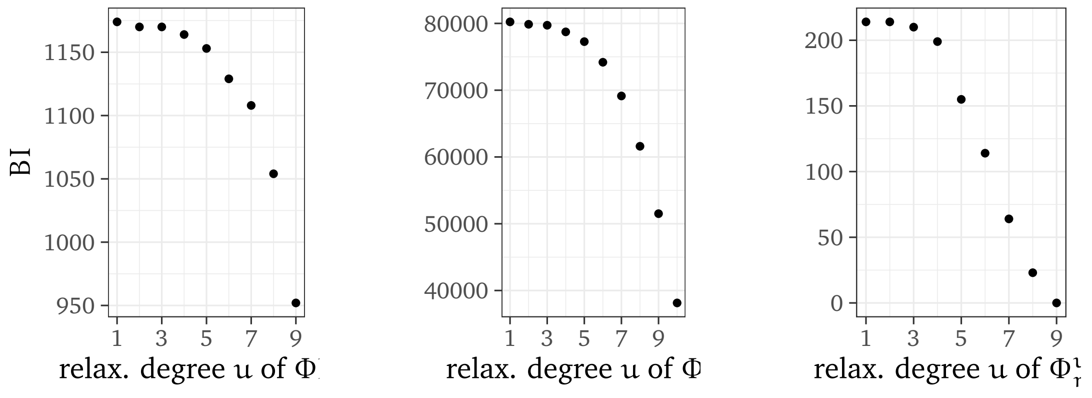
$$\Phi^3_{rel} = (\{A, A, B, D, A\}, \{E, F, F, A, C\}) \tag{4.32}$$

$$\Phi^4_{rel} = (\{A, A, B, D, A, E, F, F, A, C\}) \tag{4.33}$$

In this way, the original family retrieval sequence Φ is relaxed to sequences Φ^u_{rel} where with increasing value of the "relaxation degree" u more flexibilities arise.

The results of this experiment are summarized in Figure 4.14a–4.14c. We used all instances of the data sets C and STIM. For the data set AC we included only data series 3 (parameter L), 5 (F), 6 (item distribution), since these parameters seem to be the most important parameters influencing the structure of the retrieval sequence w.r.t. to relaxations. For all three plots the number of badly placed items BI is shown on the vertical axis for different relaxation degrees on the horizontal axis. For data sets C (Figure 4.14a) and STIM (Figure 4.14c), we plotted the sum of BI-values over all instances with the same parameter setting since the average values are very small. For data set AC (Figure 4.14b), we plotted average values since these values are very large. This means, the graphs show the overall reduction by the relaxation for data sets C and STIM, and the average reduction per instance for the data set AC.

For data set C, the savings on BI for small relaxation degrees ($u \in \{1, \dots, 4\}$) are only equal to 10, but from a value of $u = 5$ the effect on BI is much stronger resulting in savings of $\mathrm{BI} = 222$ in total if the whole sequence is relaxed. The results for the data series 3, 5, and 6 of the data set AC are quite similar: at first small relaxation degrees cause only small savings, but relaxing the whole sequence leads to large improvements. Also for the data set STIM the



(a) Data set C. (b) Data set AC. (c) Data set STIM.

Figure 4.14: Relaxing the retrieval sequence with different relaxation degrees for different data sets.

basic behavior is similar. However, since for these instances the sequence length equals the number of items ($L = n$), all instances can be solved with $BI = 0$ in the last iteration because all items shall be retrieved.

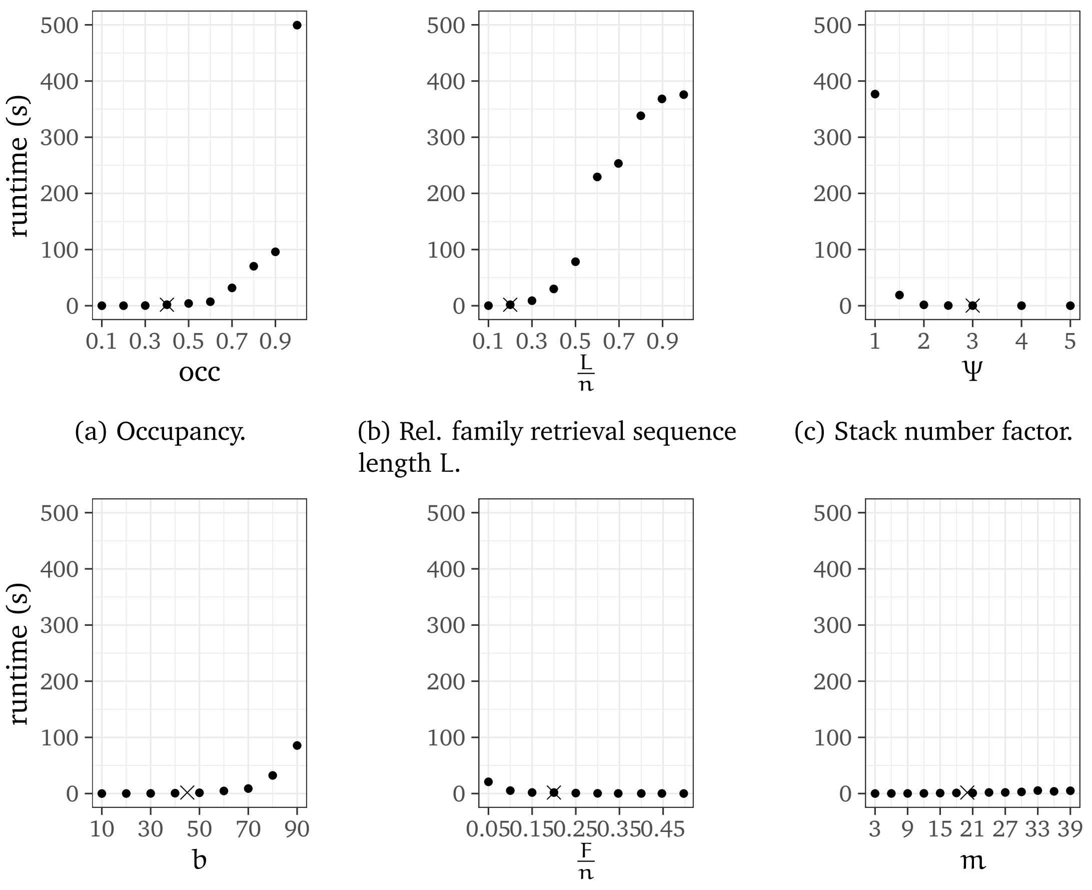
Summarizing the results, it is possible to save reshuffles by relaxing the sequence, but the strength of the relaxation in relation to the estimated profit and effort must be calculated individually as the outcome highly depends on the parameters and structure of each individual instance.

Finally, we conducted a further experiment regarding the importance of future knowledge of the retrieval sequence for the company. For this purpose, we considered the real-world data set C (consisting of data for 10 days in 2 weeks) in two different scenarios: in the first scenario we solve the problem for the whole week (i.e., we have two independent problems, each with a retrieval sequence for 5 days), in the second we solve the problem for each individual day (i.e., 10 independent problems with shorter retrieval sequences corresponding to one day). We use the heuristic IP_2^{BI}+*SA* and sum up the numbers of reshuffles for all days in the two weeks. If we compare the resulting total numbers of reshuffles, it turns out that in the first scenario (where the retrieval sequence of the whole week is known), 25% reshuffles can be saved compared to the case where only the retrieval sequence of the next day is available. This shows the importance of prospective knowledge about the family retrieval sequence.

4.5.5 Parameter analysis regarding the difficulty of instances

In our last experiment we studied how the parameters mentioned in Section 4.5.1 influence the difficulty of a problem instance to distinguish between parameters with a large impact and those with less importance. To determine the difficulty, we used IP_2^{BI}, tried to compute optimal solutions w.r.t. BI, and measured the runtime. We solved each instance of the seven different data series of data set AC with IP_2^{BI} and a time limit of one hour five times using five different seeds. The results for the IP are plotted in Figure 4.15a–4.15f. Each point represents the average over 40 instances with the same parameters, which are each solved using 5 distinct seeds, i.e., 200 different values are accumulated. The vertical axis shows the average runtime in seconds, always in the range [0,500] for a better comparison between the different parameters that are varied on the horizontal axis. The values in the typical “basis setting” for the data set AC described in equation (4.29) are marked with $\times$ in Figure 4.15a–4.15f. The average runtime of this setting is 1.7 seconds.

Figure 4.15a–4.15f state that increasing values of occ, $\frac{L}{n}$, b, m and decreasing values of $\frac{F}{n}$,



(a) Occupancy. (b) Rel. family retrieval sequence length L. (c) Stack number factor.

(d) Stack height b. (e) Rel. number of families F. (f) Number of stacks m.

Figure 4.15: Varying different instance parameters.

Ψ increase the runtime of the IP. The parameters m and $\frac{F}{n}$ show only a minor effect compared to the basis setting ($\times$), the effects caused by parameter b are only slightly stronger. The strongest effect among these six different parameters can be seen for the parameters occ, $\frac{L}{n}$, and Ψ. Particularly, an increase of parameter occ also decreases the value of parameter Ψ, since Ψ is related to the occupancy of a storage as mentioned in Section 4.5.1. Similarly, an increase in the parameter $\frac{L}{n}$ increases the value of Ψ as well, since $\frac{|\mathcal{I}_{can}|}{n}$ naturally increases with L, which in turn leads to larger values of Ψ. Thus, instead of using parameter Ψ for measuring the difficulty of an instance, one could also use occ and $\frac{|\mathcal{I}_{can}|}{n}$, which give deeper insight in the actual reasons for the difficulty. The parameter b affects the computational runtime visibly, but not as strong as Ψ, occ, and $\frac{L}{n}$.

Finally, we varied also the item distribution. While the runtimes of IP_2^{BI} for constant, linear (basis setting), and quadratic distributions exhibit a small average runtime of 1.0 to 1.7 seconds, the instances with exponential item distribution need much more time (2445 seconds on average). This effect may be explained by the huge number of variables and constraints.

Summarizing the results, not all parameters considered in the literature influence the difficulty of an instance noticeable. Especially, the parameters $\frac{F}{n}$ and m have only a minor effect and parameter Ψ seems to indicate the effect of parameters occ and $\frac{|\mathcal{I}_{can}|}{n}$. Hence, actually important parameters for analyzing the difficulty seem to be b, occ, $\frac{|\mathcal{I}_{can}|}{n}$, and the item distribution.

4.6 Conclusions

In this chapter, we studied the blocks relocation problem with item families as a generalization of the BRP and the SSS. We derived new complexity results and especially showed that the BRPIF with $b = 2$ is already strongly $\mathcal{NP}$-complete, strengthening the result for $b = 3$ from the literature.

In contrast to existing literature we assumed that the space in the storage area is limited and all movements have to be carried out within this area. We treated the practically important objective “number of reshuffles” which is more complicated than only counting the “number of badly placed items” commonly done in the literature. We proposed a new IP formulation addressing the number of badly placed items to calculate lower bounds for the objective “number of reshuffles”. Together with three new dominance properties to speed up the IP, all instances used in this work could be solved to optimality w.r.t. BI. Furthermore, we developed a two-stage simulated annealing algorithm where in the first stage appropriate items are selected and in the second stage the unloading process is optimized by a fast BRP heuristic. The results of this approach are much better than results obtained by a simple company heuristic. Using the lower bounds calculated by the new IP formulation, we were able to verify optimality of solutions for many instances (95% of the real-world data and the benchmark instances from the literature).

Additionally, we introduced a new variant of the problem where the family retrieval sequence may be relaxed. In practical experiments it was shown that relaxing the retrieval sequence as well as incorporating further prospective knowledge about the retrieval sequence may reduce the number of reshuffles considerably.

Although we made promising progress on the basic problem and some variants, additional practical features could be considered in future research (for example, concerning uncertainty and robustness). For instance, instead of exploiting that the retrieval sequence may be relaxed to a certain degree, it could also be seen from the perspective that uncertain changes in the retrieval sequence have to be incorporated in the planning of the retrieval process.

Chapter 5

Conclusions

In this chapter, we summarize the considered problems and obtained results of this thesis as well as complete the discussion with future research questions in connection with problems arising in stack-based storage systems.

In Chapter 2, we investigated the so-called *parallel stack loading problem* (PSLP) considering the surrogate objectives *total number of unordered stackings of adjacent items* ($\mathrm{US}_{\mathrm{adj}}$) and *total number of badly placed items* (BI) as well as the more realistic objective *total number of reshuffles* (RS). Our computational experiments showed that the objective BI is preferable to $\mathrm{US}_{\mathrm{adj}}$, and it may be used to sufficiently approximate the commonly used objective function RS up to a stack height of six. Nonetheless, $\mathrm{US}_{\mathrm{adj}}$ as well as BI can only approximate the objective RS. Hence, we used our developed *integer program* (IP) formulation for BI to compute optimal solutions which can be used as lower bounds to evaluate our heuristic results. Moreover, we were able to show that our simulated annealing algorithm tackling the objective RS heuristically has great advantages in the solution quality w.r.t. to the more complex objective function. In this context, we used a two-stage approach, modifying the storage assignment in the first stage with a simple neighborhood while evaluating this storage assignment in the second stage with a fast *blocks relocation problem* (BRP) heuristic.

In Chapter 3, we considered the well-known *premarshalling problem* (PMP) but incorporated uncertainty to better represent the problem in reality with this model. As the target of the PMP is to sort the items in a storage system before these items are retrieved in a specific order, we assumed that this order is uncertain and change in a certain range. We developed IP formulations and used them in connection with our theoretical analysis of the problem to find robust storage configurations in a first step. Subsequently, we used a branch-and-bound algorithm in a second step to achieve the best possible configuration (in terms of robustness) with the least possible effort (regarding reshuffles). It turned out that investing a few more movements in the premarshalling phase could cause great benefits in the unloading phase by reducing the risk of unexpected incidents.

In Chapter 4, we considered the *blocks relocation problem with item families* (BRPIF) as generalized problem of the BRP where a sequence of items specified by certain properties is demanded and items with these properties exist in a storage. We rejected an assumption commonly used in the literature which implies that every item has to be moved at most once and that it is therefore sufficient to count blocking items only. In contrast, we did not impose any requirements on the storage and focused on the more realistic objective RS instead of approximating it by just counting badly placed items. Again, we developed a simulated annealing algorithm that varies the selection of certain items regarding the demanded sequence in a first step and used a fast BRP solver from the literature to solve the problem with the already selected (and therefore fixed) items in a second step. Furthermore, we proposed a new IP formulation to further enhance the solution quality and evaluate our solutions with lower bounds obtained by this formulation.

In conclusion, it can be useful to question common assumptions frequently used in practice or in the literature in order to redefine the existing limits of a problem. The resulting freedom usually leads to an increase in the complexity of the problem, which must be compensated for. In our case, using more accurate objective functions for the PSLP and the BRPIF as well as incorporating uncertainty in the PMP made the problem models practically more relevant but did not allow to use existing solution approaches from the literature. Related to the increased complexity, exact solution methods alone are often not suited to compute applicable solutions in a reasonable amount of time on its own. However exact solutions methods may be applied to solve smaller parts of the problem and increase the solution quality noticeable. For instance in the context of the BRPIF, it was possible to apply IP formulations generating a well-structured start solution for our local search approach. Nonetheless, in many cases the increased complexity can be handled by breaking the problem down into related subproblems, which means that significantly better results can be achieved despite or more precisely because of the larger solution space. For the stated problems in the context of stack-based storage systems we were able to identify suitable subproblems by separating the evaluation of the unloading process in a separate stage. In the case of the PSLP, the decisions regarding the placement of every item in the storage could be fixed beforehand which reduced the complexity of evaluating the unloading problem considerably so that a fast BRP heuristic could be applied for that reason. In the setting regarding the BRPIF, the approach was similar, but decisions about which items to retrieve were made in a separate phase. Finally, for the robust PMP a parameter indicating a best possible robust storage configuration was computed beforehand to be able to identify such a configuration during the solving process of the PMP. In summary, it was possible for us to identify suitable subproblems in each case and to develop convenient solution algorithms in order to generate high-quality solutions overall.

Future research

In further research, other assumptions from the literature can be scrutinized in order to better relate them on real application scenarios, e.g., unlimited time for sorting all items in a storage or the exclusion of temporary additional space in the PMP which have only been investigated in [126] and [115], respectively. Both assumptions simplify the problem a lot but prevent appreciable resources from being used or important constraints from being taken into account. In addition to the options described, other assumptions exist such as moving more than one item at a time, including more than one crane and others that we mentioned in Chapter 1. Moreover, the next comprehensible step would be to replace the relatively accurate objective RS by considering the real travel time of items in a storage instead of just counting the movements performed. Indeed, attempts related to this topic can be found in [1, 6, 52, 53, 71, 76, 93, 94, 92, 109, 111] for the BRP. But already for the PMP only one (cf. [25]) and for the PSLP as well as for the BRPIF, we are not aware of a single publication which includes approaches for incorporating the mentioned objective. In this situation, the approaches for the BRP could be used to act as a promising starting point and recent publications highlight the importance of this topic. Another overall important aspect, not often considered in the literature concerns the modeling of uncertainty. For the BRP (cf. [4, 12, 13, 42, 68, 125]) and the PMP (cf. [80, 89, 105])), only a small number of publications take uncertainty into consideration. The topic becomes more and more important since uncertainties may lead to unexpected massive consequences in the daily business of companies. Therefore, it must be taken into account in several fields and could also be an important factor in stack-based storages. Particularly, the retrieval order as a central point in each of the problem stages can be subject to uncertainty.

Bibliography

[1] Azari, E., Eskandari, H., and Nourmohammadi, A. Decreasing the crane working time in retrieving the containers from a bay. *Scientia Iranica* 24.1 (2017), pp. 309–318.

[2] Bacci, T., Mattia, S., and Ventura, P. A branch-and-cut algorithm for the restricted block relocation problem. *European Journal of Operational Research* 287.2 (2020), pp. 452–459.

[3] Bacci, T., Mattia, S., and Ventura, P. The bounded beam search algorithm for the block relocation problem. *Computers & Operations Research* 103 (2019), pp. 252–264.

[4] Bacci, T., Mattia, S., and Ventura, P. The realization-independent reallocation heuristic for the stochastic container relocation problem. *Soft Computing* (2022).

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

[6] Bian, Z. and Jin, Z.-H. Optimization on retrieving containers based on multi-phase hybrid dynamic programming. *Procedia - Social and Behavioral Sciences* 96 (2013), pp. 844–855.

[7] Blasum, U., Bussieck, M. R., Hochstättler, W., Moll, C., Scheel, H.-H., and Winter, T. Scheduling trams in the morning. *Mathematical Methods of Operations Research* 49.1 (1999), pp. 137–148.

[8] Boge, S. and Knust, S. The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage. *European Journal of Operational Research* 280.3 (2020), pp. 940–952.

[9] Boge, S., Goerigk, M., and Knust, S. Robust optimization for premarshalling with uncertain priority classes. *European Journal of Operational Research* 287.1 (2020), pp. 191–210.

[10] Boge, S. and Knust, S. The blocks relocation problem with item families minimizing the number of reshuffles. *OR Spectrum* (working paper, revision submitted in 2022).

[11] Borgman, B., Asperen, E. van, and Dekker, R. Online rules for container stacking. *OR Spectrum* 32.3 (2010), pp. 687–716.

[12] Borjian, S., Galle, V., Manshadi, V. H., Barnhart, C., and Jaillet, P. *Container relocation problem: approximation, asymptotic, and incomplete information*. https://arxiv.org/abs/1505.04229. Accessed: 2022-05-30. 2015.

[13] Borjian, S., Manshadi, V. H., Barnhart, C., and Jaillet, P. *Dynamic stochastic optimization of relocations in container terminals*. https://web.mit.edu/jaillet/www/general/container13.pdf. Accessed: 2022-05-30. 2013.

[14] Bortfeldt, A. and Forster, F. A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research* 217.3 (2012), pp. 531–540.

[15] Boysen, N. and Emde, S. The parallel stack loading problem to minimize blockages. *European Journal of Operational Research* 249.2 (2016), pp. 618–627.

[16] Boywitz, D. and Boysen, N. Robust storage assignment in stack- and queue-based storage systems. *Computers & Operations Research* 100 (2018), pp. 189–200.

[17] Brink, M. van and Zwaan, R. van der. A branch and price procedure for the container premarshalling problem. *Algorithms - ESA 2014*. Ed. by A. S. Schulz and D. Wagner. Springer Berlin Heidelberg, 2014, pp. 798–809.

[18] Bruns, F., Knust, S., and Shakhlevich, N. V. Complexity results for storage loading problems with stacking constraints. *European Journal of Operational Research* 249.3 (2016), pp. 1074–1081.

[19] Caserta, M., Schwarze, S., and Voß, S. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research* 219.1 (2012), pp. 96–104.

[20] Caserta, M., Schwarze, S., and Voß, S. A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. *Evolutionary Computation in Combinatorial Optimization*. Ed. by C. Cotta and P. Cowling. Vol. 5482. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 37–48.

[21] Caserta, M. and Voß, S. A corridor method-based algorithm for the pre-marshalling problem. *Applications of Evolutionary Computing*. Ed. by M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, and P. Machado. Springer Berlin Heidelberg, 2009, pp. 788–797.

[22] Caserta, M., Voß, S., and Sniedovich, M. Applying the corridor method to a blocks relocation problem. *OR Spectrum* 33.4 (2011), pp. 915–929.

[23] Cheng, X. and Tang, L. A scatter search algorithm for the slab stack shuffling problem. *Advances in Swarm Intelligence*. Ed. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 6145. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 382–389.

[24] Choe, R., Park, T., Oh, M.-S., Kang, J., and Ryu, K. R. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. *Journal of Intelligent Manufacturing* 22.2 (2009), pp. 201–217.

[25] Dayama, N. R., Krishnamoorthy, M., Ernst, A., Narayanan, V., and Rangaraj, N. Approaches for solving the container stacking problem with route distance minimization and stack rearrangement considerations. *Computers & Operations Research* 52 (2014), pp. 68–83.

[26] de Melo da Silva, M., Toulouse, S., and Calvo, R. W. A new effective unified model for solving the pre-marshalling and block relocation problems. *European Journal of Operational Research* 271.1 (2018), pp. 40–56.

[27] Dekker, R., Voogd, P., and Asperen, E. van. Advanced methods for container stacking. *OR Spectrum* 28.4 (2006), pp. 563–586.

[28] Delgado, A., Jensen, R. M., Janstrup, K., Rose, T. H., and Andersen, K. H. A constraint programming model for fast optimal stowage of container vessel bays. *European Journal of Operational Research* 220.1 (2012), pp. 251–261.

[29] Dokka, T., Goerigk, M., and Roy, R. Mixed uncertainty sets for robust combinatorial optimization. *Optimization Letters* 14.6 (2019), pp. 1323–1337.

[30] Eglese, R. W. Simulated annealing: a tool for operational research. *European Journal of Operational Research* 46.3 (1990), pp. 271–281.

[31] ElWakil, M., Eltawil, A., and Gheith, M. On the integration of the parallel stack loading problem with the block relocation problem. *Computers & Operations Research* 138 (2022), p. 105609.

[32] Even, S. and Kariv, O. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. *16th Annual Symposium on Foundations of Computer Science*. IEEE. 1975, pp. 100–112.

[33] Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, J. M. A domain-specific knowledge-based heuristic for the blocks relocation problem. *Advanced Engineering Informatics* 28.4 (2014), pp. 327–343.

[34] Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, J. M. An exact approach for the blocks relocation problem. *Expert Systems with Applications* 42.17 (2015), pp. 6408–6422.

[35] Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, M. Pre-marshalling problem: heuristic solution method and instances generator. *Expert Systems with Applications* 39.9 (2012), pp. 8337–8349.

[36] Fechter, J., Beham, A., Wagner, S., and Affenzeller, M. Modeling a lot-aware slab stack shuffling problem. *Computer Aided Systems Theory – EUROCAST 2015*. Springer International Publishing, 2015, pp. 334–341.

[37] Feillet, D., Parragh, S. N., and Tricoire, F. A local-search based heuristic for the unrestricted block relocation problem. *Computers & Operations Research* 108 (2019), pp. 44–56.

[38] Fernandes, E., Freire, L., Passos, A., and Street, A. Solving the non-linear slab stack shuffling problem using linear binary integer programming. *EngOpt 2012 – 3rd International Conference on Engineering Optimization*. Ed. by J. Herskovits. 2012.

[39] Forster, F. and Bortfeldt, A. A tree search procedure for the container relocation problem. *Computers & Operations Research* 39.2 (2012), pp. 299–309.

[40] Fredman, M. L. On computing the length of longest increasing subsequences. *Discrete Mathematics* 11.1 (1975), pp. 29–35.

[41] Gabrel, V., Murat, C., and Thiele, A. Recent advances in robust optimization: an overview. *European Journal of Operational Research* 235.3 (2014), pp. 471–483.

[42] Galle, V., Manshadi, V. H., Boroujeni, S. B., Barnhart, C., and Jaillet, P. The stochastic container relocation problem. *Transportation Science* 52.5 (2018), pp. 1035–1058.

[43] Galle, V., Barnhart, C., and Jaillet, P. A new binary formulation of the restricted container relocation problem based on a binary encoding of configurations. *European Journal of Operational Research* 267.2 (2018), pp. 467–477.

[44] Ge, P., Meng, Y., Liu, J., Tang, L., and Zhao, R. Logistics optimisation of slab pre-marshalling problem in steel industry. *International Journal of Production Research* 58.13 (2019), pp. 4050–4070.

[45] Ge, P., Zhao, R., Sun, D., and Dong, Y. Integrated optimisation of storage and pre-marshalling moves in a slab warehouse. *International Journal of Production Research* 60.6 (2021), pp. 1–23.

[46] Gheith, M. S., Eltawil, A. B., and Harraz, N. A. A rule-based heuristic procedure for the container pre-marshalling problem. *2014 IEEE International Conference on Industrial Engineering and Engineering Management*. 2014, pp. 662–666.

[47] Gheith, M., Eltawil, A. B., and Harraz, N. A. Solving the container pre-marshalling problem using variable length genetic algorithms. *Engineering Optimization* 48.4 (2016), pp. 687–705.

[48] Goerigk, M. and Schöbel, A. Algorithm engineering in robust optimization. *Algorithm Engineering: Selected Results and Surveys*. Ed. by L. Kliemann and P. Sanders. Vol. 9220. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2016, pp. 245–279.

[49] Hottung, A., Tanaka, S., and Tierney, K. Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research* 113 (2020), p. 104781.

[50] Hottung, A. and Tierney, K. A biased random-key genetic algorithm for the container pre-marshalling problem. *Computers & Operations Research* 75 (2016), pp. 83–102.

[51] Huang, S.-H. and Lin, T.-H. Heuristic algorithms for container pre-marshalling problems. *Computers & Industrial Engineering* 62.1 (2012), pp. 13–20.

[52] Inaoka, Y. and Tanaka, S. A branch-and-bound algorithm for the block relocation problem to minimize total crane operation time. *19th International Conference on Harbor, Maritime and Multimodal Logistics Modeling and Simulation (HMS 2017)*. Ed. by E. Bottani, Y. Merkuryev, A. G. Bruzzone, M. A. Piera, and F. Longo. CAL-TEK S.r.l., 2017, pp. 98–104.

[53] Inaoka, Y. and Tanaka, S. The block relocation problem under a realistic model of crane trajectories. *20th International Conference on Harbor, Maritime and Multimodal Logistics Modeling and Simulation (HMS 2018)*. Ed. by E. Bottani, Y. Merkuryev, A. G. Bruzzone, M. A. Piera, and F. Longo. CAL-TEK S.r.l., 2018, pp. 62–66.

[54] International Chamber of Shipping. *Shipping and world trade: top containership operators*. `https://www.ics-shipping.org/shipping-fact/shipping-and-world-trade-top-containership-operators/`. Accessed: 2022-05-30. 2020.

[55] International Chamber of Shipping. *Shipping and world trade: top containership operators*. `https://www.ics-shipping.org/shipping-fact/shipping-and-world-trade-driving-prosperity/`. Accessed: 2022-05-30. 2020.

[56] Jansen, K. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation* 180.2 (2003), pp. 71–81.

[57] Jin, B., Zhu, W., and Lim, A. Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research* 240.3 (2015), pp. 837–847.

[58] Jovanovic, R., Tuba, M., and Voß, S. A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research* 25.1 (2015), pp. 1–28.

[59] Jovanovic, R., Tuba, M., and Voß, S. An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research* 274.1 (2019), pp. 78–90.

[60] Jovanovic, R. and Voß, S. A chain heuristic for the blocks relocation problem. *Computers & Industrial Engineering* 75 (2014), pp. 79–86.

[61] Kang, J., Ryu, K. R., and Kim, K. H. Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent Manufacturing* 17.4 (2006), pp. 399–410.

[62] Kaufman, L. and Broeckx, F. An algorithm for the quadratic assignment problem using bender's decomposition. *European Journal of Operational Research* 2.3 (1978), pp. 207–211.

[63] Kendall, M. G. A new measure of rank correlation. *Biometrika* 30.1/2 (1938), pp. 81–93.

[64] Kim, K. H. and Hong, G.-P. A heuristic rule for relocating blocks. *Computers & Operations Research* 33.4 (2006), pp. 940–954.

[65] Kim, K. H., Park, Y. M., and Ryu, K. R. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research* 124.1 (2000), pp. 89–101.

[66] Knuth, D. E. *The Art of Computer Programming: Fascicle 2: Generating All Tuples and Permutations*. Addison-Wesley, 2005.

[67] Knuth, D. E. *The Art of Computer Programming: Sorting and Searching*. Vol. 3. Addison-Wesley, 1998.

[68] Ku, D. and Arthanari, T. S. Container relocation problem with time windows for container departure. *European Journal of Operational Research* 252.3 (2016), pp. 1031–1039.

[69] Le, X. T. and Knust, S. MIP-based approaches for robust storage loading problems with stacking constraints. *Computers & Operations Research* 78 (2017), pp. 138–153.

[70] Lee, Y. and Hsu, N.-Y. An optimization model for the container pre-marshalling problem. *Computers & Operations Research* 34.11 (2007), pp. 3295–3313.

[71] Lee, Y. and Lee, Y.-J. A heuristic for retrieving containers from a yard. *Computers & Operations Research* 37.6 (2010), pp. 1139–1147.

[72] Lee, Y. and Chao, S.-L. A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research* 196.2 (2009), pp. 468–475.

[73] Lehnfeld, J. and Knust, S. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research* 239.2 (2014), pp. 297–312.

[74] Lersteau, C., Nguyen, T. T., Le, T. T., Nguyen, H. N., and Shen, W. Solving the problem of stacking goods: mathematical model, heuristics and a case study in container stacking in ports. *IEEE Access* 9 (2021), pp. 25330–25343.

[75] Li, T., Luan, Z., Wang, B., and Dong, G. Estimation of distribution algorithm for solving the slab stack shuffling and relocation problem. *Xitong Gongcheng Lilun yu Shijian/System Engineering Theory and Practice* 37 (2017), pp. 2955–2964.

[76] Lin, D.-Y., Lee, Y.-J., and Lee, Y. The container retrieval problem with respect to relocation. *Transportation Research Part C: Emerging Technologies* 52 (2015), pp. 132–143.

[77] López-Plata, I., Expósito-Izquierdo, C., Lalla-Ruiz, E., Melián-Batista, B., and Moreno-Vega, J. M. Minimizing the waiting times of block retrieval operations in stacking facilities. *Computers & Industrial Engineering* 103 (2017), pp. 70–84.

[78] López-Plata, I., Expósito-Izquierdo, C., and Moreno-Vega, J. M. Minimizing the operating cost of block retrieval operations in stacking facilities. *Computers & Industrial Engineering* 136 (2019), pp. 436–452.

[79] Lu, C., Zeng, B., and Liu, S. A study on the block relocation problem: lower bound derivations and strong formulations. *IEEE Transactions on Automation Science and Engineering* 17.4 (2020), pp. 1829–1853.

[80] Maniezzo, V., Boschetti, M. A., and Gutjahr, W. J. Stochastic premarshalling of block stacking warehouses. *Omega* 102 (2021), p. 102336.

[81] Martí, R. and Reinelt, G. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Vol. 175. Springer Science & Business Media, 2011.

[82] Melo da Silva, M. de, Erdoğan, G., Battarra, M., and Strusevich, V. The block retrieval problem. *European Journal of Operational Research* 265.3 (2018), pp. 931–950.

[83] Oelschlägel, T. and Knust, S. Solution approaches for storage loading problems with stacking constraints. *Computers & Operations Research* 127 (2021), p. 105142.

[84] Parreño-Torres, C., Alvarez-Valdes, R., and Ruiz, R. Integer programming models for the pre-marshalling problem. *European Journal of Operational Research* 274.1 (2019), pp. 142–154.

[85] Petering, M. E. H. and Hussein, M. I. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research* 231.1 (2013), pp. 120–130.

[86] Prandtstetter, M. A Dynamic Programming Based Branch-and-Bound Algorithm for the Container Pre-marshalling Problem. PhD thesis. Austrian Institute of Technology, 2013.

[87] Quispe, K. E. Y., Lintzmayer, C. N., and Xavier, E. C. An exact algorithm for the blocks relocation problem with new lower bounds. *Computers & Operations Research* 99 (2018), pp. 206–217.

[88] Ren, H. and Tang, L. Modeling and an ILP-based algorithm framework for the slab stack shuffling problem considering crane scheduling. *2010 International Conference on Computing, Control and Industrial Engineering*. Vol. 2. 2010, pp. 3–6.

[89] Rendl, A. and Prandtstetter, M. Constraint models for the container pre-marshaling problem. *International Workshop on Constraint Modelling and Reformulation*. 2013, pp. 44–56.

[90] Scholl, J., Boywitz, D., and Boysen, N. On the quality of simple measures predicting block relocations in container yards. *International Journal of Production Research* 56.1–2 (2018), pp. 60–71.

[91] Shi, Y. and Liu, S. Very large-scale neighborhood search for steel hot rolling scheduling problem with slab stack shuffling considerations. *IEEE Access* 9 (2021), pp. 47856–47863.

[92] Silva Firmino, A. da, Abreu Silva, R. M. de, and Times, V. C. A reactive GRASP metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics* 25.2 (2018), pp. 141–173.

[93] Silva Firmino, A. da and Times, V. C. A coronavirus optimization algorithm for solving the container retrieval problem. *Frontiers in Nature-Inspired Industrial Optimization*. Ed. by M. Khosravy, N. Gupta, and N. Patel. Springer Tracts in Nature-Inspired Computing. Springer Singapore, 2021, pp. 49–76.

[94] Silva Firmino, A. da, Times, V. C., and Abreu Silva, R. M. de. Optimizing the crane's operating time with the ant colony optimization and pilot method metaheuristics. *Frontier Applications of Nature Inspired Computation*. Ed. by M. Khosravy, N. Gupta, N. Patel, and T. Senjyu. Springer Singapore, 2020, pp. 364–389.

[95] Singh, K. A., Srinivas, and Tiwari, M. Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved parallel genetic algorithms. *International Journal of Production Economics* 91.2 (2004), pp. 135–147.

[96] Tanaka, S. and Mizuno, F. An exact algorithm for the unrestricted block relocation problem. *Computers & Operations Research* 95 (2018), pp. 12–31.

[97] Tanaka, S. and Tierney, K. Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. *European Journal of Operational Research* 264.1 (2018), pp. 165–180.

[98] Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., and Ruiz, R. A branch and bound approach for large pre-marshalling problems. *European Journal of Operational Research* 278.1 (2019), pp. 211–225.

[99] Tanaka, S. and Takii, K. A faster branch-and-bound algorithm for the block relocation problem. *IEEE Transactions on Automation Science and Engineering* 13.1 (2016), pp. 181–190.

[100] Tanaka, S. and Voß, S. An exact approach to the restricted block relocation problem based on a new integer programming formulation. *European Journal of Operational Research* 296.2 (2022), pp. 485–503.

[101] Tang, L., Liu, J., Rong, A., and Yang, Z. An effective heuristic algorithm to minimise stack shuffles in selecting steel slabs from the slab yard for heating and rolling. *Journal of the Operational Research Society* 52.10 (2001), pp. 1091–1097.

[102] Tang, L., Liu, J., Rong, A., and Yang, Z. Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules. *International Journal of Production Research* 40.7 (2002), pp. 1583–1595.

[103] Tang, L. and Ren, H. Modelling and a segmented dynamic programming-based heuristic approach for the slab stack shuffling problem. *Computers & Operations Research* 37.2 (2010), pp. 368–375.

[104] Tierney, K., Pacino, D., and Voß, S. Solving the pre-marshalling problem to optimality with A* and IDA*. *Flexible Services and Manufacturing Journal* 29.2 (2017), pp. 223–259.

[105] Tierney, K. and Voß, S. Solving the robust container pre-marshalling problem. *International Conference on Computational Logistics*. Ed. by A. Paias, M. Ruthmair, and S. Voß. Springer International Publishing, 2016, pp. 131–145.

[106] Ting, C.-J. and Wu, K.-C. Optimizing container relocation operations at container yards with beam search. *Transportation Research Part E: Logistics and Transportation Review* 103 (2017), pp. 17–31.

[107] Tricoire, F., Scagnetti, J., and Beham, A. New insights on the block relocation problem. *Computers & Operations Research* 89 (2018), pp. 127–139.

[108] UNCTAD. *Review of Maritime Transport 2020*. Tech. rep. United Nations Conference on Trade and Development, 2020.

[109] Ünlüyurt, T. and Aydin, C. Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation* 46.4 (2012), pp. 378–393.

[110] Voß, S. Extended mis-overlay calculation for pre-marshalling containers. *Computational Logistics*. Ed. by H. Hu, X. Shi, R. Stahlbock, and S. Voß. Vol. 7555. Lecture Notes in Computer Science. Springer, 2012, pp. 86–91.

[111] Voß, S. and Schwarze, S. A note on alternative objectives for the blocks relocation problem. *Computational Logistics*. Ed. by G. Goos and J. Hartmanis. Vol. 11756. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 101–121.

[112] Wan, Y.-W., Liu, J., and Tsai, P.-C. The assignment of storage locations to containers for a container stack. *Naval Research Logistics* 56.8 (2009), pp. 699–713.

[113] Wang, N. Optimization Study on Container Operations in Maritime Transportation. PhD thesis. City University of Hongkong, 2014.

[114] Wang, N., Jin, B., and Lim, A. Target-guided algorithms for the container pre-marshalling problem. *Omega* 53 (2015), pp. 67–77.

[115] Wang, N., Jin, B., Zhang, Z., and Lim, A. A feasibility-based heuristic for the container pre-marshalling problem. *European Journal of Operational Research* 256.1 (2017), pp. 90–101.

[116] Winter, T. and Zimmermann, U. T. Real-time dispatch of trams in storage yards. *Annals of Operations Research* 96.1 (2000), pp. 287–315.

[117] Zehendner, E., Caserta, M., Feillet, D., Schwarze, S., and Voß, S. An improved mathematical formulation for the blocks relocation problem. *European Journal of Operational Research* 245.2 (2015), pp. 415–422.

[118] Zehendner, E. and Feillet, D. A branch and price approach for the container relocation problem. *International Journal of Production Research* 52.24 (2014), pp. 7159–7176.

[119] Zhang, C., Guan, H., Yuan, Y., Chen, W., and Wu, T. Machine learning-driven algorithms for the container relocation problem. *Transportation Research Part B: Methodological* 139 (2020), pp. 102–131.

[120] Zhang, C., Wu, T., Kim, K. H., and Miao, L. Conservative allocation models for outbound containers in container terminals. *European Journal of Operational Research* 238.1 (2014), pp. 155–165.

[121] Zhang, H., Guo, S., Zhu, W., Lim, A., and Cheang, B. An investigation of IDA* algorithms for the container relocation problem. *Trends in Applied Intelligent Systems*. Ed. by N. García-Pedrajas, F. Herrera, C. Fyfe, J. Manuel, and B. M. Ali. Vol. 6096. Lecture Notes in Artificial Intelligence. Springer Berlin Heidelberg, 2010, pp. 31–40.

[122] Zhang, R., Jiang, Z.-Z., and Yun, W. Stack pre-marshalling problem: a heuristic-guided branch-and-bound algorithm. *International Journal of Industrial Engineering: Theory, Applications, and Practice* 22.5 (2015).

[123] Zhang, R., Liu, S., and Kopfer, H. Tree search procedures for the blocks relocation problem with batch moves. *Flexible Services and Manufacturing Journal* 28.3 (2015), pp. 397–424.

[124] Zhu, W., Qin, H., Lim, A., and Zhang, H. Iterative deepening A* algorithms for the container relocation problem. *IEEE Transactions on Automation Science and Engineering* 9.4 (2012), pp. 710–722.

[125] Zweers, B. G., Bhulai, S., and Mei, R. D. van der. Optimizing pre-processing and relocation moves in the stochastic container relocation problem. *European Journal of Operational Research* 283.3 (2020), pp. 954–971.

[126] Zweers, B. G., Bhulai, S., and Mei, R. D. van der. Pre-processing a container yard under limited available time. *Computers & Operations Research* 123 (2020), p. 105045.

Acknowledgements

Above all, I would like to thank my supervisor Sigrid Knust, who always supported me in every respect and at all times. During my time at the university, she not only shared her wealth of knowledge with me, she also always attached importance to conveying her working methods and working techniques to me. Moreover, I am glad that she always supported the attendance at interesting further training courses, conferences and working groups.

I would also like to thank Benjamin Graf, with whom I had many discussions during our joint doctoral studies. I'm glad that we shared an office and were able to support each other with many small questions, but also with difficult problems. Furthermore, I thank Jan-Niklas Buchkow and Tobias Oelschlägel, I had some interesting and important discussions with both of them and they were always at my side with help and advice. We were all a great team together and my time at the university would only have been half as good without you. Many thanks for everything!

Moreover, I thank Friedhelm Hofmeyer for his support in technical questions and administration, whatever the problem was, I (like everyone else in the institute) could 100% rely on him for finding a solution.

For proofreading drafts I thank Bejamin Graf and Jan-Niklas Buckow.

I would also like to thank all my scientific colleagues for a stimulating exchange during my doctoral time, including in particular my colleagues from the "Graduiertenkolleg Graphs and Networks", all anonymous reviewers of the research articles, that are the basis of this doctoral thesis, as well as Simon Emde and Nils Boysen for providing me their test data and results for the PSLP and Thomas Winter for providing me his test data and results for the BRPIF.

Last but not least, I want to point out the role of my wife Nicole and my daughter Marie, who have been an important source of inspiration for me throughout this time and have probably influenced this work more than they realize. Thank you, I dedicate this work to you both!