Sorting Algorithms for Autonomously Organized Block Stacking Warehouses

To attain the academic title of
Dr.-Ing.

Dissertation approved by
the Faculty of Mechanical Engineering
TU Dortmund University

Jakob Clemens Pfrommer, M. Sc.
from
Sindelfingen, Germany

Dortmund, May 2023

Day of the oral examination: 26.10.2023
First Supervisor: Prof. Dr. Dr. h. c. Michael Henke
Second Supervisor: Prof. Dr. Kevin Tierney
Funded by the German Research Foundation – project number 276879186 (GRK2193/2)

Preface and Acknowledgments

Before starting my research career in 2019 and writing this thesis at the chair for enterprise logistics at TU Dortmund, I worked for almost five years on supply chain topics at WEPA, a hygienic paper production company. One of their main challenges was dealing with the high fluctuations in demand and the associated evolving and constantly changing need for storage capacities. There, the main type of storage used was block stacking storage, which became the focus of my research. Therefore, my first thanks go to my former employer WEPA and the wonderful people I met there. Altogether, they have been a great source of inspiration on challenges in logistics and supply chain management.

Besides inspiration, the most important step was to gain the opportunity to work on this topic, as well as the support, guidance, and encouragement needed for the long research journey. For this, I extend my deep thanks to my supervisor Prof. Dr. Dr. h. c. Michael Henke. A very special thanks also goes to Prof. Dr. Anne Meyer, who not only guided me throughout the entire research period but also helped me to improve and develop as a scholar. Her ideas, insights, and expertise have been instrumental in shaping the direction of my research. Also, I want to thank my current and former colleagues as well as the co-authors, student assistants, interns, graduates and everyone else who supported and worked together with me.

Most of all, I want to thank Melina for walking this path with me!

Dortmund, May 2023 *Jakob Clemens Pfrommer*

Abstract

Block stacking storage systems are a very simple and widely used type of warehouse, where unit loads like pallets, boxes, containers, or movable shelves are placed on the ground and may be stacked atop one another. The advantages of block storage are that it is highly flexible, easily scalable, and mobile, without requiring large investments in infrastructure. Autonomous mobile robots (AMRs) are increasingly used to automate operations in such storage systems. However, even though automated block stacking warehouses can be very efficient, they still lack characteristics such as the adaptability and creativity of human operators.

This thesis introduces a vision of autonomously organized block stacking storage systems that integrate the advantageous characteristics of both automated and manually operated warehouses. We describe the related major operational decision problems, but because it would exceed the scope of this paper to consider all such decision problems, we select one of the most interesting, namely the unit-load pre-marshalling problem (UPMP), and study it in depth. The UPMP deals with sorting such a system in off-peak hours to prepare for future orders and hence allows making use of the constant availability of AMRs. The goal is to sequence all stored unit loads in ascending order based on the retrieval priority group of each unit load. A similar problem is the container pre-marshalling problem in maritime transportation, where each stack is accessible from the top via a crane. In the case of the UPMP, a bay (storage area consisting of multiple columns and rows of stacks) is accessed by AMRs from the side, from as many as four access directions. We present a novel two-step solution approach that is able to consider multiple access directions. In the first step we determine the access direction to each stack based on a network flow model, which allows us in the second step to adapt an existing optimal tree-search procedure and lower bound heuristics from container pre-marshalling.

Our solution approach is first tested on different sizes of single-bay experiments. In a second step, we extend and improve our approach and demonstrate its practicality on larger multi-bay block stacking warehouses containing multiple columns and rows of bays. Both experiments on randomly generated problem instances show that multiple access directions greatly improve the access times to storage locations and the overall sorting effort. We outline the need for a future long-term research program related to pre-marshalling and other interrelated decision problems to deal with the associated challenges in block stacking storage systems.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

ABSWP	autonomous block stacking warehouse problem
AGV	automated guided vehicle
AMR	autonomous mobile robot
API	application programming interface
AS/RS	automated storage and retrieval system
BB	bad-bad
BBD	best before dates
BFIFO	batch first-in, first-out
BG	bad-good
BX	bad-good and bad-bad
CB	class-based
COI	cube order per index
COL	closest open location
CPMP	container pre-marshalling problem
CPU	Central Processing Unit
CRP	container relocation problem
CSP	coil shuffling problem
DC	dual command
DOS	duration-of-stay
ERP	Enterprise Resource Planning
FCFS	first-come first-serve
FEFO	first-expiry, first-out
FIFO	first-in first-out
FMCG	fast-moving consumer good

GB	good-bad
GG	good-good
GTIN	global trade item number
GX	good-good and good-bad
I-point	input-point
I/O-point	input/output-point
II	item information
IT	information technology
LIDAR	light imaging, detection and ranging
LIFO	last-in first-out
LWR	longest waiting retrieval
MC	multi command
MDP	markov decision process
MIP	mixed-integer programming
MUPMP	multibay unit-load pre-marshalling problem
NI	no information
NN	nearest-neighbor
NR	nearest retrieval
NSR	nearest storage/retrieval
O-point	output-point
PI	product information
RAM	random-access memory
RB	random-based
RFID	radio-frequency identification
RMFS	robotic mobile fulfillment system
RMP	remarshalling problem
SC	single command
SKU	stock keeping unit
SL	shortest leg
SLAM	simultaneous localization and mapping
SLAP	storage location assignment problem
TOB	turnover-based
TSP	traveling salesperson problem
UPMP	unit-load pre-marshalling problem
VRP	vehicle routing problem
WMS	Warehouse Management System

Chapter 1
Introduction

1.1 Motivation

In our constantly evolving world, a decisive competitive factor of factories, organizations and entire value chains is the capability to adapt to a variety of influencing factors, such as changes in markets and technologies. Companies must react to shifting material flows in their supply chain networks caused by evolving markets on both the demand and supply sides. Supply chains that move agilely and can adapt quickly to new circumstances, while aligning all interests, will have the advantage over their competition [Lee+04].

Storage systems are an important element of a supply chain and can usually be found at many points of a network because they enable a time offset for the in- and outbound flow of goods. Thus, efficient and adaptable warehouses, in which goods can be stored until they are needed, are vital.

Examples abound for the benefits of goods storage, such as quantity balancing to decouple production processes, safety stock to bridge any disruptions, speculation to exploit market price differences, and acquisition to ensure delivery capability (see [GF08, pp. 142] [TSD18, pp. 52], [FK19, pp. 161], [Tem05, pp. 1, pp. 48]).

An ever-present challenge of storage systems is matching required and available storage capacity. No matter how accurately the required capacity was estimated at the time a storage space was established, odds are high that after a certain time it will no longer be appropriate. For instance, demand growth or technological investments in production machines lead to the requirement of additional storage space. Shrinking demand, on the other hand, would lead to spare capacity and unused empty storage space that could be reallocated to fulfill other purposes.

Many forms of technical warehouse infrastructure, like automated high-bay warehouses, require large investments and long setup times. Therefore, the logistic space is ideally empty [TH14]. Block stacking storage systems are highly flexible storage systems capable of instantly scaling warehouse

capacity. As the simplest form of storage system, unit loads (e.g., pallets, containers, boxes) are placed on the floor and may be stacked on top of one another. They require no infrastructure besides transportation vehicles (e.g., forklifts). Due to their simplicity, flexibility, easy setup, and low investment costs, block stacking storage systems are particularly relevant in practice and are used in a wide range of applications, including buffer storage for the material supply of production systems, finished goods warehouses, and cross-docking.

Today, most block stacking storage systems are organized manually by human operators (e.g., forklift drivers), which is labor-intensive and accounts for 75% of their operating costs [Mar16, p. 285]. Even though the spontaneity and problem-solving skills of humans (e.g., layout adjustments when storage is full) are hard to replicate via automated solutions (see [Böv+19], nonautomated systems bring a variety of problems and challenges. Firstly, warehouse employees expose themselves to great risks. The U.S. Bureau of Labor Statistics collects data on reported fatal and nonfatal occupational injuries involving forklifts in the United States. The statistics from 2011 to 2020 show that each year, approximately 71 people die and 7242 are injured, leading to an average of 16 days away from work [US 23]. Secondly, logistics personnel are hard to find (see, e.g., shortage of skilled workers in Germany [IFO22]). The main reasons for this, according to a survey by the German Logistics Association (Bundesvereinigung Logistik), are poor working conditions and low pay [Bun18].

Thirdly, human operators typically organize a block stacking warehouse by grouping goods into lanes and blocks, whereby floor space is divided into storage areas (so-called bays) and pathways for traveling (so-called aisles and orthogonal cross-aisles) to allow fast access and travel times. Humans require simple, well-defined structures such as fixed warehouse layouts with zones, or the ABC classification of products, to make storage location assignment decisions. However, not all human workers adhere to the rules and keep the warehouse clean. Such irregularities lead to search times and require daily stocktaking. High-performance work depends not only on the skillset but also on other factors such as motivation or mood.

On the other, hand, automated block stacking storage systems based on robotic forklifts require huge investments and long setup times. For those reasons, such systems have been implemented mainly by larger corporations in scenarios with multiple work shifts. The interrelated operational decision problems are solved separately in a decision pipeline of hierarchically connected systems (e.g., Warehouse Management System (WMS) and fleet management), which leads to sub-optimal decisions. In addition, the potential flexibility of block stacking warehouses is not fully utilized. Storage strategies and optimization objectives are often not adapted, even though the material flows are changing over time, leading to varying warehouse

and forklift utilization rates. Furthermore, the internal layouts are not automatically adjusted, and off-peak hours are rarely used for rearrangement operations. Today's automated systems are simply not as flexible, adaptable, or easy to set up and operate as they could be. Hence, a new generation of automated systems—so-called autonomous block stacking storage systems—is needed that incorporates key advantage characteristics of humans like creativity, adaptability, learning ability, self-governance, and curiosity.

1.2 Research objectives and scientific contribution

The overall scope and contribution of this thesis is threefold. We aim to (1) provide a vision for autonomous block stacking storage systems, (2) determine and investigate the major related operational decision problems, and (3) focus in depth on one of the most interesting problems, given that the topic is far too broad to cover here in full.

The initial goal is to provide a vision of autonomous block stacking warehouses and the associated desired storage system characteristics. Before defining a target state, it is first necessary to understand exemplary areas of application together with companies' respective practical requirements. Thus, we aim to grasp the differences between various block stacking storage application scenarios. Furthermore, it is necessary to elaborate on areas where human operators excel over automated systems. This is, on the one hand, a fundamental step toward identifying needs and fields of action; on the other hand, it is also an important cornerstone to gain a better understanding of desirable properties when elaborating a vision for autonomously organized block stacking storage systems. Therefore, the first contribution is:

Contribution 1 – Vision:

- ***A vision for autonomously organized block stacking storage systems.***

In storage systems, a large number of decision problems must be solved, and an autonomous system requires identifying and modeling these decisions. Consequently, the second research goal is to determine the decision problems within the scope of a block stacking warehouse. Based on these findings, the perimeters of this work are set and the considered decisions are described and mapped to existing decision problems in research. Moreover, we introduce the autonomous block stacking warehouse problem (ABSWP) and discuss major challenges. The highlights of the second contribution are:

Contribution 2 – Decision problems (based on Pfrommer and Meyer [PM20]):

- ***The description of the major operational decision problems in autonomously organized block stacking warehouses.***
- ***A review of related literature for each decision problem.***
- ***The introduction of the overarching ABSWP.***
- ***An overview of key challenges.***

In reference to the first two contributions, the next research goal is to increase the level of detail by selecting one of the decision problems and developing sophisticated approaches to resolving it.

We identify relocation and pre-marshalling as one of the most interesting research topics, because sorting and rearrangement of items in off-peak hours is a special feature of autonomous systems.

autonomous mobile robots (AMRs) are not only constantly available but also allow the use of a more complex and fully shared storage strategy[1] that enables highly dense block storage systems improving the space utilization in warehouses. Therefore, we build on a solution procedure from the CPMP and transfer it to the UPMP. An additional desired property in block stacking storage systems is that the solution approach be suitable for handling up to four access directions to a storage bay and several tiers of stacking.

Furthermore, the developed solutions should allow the possibility to quickly evaluate the state of a storage system (e.g., in terms of blockage). This can be directly used to tackle other decision problems, such as the assignment of items to storage locations. The third contribution itself entails two sets of sub-contributions, of which the first are:

Contribution Part 3.1 – Sorting block stacking storage systems in off-peak hours (based on Pfrommer, Meyer, and Tierney [PMT22] (submitted)):

- ***The formal definition of the novel UPMP for sorting a block stacking storage system with multiple access directions.***
- ***The introduction of a novel two-step solution approach based on a network flow model and MIP formulation to calculate the minimum number of moves to sort a storage system with up to four access directions.***
- ***A publicly available benchmark dataset.***
- ***The evaluation and detailed analysis of the proposed solution approach in computational experiments.***

[1] A fully shared storage strategy means that every storage location is available for any item. Thus, no storage space is reserved for only certain products.

We then extend the scope of the UPMP to larger warehouses consisting of multiple bays. Increasing the warehouse size greatly extends the possible scope of application, but it also leads to additional challenges, like an enormous solution space. Hence, the goal is to conduct several adaptions and improvements to our solution approach for the UPMP so that we are able to solve large multibay warehouse instances. Additionally, an overview of related sorting problems and solution approaches in multibay warehouse setups should be provided. Finally, the capabilities and utility of the developed solution approach must be validated and supported. The second set of sub-contributions for our third contribution are:

Contribution Part 3.2 – Sorting multibay warehouses:

- ***A formal definition of the multibay UPMP for sorting multibay block stacking storage systems with multiple access directions.***
- ***An adapted and improved version of our two-step solution approach that is able to solve large multibay UPMP instances with more than 1000 storage locations.***
- ***A large, publicly available benchmark dataset.***
- ***Computational experiments for the detailed analysis of the extended solution approach and a comparison of different warehouse setups.***

Finally, we also contribute with our outlook and future research program that identifies ample opportunities for future research.

1.3 Outline

In this section, we present the structure and main research activities of the dissertation project that are shown in Fig. 1.1. This work is divided into six main chapters based on the previously defined research contribution clusters, complemented by an introduction and conclusion. Although we generally recommend that this thesis be read in chronological order, the third chapter, as well as the fourth and fifth chapters together, are self-contained tracks based essentially on two publications: [PM20]; [PMT22]

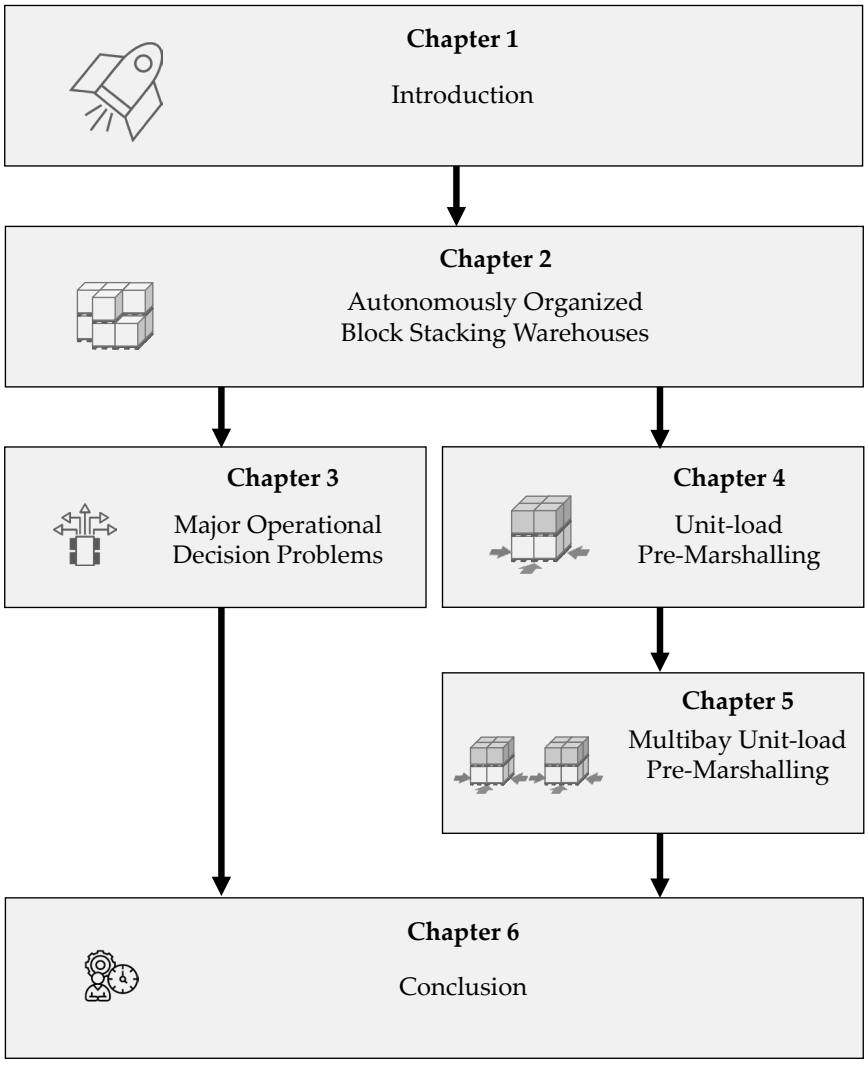


Fig. 1.1 General approach and structure of the thesis.

After the introduction here in Chapter 1, Chapter 2 commences with the fundamentals of block stacking storage systems, including a description of the main components and the basic process flow. We also provide an overview of areas of application and, by way of examples, introduce four real use cases and the associated practical requirements. This is followed by a brief introduction to the terminology of autonomous systems in the industrial and engineering contexts. Finally, we describe the characteristics of automated and nonautomated systems and present our vision for autonomously organized block stacking warehouses.

In Chapter 3, which is based on [PM20], we give an overview and provide a classification of decision problems in autonomous block stacking storage systems. Starting with Section 3.2, we formally define the considered operational decisions and provide a literature review for each problem, mapping them onto existing decision problems in the research literature. Based on the defined decision problems, we introduce the overarching ABSWP and discuss the major challenges and needs of future research.

In Chapter 4 and Chapter 5, we tackle one of these challenges—namely the UPMP to sort block stacking storage systems during off-peak hours—and solve it in single and multibay setups. Chapter 4, based on [PMT22] (submitted, under review), is about the UPMP for sorting a single-bay storage system during off-peak hours under a shared storage strategy based on the retrieval sequence of items. We first formally introduce the UPMP and provide an extended literature review of related research. In a first step, we transfer solution approaches based on exact tree search methods from container pre-marshalling to the UPMP. We show that a direct transfer is possible for the simplest case with one access direction. For the cases with multiple access directions, we present a novel two-step approach based on a network flow model and a MIP formulation. Subsequently, our solution approach is validated in computational experiments on a large dataset of randomly generated UPMP instances. Finally, we indicate further areas of applications and potential avenues for future research.

Chapter 5 deals with the multibay UPMP. Similarly to the previous chapter, here we formally introduce the problem and present related work. In the next step, the adaptions and additions of our two-step solution approach are presented before we carry out extensive computational experiments. The experiments are based on several datasets consisting of randomly generated instances. A detailed analysis is followed by a description of the main insights and the delivery of final remarks.

In our overall conclusion and outlook in Chapter 6, we provide a summary of the work and discuss the implication of autonomous block stacking storage systems for supply chain networks. To complete the thesis, we offer some ideas of new concepts and propose potential pathways for future research to build upon this work.

Chapter 2
Autonomously organized block stacking warehouses

This chapter starts by presenting the most important elements of a block stacking storage system, providing an explanation of the basic process flow, and enumerating exemplary areas of applications as well as real use cases. Subsequently, we introduce definitions of autonomous systems, advantageous characteristics of automated and nonautomoated systems, followed by our vision of autonomous block stacking warehouses.

2.1 Fundamentals of block stacking warehouses

Block stacking storage systems are the simplest type of warehouse used in a wide field of applications such as buffering, cross-docking, or finished goods storage. Generally, any type of unit load—such as pallets, boxes, containers or pods (movable racks)—can be stored by simply placing them on the floor, and possibly stacking them on top of one another.

In order to use the available space as efficiently as possible, stackable and standardized units of handling are most suitable. The length and width of a single stack are defined by the geometry of the stored goods, whereas the height of a stack (i.e., number of tiers) depends on the specific material statics (weight limit), the height restrictions of a warehouse building, and the operating technology (e.g., possible lifting height). The stored goods can generally be arranged in any pattern; this pattern immediately affects the system performance in terms of space utilization, travel distances, and accessibility of goods.

Fig. 2.1 illustrates the basic structure of a pallet-based block stacking storage system with a single storage bay and indicated aisles for traveling. A defined standardized rectangular footprint allows the arrangement of these units as a grid-based storage. Each lane of the bay contains a sequence of stacks, where only the outermost stacks can be accessed by transport vehicles. Because we consider only side access from a free direction, blockages may have to be resolved in order to reach a desired unit load.

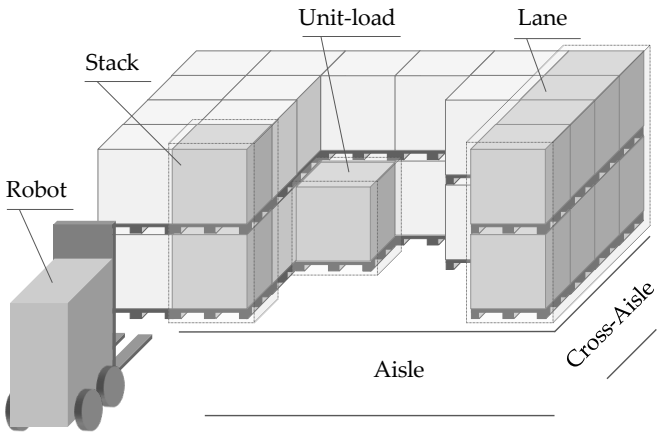


Fig. 2.1 Example of a pallet-based block stacking storage system with a robotic forklift.

2.1.1 Types of transport vehicles

Block stacking storage systems can generally be operated by different kinds of systems, including cranes and vehicles [TH14, pp. 63]. We consider only ground-based transport vehicles that require no additional infrastructure. The eligible types of transport vehicles very much depend on the characteristics of the specific stock keeping units (SKUs) that must be stored. There are many different types, of which two well-known examples are forklifts and pallet jacks.

In terms of automated transport vehicles, automated guided vehicles (AGVs) and AMRs are able to take over operational tasks. To our understanding, AGV is a more general definition including vehicles that can only follow physical guidelines on the floor, such as magnetic strips, whereas AMR is a newer, and increasingly common term for a transport vehicle capable of some autonomous functions (see [Fra+21]; [Hor+21]). Such autonomous functions may be as simple as automatic collision avoidance, but they also include more sophisticated functions, like the capability to move and navigate freely via simultaneous localization and mapping (SLAM) by recognizing and continuously updating the environment using, e.g., light imaging, detection and ranging (LIDAR) sensors or cameras. Because AMRs do not need to follow a predetermined path, this independence of movement opens up new possibilities in warehouse operations (e.g., in terms of additional routing options and enhanced collision avoidance) and makes setup easier and quicker [Ull+15]; [Fra+21, pp. 25, pp. 98].

In Fig. 2.1 and throughout this work, we consider AMRs that belong to the category of robotic forklifts with the capability to move freely within a warehouse.

2.1.2 Types of unit loads

Table 2.1 shows examples of different types of unit loads (see [TH14, pp. 23]). Unit loads can be based on load carriers or loading equipment that can function as a standardized base for handling, and which may enclose the goods to provide extra protection (see [TSD18, pp. 23]). Most load carriers are rectangular and may allow access from one to four cardinal directions (typically at least two opposite access directions).

With approximately five billion pallets in global circulation, the pallet is one of the most important load carriers in logistics [Pla22]. Although usually wooden, pallets may also be made of other materials like plastic (more durable and hygienic) or cardboard (cheap and lightweight, but disposable/one-way). Pallet sizes and dimensions vary across regions such as North America, Europe, Australia, and Asia (see [BH19, pp. 37]). Other types of unit loads include pallet cages, movable shelves, containers, and any type of stackable boxes or materials that can be moved by the transport vehicles described in Section 2.1.1.

2.1.3 Process flow

Fig. 2.2 shows the general process flow of a unit load through a block stacking storage system. Goods arrive at an inbound dock (In), where they wait until they are picked up. In the following step, goods can be either stored in the warehouse or retrieved directly and delivered to an outbound dock (Out), if a retrieval order for this good—a so-called SKU—already exists at the time of arrival. If a good is placed at a storage location in the warehouse, it waits until it is requested by a retrieval order. The optional relocation process allows the position of a stored unit load to be changed, which may be helpful in sorting and re-organizing a warehouse or in reaching an item that is blocked by others.

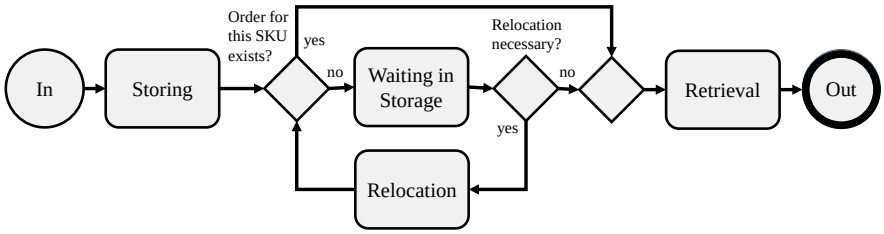


Fig. 2.2 General process flow of a block stacking warehouse.

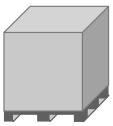
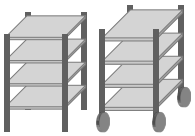
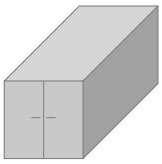
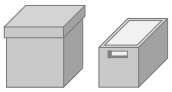
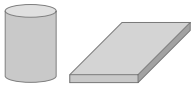
Name	Sketch	Description
Pallet-based unit load		• Load carrier: Standardized pallet • Load examples: Beverage crates, cartons, boxes, secondary packaging • Stackable: Depends on the load characteristics and palletizing scheme
Pallet cage		• Load carrier: Standardized pallet cage • Load examples: Any kind that fits into the pallet cage • Stackable: Generally yes
Movable shelf		• Load carrier: Material-handling trolley or underrunable rack that can be lifted/moved by robots or forklifts • Load examples: Customizable heights of shelves allow to fit in many types of items. • Stackable: Generally no
Container		• Load carrier: standardized 20 or 40 feet container • Load examples: Any item that fits into it. • Stackable: Generally yes
Boxes		• Load carrier: Cartons, Crates or small load carriers • Load examples: Any kind of small parts • Stackable: Generally yes
Materials		• Load carrier: Non. Requires gripping device. • Load examples: Paper reels, steel coils, steel plates. • Stackable: Typically yes

Table 2.1 Examples of different types of unit loads.

2.1.4 Areas of application and examples of real use cases

Due to the system's simplicity, the areas of application for block stacking storage are countless, ranging from small, temporary buffer storage to vast warehouses with thousands of unit loads. Thus, we cannot hope to offer a complete picture that can be generalized to every case; instead, we aim to highlight specific areas of application and key differences. Industries where block stacking storage systems are used extensively include fast-moving consumer goods (FMCG), building materials production, steel production, agricultural production, online and offline commerce, logistics and warehous-

ing service providers, maritime transportation, event logistics, and military production and logistics.

In the following, we provide a description of four application scenarios for block stacking storage systems. Throughout the dissertation project, we gained practical insights from various corporations with industrial partners in the beverage industry, the hygienic paper industry, a manufacturer of technical equipment, and an industrial coating producer. These are only a few of many examples where block stacking storage systems are in use. Dependent on the industry, as well as the usage of block storage within the value stream, storage systems fulfill various purposes: The requirements of each warehouse and their prioritization is use-case-specific.

Beverage industry: In the beverage industry, pallet-based block stacking warehouses are used at various points in the supply chain. Beverage crates on pallets allow stable stacking of multiple tiers for finished goods or empties. In the case of a deposit system, a block storage for crates of empties is often located outdoors, as shown in Fig. 2.3. The block storage for full loads is usually located indoors to avoid any soiling of sales products.

Fig. 2.3 Pallets with crates of empties outside a brewery.

For beverage storage and retrieval, rules based on production batches and best before dates (BBD) are important. If more than one batch is stored in a single lane, sorting and rearrangement to reach and retrieve the right products may be necessary.

Operational rules in the beverage industry are usually not strict (so-called weak storage rules). Usually a weak first-in first-out (FIFO) or FIFO based on production batches is applied for selecting a storage block or lane. Within

a storage lane, pallets are retrieved according to the last-in first-out (LIFO) principle to avoid the relocation of unit loads.

Hygienic paper industry: In the hygienic paper industry as well, block stacking storage systems are used for a variety of purposes. Examples include the storage of raw materials like pulp and waste paper, half-finished goods (so-called jumbo reels) and pallet-based finished goods storage. Whereas raw materials are sometimes stored outdoors, half-finished and finished goods are usually stored indoors to avoid exposure to the elements and to prevent soiled products and quality issues.

The hygienic paper industry and the beverage industry follow the same storage rules, albeit with the difference that hygienic paper has no BBD. However, rules based on production batches are also relevant, because product design and specifications change over time.

Fig. 2.4 Loading zone of a block stacking warehouse for finished goods at a hygienic paper manufacturer.

Stacking hygienic paper products can be trickier because there is usually no crate, box, or other stable packaging. Stability and weight depend on the varying product characteristics. Elastic or overhanging goods on pallets, as well as uneven top surfaces, require individual assessment for storage decisions. Sometimes, when stacking pallets directly atop one another is not stable, a pyramidal stacking pattern, as shown in Fig. 2.5, works very well.

Supermarket for material supply in manufacturing: At the manufacturer of technical equipment, we encountered a buffer storage for the material supply of their production, which allowed them to change the setup and layout of the production and storage system flexibly over time. Fig. 2.6

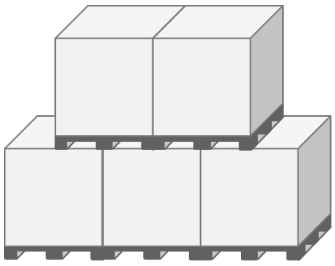


Fig. 2.5 Pyramidal stacking pattern of pallet-based unit loads.

shows a part of the block storage. Because it was necessary to quickly reach stored products, all storage locations were directly accessible. Most of the storage locations were fixed and marked with barcodes so that workers could find the desired products. It is worth remarking on the number of storage locations with empty pallets: Fixed storage locations with empty spots and directly accessible items lead to a large space requirement of the block stacking storage system. Therefore, an obvious requirement is that block stacking storage systems, when used as buffer storage for the material supply of production systems, should use costly available space as efficiently as possible. At the same time, access times must remain acceptable to ensure responsiveness and timely delivery of materials.

Fig. 2.6 Top view on a buffer storage for the material supply of a production system at a manufacturer of technical equipment.

Buffer storage of material-handling trolleys at an industrial coating company: We encountered another example of single-tier block storage at a coating and painting company for automotive parts. They use a huge number of heavy material-handling trolleys to carry the frames, as shown in Fig. 2.7, which the parts are attached to. The trolleys are buffered before going to production and then circulate back with new parts that need to be coated/painted. While in buffer storage, the trolleys are placed in multi-deep storage bays under a shared storage policy, meaning that every storage

location is available for any material-handling trolley. However, this makes organizing the storage quite complicated because each trolley's retrieval time must be considered to avoid blockage. To reach a desired trolley in the back, human operators may need to search for the target trolley or relocate blocking trolleys—a physically demanding task due to the heavy weight of the trolleys. Also, ongoing traffic of different forklifts, movement of people and narrow aisles/bottlenecks must be handled.

Fig. 2.7 Buffer storage with material-handling trolleys at a coating company.

In all industries, we encountered creative and inventive ideas from human operators. Aisle space was used temporarily to speed up a loading or relocation process, or to increase the warehouse capacity in peak times. In another case, special forks were developed to increase efficiency by lifting and carrying more items—an innovative technique that required the forklift operators to drive in reverse.

2.2 Autonomous block stacking warehouses

Based on autonomously controlled decentralized structures, a supply chain can be divided into fractals that lead to a high degree of modularity, compatibility, mobility, scalability, and universality. In a probabilistic world, the main goal is to increase the response speed and flexibility of logistics systems. [TH14] In the following, we present our vision of autonomous block stacking warehouses, which could be one of the key elements to increase the adaptability of future supply networks. Before introducing the vision, we first briefly introduce the concepts of autonomy and autonomous systems.

2.2.1 Definitions of autonomous systems

Overall, the concept of autonomy is complex and multifaceted. The origins of the term "autonomy" go back to an Ancient Greek word that translates into "living by one's own laws" and is a combination of *auto* (self) and *nomos* (law). Today, there exist a variety of definitions depending on the discipline, be it engineering, biology, medicine, politics, or philosophy. [Swa16]; [VTF16] Here we try to give an overview of the most widely accepted definitions of autonomous systems in the industrial and engineering contexts. Further overviews regarding definitions of autonomy can be found in [Che+21]; [HMA15].

Vagia, Transeth, and Fjerdingen [VTF16] review the evolution of taxonomies dealing with the levels of autonomy and/or automation in engineering. In their understanding, the autonomy of a system denotes independence from an outside supervisor (human operator or exogenous system) being able to decide and change the way of action as far as appointed a priori by the system designer. Automation, on the other hand, means that the system is acting in a predefined way, precisely as it is programmed to do, without the possibility to adapt and change its actions in the future. The authors show that most of the taxonomies are based on similar boundary levels. The lowest level of maturity is usually fully manual, whereas the highest level of maturity constitutes a fully autonomous state.

Müller et al. [Mül+21] study different definitions, characteristics, and related abilities of autonomy with regard to industrial autonomous systems. Based on the investigated literature, they define an industrial autonomous system as a delimited technical system that achieves its objectives systematically and without external intervention despite an uncertain environment. Their definition of an autonomous system comprises four criteria: (1) systematic process execution, (2) adaptability to uncertain environmental conditions, (3) self-governance, and (4) self-containedness. The authors also provide a number of related abilities derived from the literature, as well as a detailed example of a flexible manufacturing system that includes a mobile robot to represent an industrial autonomous system.

In the context of assembly systems, Scholz-Reiter and Freitag [SF07] define autonomy based on two criteria, whereby a system must exhibit (1) independence from other systems and its environment, as well as (2) the means to act and control itself. The authors state that the first criterion can be met by clustering a system into subsystems with standardized interfaces; the second requires that the decision-making be decentralized, thereby empowering the respective subsystems. A degree of autonomy can then be derived by the freedom of action and the respective subsystems' ability to use this freedom for independent action.

In the case of robotics, Pfeifer and Scheier [PS01] define autonomy as independence of control, which is more specifically a property of the relationship between two agents (i.e., a system designer and an autonomous robot). They also posit that autonomy is not an all-or-nothing issue; instead, there are different degrees of autonomy and no agent is totally autonomous. Within their definition, there are always external factors, such as dependence on the environment or other agents, that cannot be controlled by an agent. (Although an environment is not usually presumed to possess the capacity for intent, other agents may be.) According to the authors, the degree of autonomy depends on self-sufficiency, evolution, and learning/development.

Fottner et al. [Fot+21] present a definition of autonomous intralogistics systems, a classification matrix, and state-of-the-art for enabling technologies and methods as well as application examples. The definition characterizes autonomous intralogistics systems as self-contained with material and information flows that are planned, executed, controlled, and optimized decentrally. Such systems cooperate and interact with other systems and humans. The classification matrix consists of two dimensions: automation stage and task level. Each task level, from system execution and real-time control to overall system planning, can achieve six different stages of automation. The stages span from the lowest (no automation) to the highest (full autonomy).

In the context of mobile robots in intralogistics, members of the VDI[2] Technical Committee FA 309 "Automated Guided Vehicle Systems" published an autonomy guide [Alb+22], in which they explain various terms and differentiate between automatic and autonomous functions of mobile robots for driving, safety, and load handling. According to the authors, autonomous functions imply situative reactions to changing external conditions and system states. Furthermore, they introduce an autonomy and fulfillment index with a scoring system based on the previously defined autonomous functions, which also captures the necessity for the use case. They state that autonomous functions are not fundamentally good or bad; rather, they must fit the requirements of the application.

Summarizing the definitions of autonomous systems, autonomy generally refers to the ability to self-govern and make decisions independently with minimum or zero external influence or control. Because few systems are fully independent from external factors, most are not totally autonomous but nonetheless achieve a certain degree of autonomy.

2.2.2 Characteristics of automated and manually operated systems

Fig. 2.8 shows exemplary key advantageous characteristics of automated and manually operated warehouses. Autonomous warehouses essentially

[2] Association of German Engineers (German: Verein Deutscher Ingenieure - VDI)

combine the advantages of both. In the following paragraphs, we describe these characteristics before introducing our vision of an autonomous block stacking warehouse.

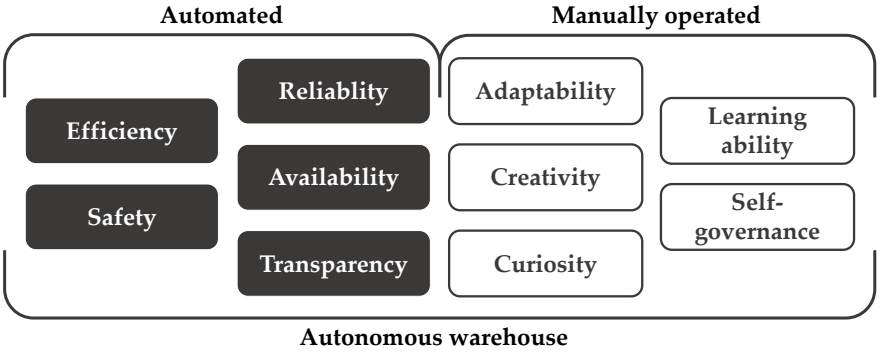


Fig. 2.8 Exemplary key advantageous characteristics of automated and manually operated warehouses.

Automated block stacking warehouses: The first characteristic where automated warehouses excel is **efficiency** because sophisticated storage strategies and algorithms can be applied (e.g., for path planning, storage location assignment, and sorting). The second advantageous characteristic is **safety** (i.e., avoiding injury to people or damage to goods). Automated systems can be designed to minimize incidents and take no risks. The third advantageous characteristic, **reliability**, is linked to safety but instead focuses on process execution—meaning that robust processes ensure a high degree of predictability. Stable predictable processes help to reduce fluctuations of up- and downstream processes and facilitate planning. Redundancies of core functions reduce the risk of system breakdowns. The fourth advantageous characteristic is **availability**, which can be ensured around the clock as long as it is sufficiently maintained. The availability of automated systems during off-peak hours can be used for rearrangement operations. Finally, the fifth advantageous characteristic of automated block stacking warehouses is **transparency**. All required information of the current system state is available at the respective entities and can be used for decision-making. This is also a key benefit compared to manually operated warehouses, where warehouse managers usually need to conduct frequent stocktaking.

Manually operated block stacking warehouses: The advantageous characteristics of manually operated warehouses are linked mainly to human intelligence. The first such characteristic is **adaptability**. Humans can assess and change storage strategies and objectives according to the situation while simultaneously planning ahead of time. For example, humans may deviate temporarily from a defined strategy to be more efficient when the warehouse is at full capacity: Instead of bringing unit loads to their designated storage

locations, only the closest available can be selected. The unit loads can be relocated to the correct storage locations later on when there is less activity in the warehouse. Adaptability is often linked to the second characteristic, **creativity**. Humans are capable of coming up with clever ideas and applying problem-solving skills. Especially in exceptional situations, humans are often able to find creative ways to deal with problems. For example, humans may temporarily use aisle space as storage locations or spontaneously change the internal layout. The third characteristic is **curiosity**, such as when humans are able to sense and investigate acoustic or visual anomalies in the warehouse, follow their instincts, and eagerly test new ideas. The fourth characteristic is **learning ability**, whereby Humans are able to acquire and improve their knowledge and skillsets based on experience and feedback. The fifth characteristic is **self-governance**, which means that humans can control their own behavior and make decisions without external influence. This involves self-assessment (i.e., being able to monitor one's own performance in detecting and responding to errors or malfunctions).

Moreover, the combination of these advantageous characteristics imbues humans with great versatility. Versatile, flexible human operators can switch spontaneously between different tasks and areas (e.g., floaters) dependent on warehouse utilization.

2.2.3 Vision of autonomous block stacking warehouses

This vision aims to describe desired target capabilities of an autonomous block stacking warehouse operated by AMRs. The intent of autonomous block stacking warehouses is to bring together the advantageous characteristics of both automated and manually operated warehouses (see Fig. 2.8). The goal is a self-regulating system capable of adapting to changes in a dynamic environment with a minimum of dependencies on the outside world or external systems. As explained by Pfeifer and Scheier[PS01] (see Section 2.2.1), there are no fully autonomous systems—only different degrees of autonomy.

In the following, we present the three core benefits of autonomous block stacking warehouses and propose how they could be achieved:

1. Adaptive fully autonomous warehouse operation by solving all short- to long-term decision problems involved.
2. Extremely low setup time and costs.
3. Efficiency increases with respect to configurable optimization targets.

(1) Adaptive fully autonomous warehouse operation by solving all short- to long-term decision problems involved: Automated block stacking storage systems are operated by AMRs, which are able to navigate freely and do not need any changes in the infrastructure of a warehouse (see Section 2.1.1). Even though such robots are an essential part of an adaptive system, we consider autonomous block stacking warehouses to have a much broader

scope, including all subordinate decision-makers that control operations in the warehouse. Such decision-makers include fleet management software or a guidance control system, an order management system, the functionality of a WMS/Enterprise Resource Planning (ERP) system, or a human operator/warehouse manager. Combinations of these systems are already able to fully automate a block stacking warehouse. However, they usually require human input to define/change storage rules or internal layouts. Future autonomous systems should be able to self-adapt, which could be achieved by simulation and learning-based approaches. Also, as discussed in the following section, the integration of the different systems is complex.

(2) Extremely low setup time and costs: Because block stacking warehouses need no infrastructure besides transport vehicles (including charging stations), the initial setup effort consists mainly of configuring, modifying, and integrating the system to fit into the respective conditions (e.g., processes, system landscape). We believe that this setup effort can, and should, be minimized with the goal of enabling plug-and-produce. However, a plug-and-produce solution comes with several prerequisites. First of all, any modification of the hardware or software to meet the requirements of a use case must be avoided. (Hence, a flexible and customizable system is required.) Second, intensive training of personnel should not be necessary, meaning the system is intuitive and easy to use. Third, plug-and-produce will be possible only if interfaces to surrounding systems are fully standardized or if these interfaces are generally optional/not mandatory. With regard to the multiple systems that are today involved as decision-makers, a future autonomous block stacking warehouse should integrate all core functionalities related to warehouse operations with a minimum of standardized interfaces.

In theory, an autonomous block stacking warehouse can be set up with only two basic steps, as shown in Fig. 2.9. First, the available warehouse space must be known. One of the easiest ways to get information about the warehouse environment is to scan the warehouse by exploring it with AMRs. Second, the system boundaries, along with the inputs and outputs, must be defined, where the AMRs should pick up and deliver in- and outgoing orders. In a simple scenario, these are the only steps required. However, further analysis of the necessary warehouse space and the required number of vehicles to operate the warehouse may be advisable. A necessary adjustment of the warehouse perimeter or the number of vehicles can be requested by the autonomous system, although this would require external intervention.

Operations in block stacking warehouses generally require interaction with the outside world only at the input and output system boundaries for incoming deliveries or outgoing retrieval orders. A lean way to handle this is shown in Fig. 2.10. AMRs can simply identify incoming goods via a barcode like the global trade item number (GTIN) standard or a radio-frequency identification (RFID) chip. Outbound orders can be requested from the respective

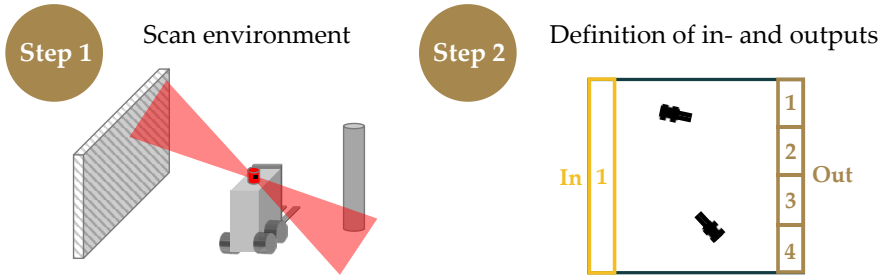


Fig. 2.9 Exemplary setup process of an autonomous block stacking warehouse: (1) AMRs scan the environment, (2) Definition of in- and outputs based on the warehouse layout (top view with black AMRs).

output-point (O-point) via a web application or an external system (e.g., WMS or ERP system) that is connected via a standard application programming interface (API). Even though interfaces to other external systems are not mandatory (in case a web application is provided), they may be useful and recommendable nonetheless. If incoming deliveries or retrieval orders are known ahead of time (e.g., production schedule or transport management system), the warehouse can prepare by rearranging goods and ensuring system availability. Also, stock levels may need to be known in a subordinate system, such as an ERP system, to check availability when planning retrieval orders (e.g., selling goods).

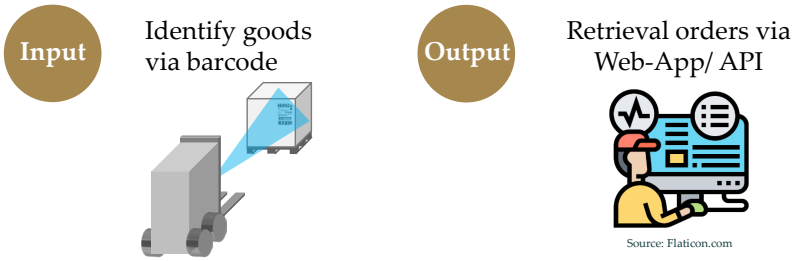


Fig. 2.10 Exemplary interfaces of autonomous block stacking warehouses with the outside world.

(3) Efficiency increases with respect to configurable optimization targets: Besides a simple setup process and minimal dependencies on other systems and the outside world, the main advantages of autonomous block stacking warehouses are enabled by a control system that integrates the capabilities of automated systems and human intelligence to tackle all relevant decision problems. Whereas current automated solutions often apply simple rules in a hierarchical decision pipeline, we believe that more sophisticated algorithms, as well as the consideration of interrelated problems (e.g., in a simulation), will enable further improvements of operational efficiency.

A key feature that distinguishes autonomous systems from automated systems is adaptability, which encompasses the abilities of self-perception, self-optimization, and learning. By such means, the control system can dynamically adjust its objectives and solution strategies in response to the current situation. If the input-points (I-points) are the bottleneck, allocation to the available storage locations closest to the I-points may be a viable solution—the objective being to service the inbound deliveries as quickly as possible. However, this could potentially produce chaos in the warehouse and ultimately delay future retrieval orders. To mitigate the risk of such potential complications while improving future warehouse operations, unit loads may be relocated during off-peak hours. The unit loads might simply be moved to new storage locations (slotting) or, alternatively, the actual internal layout could be changed.

We will discuss the decision problems as well as major challenges for autonomous block stacking storage systems in Chapter 3.

2.3 Conclusion

In this chapter we introduced block stacking warehouses by explaining the basic concept, the most relevant types of transport vehicles and unit loads involved, as well as the process flow. This was accompanied by a concise overview of areas of application as well as more detailed descriptions of several real use cases—together with their practical requirements—in the beverage industry, the hygienic paper industry, and in production supply scenarios. In sum, the great practical relevance and utility of block stacking storage systems was made abundantly clear. Furthermore, we discussed the concept of autonomy and its origins, providing several definitions related to engineering and industrial contexts. For the purposes of this research, autonomy refers to the ability to self-regulate and decide independently without external influence or control. Finally, we present a vision of autonomous block stacking warehouses that combines the key strengths of both automated and manually operated systems: Whereas automated systems provide superior safety, efficiency, reliability, availability, and transparency, nonautomated systems benefit from human operators' superior adaptability, creativity, curiosity, learning ability, and self-governance.

Chapter 3
Operational decision problems

In autonomously organized block stacking warehouses, AMRs control material handling without requiring any technical integration or WMS. In this chapter, we present related decision problems and provide a short literature overview for each one. We found that many existing approaches do not exploit the full potential of available flexibility. By focusing on operational decisions, we introduce the ABSWP and discuss major challenges which future solution approaches must tackle.

The adaptability of production and storage capacities in supply networks is a decisive competitive factor. Frequent changes on the supply and demand side of companies lead to dynamically shifting material flows and require continuous adaptation and relocation of storage capacities. Many forms of technical warehouse infrastructure cannot meet these requirements. A type of warehouse, which is highly flexible, easily expandable, and movable is a block stacking warehouse. In block stacking warehouses, unit loads (usually pallet-based) are placed side by side on the floor and stacked on top of each other. This requires no infrastructure and is therefore associated with low investment costs. In today's block stacking warehouses, the efficiency of material handling still depends on the skills of human operators. In this work, we consider autonomously organized block stacking warehouses, where AMRs carry out material handling and take over human decision-making without an additional WMS. This autonomous system should be able to adapt to dynamically changing variables like stochastic in- and outbound flow of unit loads as well as varying warehouse setups. Compared to other infrastructure-based warehouses with racks, lifts, or cranes, the unique feature of block stacking warehouses is that it is generally not mandatory to define any configurations or layout designs. Decisions can be made solely operational, where agents are able to evaluate each new situation and decide accordingly. That is exactly what human operators are skilled at. They are extremely flexible and often act subconsciously. An example is truck loading,

where a human operator might place pallets in the middle of a path to speed up an upcoming loading process. Improvising an increased temporary storage capacity by placing pallets against the storage rule in front of rarely used items is another example. This behavior cannot be easily reproduced by any existing automated systems. We commence this work by giving an overview and classifying related decision problems. We then describe the considered system decisions and present the state of research on related systems. Based on these findings, we introduce the ABSWP. Afterwards, we discuss major challenges and future research needs in the context of the ABSWP and then provide a conclusion to our findings.

3.1 Overview of decision problems

In the past years, several general frameworks on warehouse design and operations with associated decision problems have been presented (see, e.g., [Rou+00], [GGM07] and [GGM10])). In the following, we present major decision problems related to block stacking warehouses. Before introducing these decision problems, we present the warehouse setup. As shown in Fig. 3.1, the setup determines the environment and the available resources, and for the ABSWP we assume that both aspects are given. The system environment is defined by the following aspects:

- **Storage space dimensions**: The dimensions of the storage space are usually based on the required storage capacities and estimation of additional free space for vehicle movement, safety areas, and spare capacity.
- **Building characteristics**: This includes characteristics such as the surface condition, the load-bearing capacity of the floor, the illumination level, and the electrical connections for energy supply.
- **Number and position of load transfer areas (I/O-points or In- and Outbound docks)**: input/output-points (I/O-points) are the interfaces of the storage system to the outside world. They can be, for example, loading docks for trucks or intralogistics connections to other areas like production.

For operations in autonomous block stacking warehouses, additional resources as shown in Fig. 3.1 are necessary. These resources substantially define the operational performance of the system. In the case of block stacking warehouses, the subsequent resources have to be determined:

- **Number and type of vehicles**: The vehicle type (e.g., AMRs, robotic forklift, or pallet truck) depends on the characteristics of the transport structures and units. The number of vehicles depends on warehouse size and performance requirements.
- **Number and Position of charging stations**: An adequate capacity of charging stations must be available to recharge vehicles.

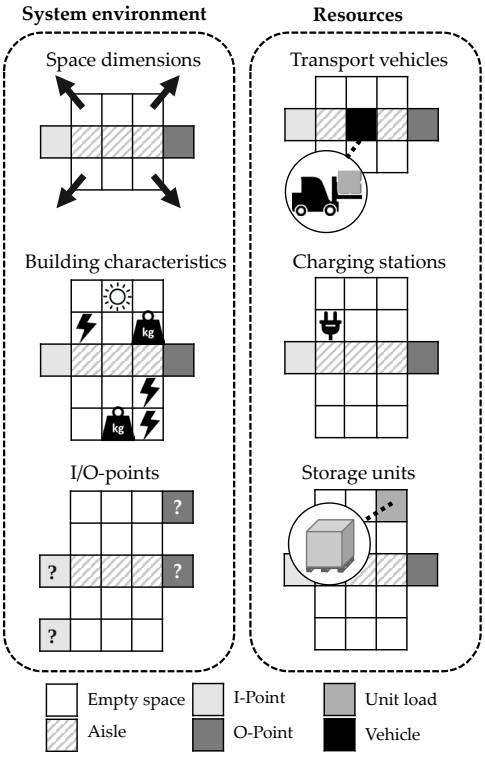


Fig. 3.1 Warehouse setup with specified system environment and resources similar to [PM20].

A variable number of vehicles is especially important in order to react to changing material flows. In the following, we assume that we have just one type of vehicle and that the number of vehicles and charging stations are fixed. Furthermore, we assume that a suitable information technology (IT) infrastructure is available. In some cases, such as in cross-docking, a flexible assignment of goods to I/O-points might have a positive impact on the system's performance. However, for the ABSWP we assume that the assignment is given and known in advance.

The ABSWP comprises the system decisions within the boundaries of a predefined warehouse setup shown in Fig. 3.2. We start by introducing each of the related decision problems briefly before providing more detailed introductions as well as literature reviews:

- **Internal layout design problem:** The internal layout design problem deals with the partitioning of a warehouse into aisle and storage space.
- **Storage location assignment problem:** The storage location assignment problem (SLAP) determines the storage location when placing a unit load in the warehouse.

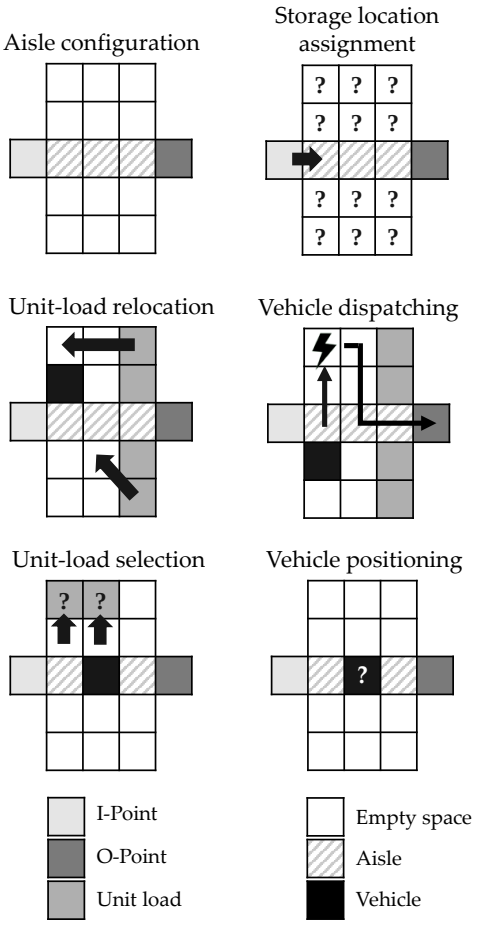


Fig. 3.2 Overview of related operational decision problems similar to [PM20].

- **Unit-load relocation problem:** The unit-load relocation problem deals with rearrangement operations that may be necessary to reach a unit load or can be helpful to improve the allocation of unit loads.
- **Vehicle dispatching problem:** The vehicle dispatching problem involves three sub-problems: Assigning tasks to AMRs, defining the sequence for executing the tasks, and determining a conflict-free path.
- **Unit-load selection problem:** The unit-load selection problem is about selecting a unit load for retrieval. This decision is only necessary when several unit loads of the same SKU are available in the warehouse.
- **Vehicle positioning problem:** The vehicle positioning problem concerns itself with the question of where an AMRs will be located when it is idle.

In the following, we introduce each of the decision problems related to the ABSWP and provide short literature reviews. Solving these decision problems requires pursuing one or multiple objectives that may be adapted over time. Objectives for storage systems are often based on cost or time, but can also be other target figures such as environmental metrics. Examples are the minimization of travel time, travel distance, makespan, tardiness, task completion times, cost of movement, expected waiting times, energy consumption, emissions output (e.g., carbon emissions), or maximization of space utilization and throughput. The metrics can be absolute numbers or averages. If several objectives are defined, they may be in conflict. A classic trade-off in storage systems is an objective related to fast traveling and the objective to maximize space utilization. Dense storage layouts usually increase travel times and the necessity of relocation due to blockages. The choice of objectives has a significant impact on decision-making.

3.2 Internal layout

In this section, we describe the internal layout problem followed by a brief review of related literature.

3.2.1 Problem setting

Layout design problems can be classified into external and internal layouts. External layouts define spatial boundaries as well as the number and positions of I/O-points and are assumed to be given as described in Section 3.1. Internal layouts define the positions and dimensions of aisles for vehicle movement and bays (also referred to as storage areas). Hence, the internal layout problem is also known as the aisle configuration problem. Especially for infrastructure-based systems, these layout problems are typically long-term decisions and not operational (see [GGM10]; [GGM07]; [Rou+00]). In internal layouts, four types of locations can be differentiated:

- **Inbound dock (I-point; also known as strip door):** New unit loads arrive at inbound docks. The exact position and orientation of inbound docks are given by the external layout.
- **Outbound dock (O-point, also known as stack door)**: The retrieval of unit loads ends up at an outbound dock, where the unit load is removed from the system. Specifications of outbound docks are also given by the external layout. Generally, in- and outbound docks are inter-changeable (e.g., open doors). However, defined in- or outbound docks are often used to simplify processes and achieve a directed material flow.
- **Aisle**: Aisles are permanently unoccupied cells, which serve as cleared space for fast movement of vehicles. Aisles are usually oriented length-

and crosswise towards cardinal directions (orthogonal aisles are cross-aisles). Nevertheless, any diagonal orientation and shape are possible.

- **Bay**: A bay is a storage area for unit loads consisting of one or multiple cells of the grid. Depending on the size and shape of a bay, it may not be possible to directly access all unit loads. It may be necessary to relocate blocking unit loads on top and in front of the target storage location.

Fig. 3.3 shows the introduced main location elements of a block stacking warehouse. The position of I/O-points is defined in the external layout. Often, a receiving or shipping area is located next to the I/O-points. A dedicated receiving area is for example important for the fast unloading of trucks and quality checks of incoming products. A shipping area allows to provide the requested goods in a required loading sequence and decouples the storage from the loading process. I/O-points are generally interchangeable. However, defined I/O-points can help to achieve a simple and directed material flow.

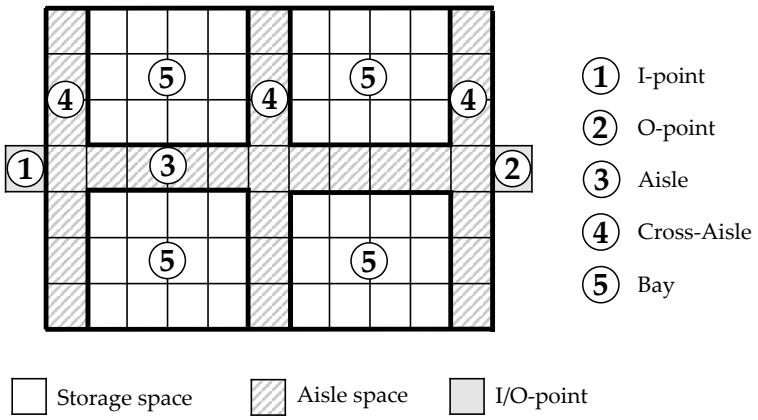


Fig. 3.3 Example of the main location elements in a grid-based block stacking storage system.

To assess different layout variants, the types of command cycles and storage policies (see Section 3.3) are defined. In single command (SC) cycles, the vehicles always return to the starting point after executing a storage (replenishment) or retrieval task. Hence, vehicles travel one way without carrying a load. dual command (DC) cycles combine storage and retrieval tasks to avoid empty runs, but also require traveling back to the starting point. Thus, it requires alternating receiving and shipping tasks, which is difficult to establish. In continuous multi command (MC) operations, traveling between any location in the warehouse is possible. This means that the travel time and travel distance depend on the assignment and sequence of transport tasks to the vehicles (see Section 3.5). In infrastructure-based warehouses, all I/O-points are connected via aisles and cross-aisles to guarantee direct access

to and from the warehouse. These aisles and cross-aisles can be uni- and bidirectional allowing directional or two-way traffic. High traffic volumes and two-way traffic may also require aisles with multiple lanes. Additionally, space for safety stock and honeycombing (see Section 3.3) must be considered as well.

However, with regard to the ABSWP, we are not convinced that it is necessary to determine internal layouts as it is the case for infrastructure-based systems. It must be ensured, that vehicles can move sufficiently within a warehouse. This requires unoccupied areas like permanently free cells in aisles (besides traffic) or temporarily empty cells of bays. Thus, it is possible that a block stacking warehouse consists of only a single bay and temporarily empty cells. On the other hand, internal layouts with aisles could be beneficial to reduce complexity and speed up processes in a way humans are already approaching it today. Internal layout designs in block stacking ware-houses can be adjusted more frequently than in infra-structure-based warehouses. In order to assess internal layouts in block stacking warehouses with multiple I/O-points properly, MC operations are necessary. Vehicles should be able to perform tasks continuously and travel between any location in the warehouse. Moreover, human drivers would not always run SC or DC cycles and return to a parking or starting position, unless they have to charge batteries or want to avoid congestion.

3.2.2 Prior work

There have been publications on the internal layout design problem since the 1960s. Berry [Ber68] developed a cost function, where occupation costs (for space utilization) and traveling costs (based on average distance) are considered for horizontally, vertically, and diagonally oriented aisles in rectangular prisms. He found that a layout, which minimizes traveled distance differs from a layout where space utilization is maximized. Later Goetschalckx and Donald Ratldff [GD91] calculated the optimal amount of lanes and the lane depth for single and multiple products to minimize the required warehouse area but did not consider material handling costs concerning the number, length, and orientation of aisles. More research dealing with the determination of the optimal lane depth in block stacking warehouses can be found in Accorsi, Baruffaldi, and Manzini [ABM17]. An approach to dynamically change the lane depth periodically requiring the relocation of unit loads was introduced by Lee [Lee19]. They define the so-called dynamic block stacking problem as a markov decision process (MDP) and apply a stationary optimal policy. By incorporating the storage and material handling costs in the reward function, they intend to find a good balance of aisle space and loss due to honeycombing.

Larson, March, and Kusiak [LMK97] divided a heuristic approach in three phases. In the first phase, the shortest travel time path from an input point to an output point is determined by minimizing the number of turns and maximizing the length of the longest arc. In phase two, the optimal row depth and the required number of storage locations are calculated. Finally, in phase three, a ranking of items is developed based on throughput and storage requirements. Most desirable storage areas in terms of travel distance are assigned to items with the highest ranks. Whereas Larson, March, and Kusiak assumed rectangular travel in a class-based (CB) storage, Gue and Meller [GM09], Gue, Ivanović, and Meller [GIM12], and Öztürkoğlu, Gue, and Meller examined diagonal travel for cross-aisles of any shape under a random storage policy. These unconventional cross-aisle designs have been able to significantly reduce expected travel distances compared to traditional designs with orthogonal cross-aisles. Derhami, Smith, and Gue recently published several papers on layout design in block stacking warehouses ([DSG16]; [Der17]; [DSG17]; [DSG19]; [DSG20]). They developed a model for wasted storage space and used a simulation-based approach to simultaneously optimize conflicting objectives; space utilization (maximize) and material handling costs (minimize).

3.3 Storage location assignment

In this section, we describe the SLAP followed by a brief review of related literature.

3.3.1 Problem setting

The SLAP belongs to the category of assignment problems, where at least two or more elements have to be matched. In the case of SLAP, incoming products (here: unit loads) are assigned to storage locations while considering a certain objective. Gu, Goetschalckx, and McGinnis [GGM07] divided SLAP into three categories based on the amount of available information regarding the arrival and departure times of unit loads.

- **Item information (SLAP-item information (II)):** All information about the arrival and departure times of each unit load are available. Hence, storage locations can be determined based on known due times of unit loads.
- **Product information (SLAP-product information (PI)):** Items (e.g., unit loads) are instances of products. Thus, information is available on the product-level (as an average) and not for each individual item. Products can be combined to classes. In class-based storage, these classes are assigned to storage locations. If each product builds its own class, this

is called dedicated storage. Another possibility is that all products are assigned to only one class. This is called random storage.

- **No information (SLAP-no information (NI)):** In this category no information is available. This can be, for example, the case for new product arrivals without historical demand information.

We assume the transport capacity of vehicles to be one unit load. If the transport capacity is greater than one (picking multiple items), probabilities of items occurring in the same order can be examined as part of the correlated storage location assignment problem. In all strategies, a trade-off between objectives like storage space utilization and efficient material handling has to be found. Wasted storage space, which cannot be used, should be prevented. Some storage policies dedicate storage space (e.g., lanes) temporarily to a SKUs. This prevents that unit loads of another SKU are obstructing the direct access and require unit-load relocations. However, if the dedicated storage space is not fully occupied and cannot be filled with unit loads of another SKU, it is a waste of storage space called honeycombing. Honeycombing can only be eliminated by fully emptying or occupying the dedicated storage space. In the ABSWP the SLAP is a major subject. Solutions (e.g., storage policies) have a substantial impact on warehouse performance (e.g., retrieval time and space utilization) and internal layout decisions. Assessing storage policies especially depends on the type of command cycles shown in Fig. 3.4. In a block stacking warehouse, there are situations where a new inbound load arrives and a vehicle solely runs SC cycles from an I/O-point to the target storage locations. Especially in warehouses with only one I/O-point, SC, and DC cycles are relevant. However, if there are many storage and retrieval tasks for multiple I/O-points simultaneously, it might be beneficial to distribute and sequence these tasks flexibly being able to execute any kind of operation. In manually organized block stacking warehouses, human operators also do not follow a strict operation mode and combine transport tasks in a smart way. This flexible sequencing and execution of operations is called MC operations. Moving strategies play a crucial role as often SC or DC cycles are assumed to be executed as a simplification in analytical models. However, if the travel distance depends on the execution sequence for stochastic storage and retrieval tasks, an accurate estimation is very difficult. Simulation-based approaches can deal with this problem, but are computationally expensive.

3.3.2 Prior work

The application of an appropriate storage policy depends on the availability of data. If the item or product history is available, it is possible to analyze these items or product activities systematically beforehand. This is called profiling. Frazelle [Fra16, pp. 56] describes item activity profiles and possible approaches to warehouse slotting. A very basic principle of pattern recogni-

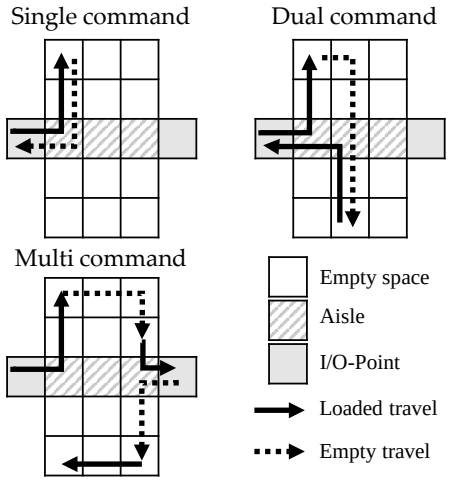


Fig. 3.4 Types of operation command cycles.

tion is Pareto's law (also referred to as ABC analysis) where, for example, a minority of SKUs may represent the highest total number of picks, indicating their popularity. Based on this, SKUs with highly ranked popularity could be assigned to the best storage locations in terms of the shortest travel time. Over many decades, a large number of storage policies and rules have been developed. Reyes, Solano-Charris, and Montoya-Torres [RSM19] provide an overview of recent publications and applied policies and rules regarding the SLAP. They show that CB and random-based (RB) policies are the most widely used. In the following, a short introduction of the most popular storage policies and rules is given.

Besides a random choice of a storage location, the closest open location (COL) or the furthest open location from the pickup location (I-point) based on the distance can be chosen. A possibility is also to chose a location that has been empty for the longest time period, a so called longest open location. [GGM07] In case of DC cycles, rules like nearest-neighbor (NN) or shortest leg (SL) may be applied [Han+87]. Both rules do a small look-ahead and consider also the travel distance to the pick location of the next retrieval task following a storage task. NN selects a storage location that minimizes the distance from the storage location to the following retrieval location. SL adds additionally the distance from the pickup location to the storage location. The rules for SC and DC cycles can also be combined by switching to a rule like COL, when a storage task is not followed by a retrieval task (e.g., [VG00]).

In the 1960s, Heskett [Hes63] proposed the storage policy cube order per index (COI). The COI rule is a turnover-based policy, where unit loads are ranked based on the ratio of inventory volume to demand rate. Afterwards, unit loads are sequentially assigned to the most desirable locations. Kallina

and Lynn [KL76] showed that COI is indeed an optimal solution. This is true in the case of single command order picking. However, for MC operations COI can be a bad choice [Sch14].

Hausman, Schwarz, and Graves [HSG76] compare CB, RB and fully turnover-based (TOB) policies. They introduced a CB storage policy based on COI, where locations are assigned randomly within classes. Their approach also considers SC operations. It could be shown that their approach potentially leads to a significant reduction in travel time. In 1990 Goetschalckx and Ratliff [GR90] developed a shared storage policy based on the duration-of-stay (DOS) of unit loads. Instead of using the average turnover rate, which is a product characteristic (one turn-over rate for all unit loads of the same product), they use the expected duration-of-stay for each unit load (time in storage for each unit load of the same product can be different from each other), which is a single unit characteristic (SLAP-II). They distinguished between static and adaptive storage policies and developed solutions for both cases based on DOS.

The more recent research by Rimélé et al. focuses on adaptive storage strategies. They published three papers ([Rim+21a]; [Rim+21b]; [Rim+21c]) proposing dynamic storage policies and learning-based approaches in an robotic mobile fulfillment system (RMFS). Regarding cycle time a novel supervised tree search strategy performs best, with an improvement of approx. 20% compared with random allocation.

3.4 Unit-load relocation and pre-marshalling

In this section, we describe the unit-load relocation and pre-marshalling problem followed by a brief review of related literature.

3.4.1 Problem setting

We categorize relocations of unit loads into two types: relocation for unit-load retrieval and relocation to improve future operations. In order to retrieve a unit load, it might be necessary to relocate obstructing unit loads stored on top or in front of it. This can, for example, be the case, if lanes contain a mix of SKUs or storage retrieval rules like due dates, limited duration of stay, FIFO, or LIFO must be considered. This type of relocations cannot be avoided and should be realized immediately. Relocation to improve the future warehouse performance deals with the question, of whether existing internal layouts should be rearranged or when stored unit loads should be repositioned. Thereby, efforts for relocation and expected benefits must be continuously investigated. If future benefits in operations surpass the required efforts, the relocation process could be triggered. This decision gets more difficult if benefits and efforts cannot be compared directly due to different measurement units. An example is an improved average retrieval

time as a benefit and additional travel distance as a repositioning effort. Finally, also the extent of rearrangement or repositioning actions must be defined. The case of small incremental actions is called healing (see [Kof+11]). The rearrangement of the whole or parts of a storage system (also known as re-warehousing or re-slotting) typically involves the relocation of goods on a larger scale. Pre-marshalling is a special case of such a rearrangement process, where unit loads are sorted in off-peak hours according to their retrieval times (see Chapter 4 and Chapter 5).

Both, relocation for retrieval and relocation to improve future operating efficiency can be combined. A relocation move might be necessary for the retrieval of another unit load and at the same time improve the future efficiency of warehouse operations. Also in the ABSWP relocations to access blocked unit loads for retrieval cannot be avoided. This kind of relocation decisions need to be made when solving the SLAP. Relocation to improve the future warehouse performance on the other hand is optional. It can reduce access times during busy hours.

3.4.2 Prior work

In this brief literature review, we give an overview of related problems. Regarding the mandatory relocation for unit-load retrieval there are related problems in maritime transportation. The problem of how to retrieve a batch of containers in a predefined order with a minimum number of moves is known as the blocks relocation problem (see e.g., [CSV12]) or container relocation problem (CRP)) (see, e.g., [ZBM19]; [MGM19]).

In terms of relocation to improve future operations, pre-marshalling problems are heavily studied for container terminals (see, e.g., [CSV11]; [LK14] and e.g., [TV16] for the stochastic problem). Although most literature considers containers to be carried by cranes, the problems are similar to the ones in block stacking warehouses. A detailed literature review on pre-marshalling problems is provided in Section 4.2 and Section 5.2.

Re-slotting/Re-warehousing is studied for many warehousing setups under varying terms. Christofides and Colloff [CC73] published one of the first approaches on warehouse rearrangement which rearranges items from an initial to a desired new location during idle time. Their solution finds a sequence minimizing total costs of operations in a dedicated storage scenario. Linn and Wysk [LW90b] proposed several control policies including a restoring policy in an automated storage and retrieval system (AS/RS). Thereby stored items are rearranged by alternating the two rules; clear-the-highest-turnover-zone-first and restore-the-highest-turnover-item-first. Muralidharan, Linn, and Pandit[MLP95] introduced heuristics for (re-)shuffling in idle time. Their approach is able to considerably increase operating efficiency and reduce waiting as well as service time in an AS/RS. Kofler et al. [Kof+11] compared

strategies for re-warehousing and healing. In the case of re-warehousing a warehouse is completely re-slotted. Afterwards, warehouse efficiency decreases over time. Healing on the other side improves existing solutions continuously. In their opinion, healing strategies are generally more appropriate for volatile environments and can be helpful as a supplementary measure to maintain operating efficiency after re-warehousing. Another publication, when relocation is performed during the runtime is from Carlo and Giraldo [CG12]. They are aiming in a similar direction with perpetually organized unit-load warehouses. Their strategy, rearrange-while-working, reorganizes an AS/RS during operations with the objective to minimize total travel distance.

3.5 Vehicle Dispatching

In this section, we describe the vehicle dispatching problem followed by a brief review of related literature.

3.5.1 Problem setting

In warehouse operations, storage and retrieval tasks are usually not known for a longer planning period (e.g., a day) but become known dynamically over the course of the day. Whenever more than one vehicle is available, each task needs to be assigned to a vehicle. If more than one task is allowed to be assigned to a vehicle, it needs to be decided in which sequence the assigned tasks are executed. Therefore, typical objective functions are the minimization of travel distances, travel times, task completion times, or tardiness. In order to calculate the distance and the time a vehicle needs for traveling from one point to another, the shortest (or fastest) path must be determined (also referred to as pathfinding methods). In a block stacking warehouse, the graph, which is used to determine the shortest paths, is variable over time. It depends on the current layout, or more precisely, on currently empty bays.

In practice, often relatively simple dispatching rules are applied to assign tasks to AMRs as soon as the tasks become known and an idle vehicle is able to take over the task. A dispatching rule can for example be assigning tasks to the AMRs closest to the starting point of a task. Especially in cases with many AMRs, congestion, live- and deadlocks of AMRs can lead to conflicts. In these scenarios, techniques for conflict-free routing (e.g., route segment reservation or designated zones) become necessary. In addition, constraints like priorities, time windows, and battery levels must be considered.

Another possibility for assigning tasks to AMRs and defining a sequence in an integrated way is to solve a vehicle routing problem (VRP). This involves the consideration of a longer planning horizon and increases the complexity. In the research of the last decades, many variations of VRP (e.g., vehicle routing problem with time windows and pick-up and delivery problem with

time windows) have been investigated. According to Pillac et al. [Pil+13] the problems can be classified into static or dynamic as well as deterministic or stochastic problems. In static and deterministic problems all required information (e.g., travel distance and time) is known prior to the planning period and the data does not change during task execution. Dynamic problems consider the evolution of information, which can be available very late, during the planning period, or even during task execution. Stochastic problems take into account the uncertainty of information, which may be expressed as a probability distribution.

Decentralized decision-making is enabled by market-based solutions: In market-based solutions, local entities, for example the AMRs, are bidders, which place a bid to an auctioneer (e.g., also a vehicle or a central entity). The auctioneer evaluates all bids and decides on the best allocation. Depending on the auction principle, a sequential single task auction or combined task bundles are auctioned [DVD20]. Vehicle dispatching is interrelated with all other decisions of the ABSWP and is required to evaluate actions. Important objective figures like travel time and distance are based on dispatching decisions.

3.5.2 Prior work

A common approach in practice of AAGV control is to apply a dispatching rule combined with a path-finding method. Le-Anh and De Koster [LD06] divide dispatching systems into decentralized and centralized dispatch-ing methods. An example for a decentralized dispatching method is the first-encountered-first-served rule, where AMRs continuously circulate and pick up the first load encountered as soon as they have available capacity. For central dispatching methods, continuous communication with all AMRs is required. A central control system guides AMRs based on real-time information like their positions and status. Central dispatching approaches are usually myopic and suitable for large instances. Egbelu and Tanchoco [ET84] classified dispatching rules as workstation-initiated (tasks at workstations claim vehicles) and vehicle-initiated (vehicles claim tasks) dispatching rules. Le-Anh and De Koster [LD06] furthermore divide dispatching rules into single-attribute (e.g., distance or workload-based), multi-attribute (e.g., shortest travel time and remaining queue space), hierarchical (based on added value), look-ahead (short forecast period) and preemption dispatching rules (allow re-assignment). Sequencing of the jobs follows often simple rules based on arrival time like the first-come first-serve (FCFS) strategy (see [FK19, pp. 178]). Van Den Berg and Gademann [VG00] studies the request selection rules longest waiting retrieval (LWR), nearest retrieval (NR), and nearest storage/retrieval (NSR) for finding a clever sequence. The LWR rule leads to fast response times, NR and especially NSR improve throughput but may lead

to longer waiting times. To cope with these long waiting times, the authors propose urgency rules for requests with long waiting times and analyzed the effect on mean and maximum response time as well as the number of late requests. The results show, that FCFS performs better than the priority rules. However, the authors note that a sophisticated approach that allows small changes in the sequence while still respecting due dates could perform better.

For the avoidance of conflicts there are a variety of solutions. Conflicts can for example be avoided by limiting the movement of robots to defined zones. However, this strategy is often not ideal since the work is usually not distributed evenly between the different zones. Some robots might be fully utilized while other are idle and cannot leave their zone. Another option is to integrate the avoidance of conflicts into path planning and motion control. Whereas path planning problems in static environments have been widely studied, problems with dynamically moving obstacles are more challenging since they also have to consider the time dimension [PL11]. To solve the problem centralized and decentralized solutions have been presented. Well-known centralized approaches are safe-interval path planning [PL11], conflict-based search [Sha+15] and post-processing of a plan using a temporal graph [Hön+16]. Decentralized solutions include among others velocity obstacles [Sna+11] and model-predictive control [Kam+17]. For more literature see e.g., [Fra+21].

In research, also combined dispatch and conflict-free routing problems have been addressed (e.g., [Des+03]). Dispatching rules are generally rather easy to implement and especially helpful in highly dynamic and stochastic environments where schedules would have to be updated frequently. However, Meersmans [Mee02] and Yang, Jaillet, and Mahmassani [YJM04] show that dynamic scheduling is able to outperform dispatching rules, but is only applicable for rather small instances. In order to make a dispatching decision (based on path information), path planning is required. A common approach for path planning is using graph search algorithms. They can also be applied to find a shortest path from a starting point to a destination after a dispatching decision is made. Well-known heuristic examples are the Dijkstra algorithm [Dij+59], the A* algorithm [HNR68], and the D* Lite algorithm [KL02].

Research on vehicle routing and scheduling problems, especially in transport logistics, is quite extensive. Since operations in block stacking warehouses are often confronted with a highly dynamic environment and stochastic variables like uncertainty in customer demand, this review is focused on dynamic solutions considering uncertainty. Thereby, Ritzinger, Puchinger, and Hartl [RPH16] distinguishes between preprocessed and online decision-making. Preprocessed decisions are based on information known prior to task execution. Based on this, general policies and rules (e.g., dispatching rules) or preprocessed rewards for actions and corresponding state representations

can be determined. Online decision-making is able to dynamically process a continuous information flow. Therefore, usually, a trade-off between decision quality and responsiveness must be found. Look-ahead strategies or rolling horizon procedures typically follow an action plan until a new event occurs. Then, a single decision for the current state or a revised action plan must be calculated. The selection of a suitable solution method depends in particular on the degree of dynamism and the desired reaction time. In order to make state-dependent decisions with respect to available stochastic information (referred to as policy), problems can be modeled as MDP or multi-stage stochastic models [RPH16].

Pillac et al. [Pil+13] divide solution methods for dynamic and stochastic problems in stochastic modeling, sampling, and other strategies. A stochastic modeling approach is approximate dynamic programming, where approximation techniques can be used to overcome the curse of dimensionality (see [Ulm17]). In sampling-based approaches, several scenarios are generated based on random distributions. Afterwards, each scenario is evaluated. Well-known is the multiple scenario approach, introduced by Bent and Van Hentenryck [BV04], which continuously updates a scenario pool by using the time between decisions.

3.6 Unit-load selection

In this section, we describe the unit-load selection problem followed by a brief review of related literature.

3.6.1 Problem setting

In many cases for the retrieval of unit loads, not a dedicated unit load needs to be picked, but a product/SKU needs to be provided for a customer. Therefore, usually storage rules or item retrieval policies like FIFO or LIFO are applied. If no item-related rules apply or a rule offers a certain degree of freedom, the unit-load selection is necessary. In this case, it is possible to select one out of many unit loads optimizing a given objective function such as minimization of travel distance or travel time. However, there are also use cases where, for example, a batch of unit loads is queued up in front of an inbound dock. This represents a waiting queue where all unit loads have an associated task. The tasks must be assigned to AMRs and sequenced as part of vehicle dispatching (see Section 3.5). In the ABSWP, item retrieval policies have a significant effect on other decisions like the internal layout. Some of the item retrieval policies are rarely found in block stacking warehouses. Often, unit loads are stored in lanes dedicated to an SKU. In this case, LIFO works well because the unit load, which has been stored last, can be easily accessed. A strict FIFO, on the other hand, would require relocation of all unit loads which are stored in front and on top. Item retrieval policies can only be applied correctly, if all

information like storage locations and arrival times of unit loads are available. In block stacking warehouses without additional WMS this is not always the case. An autonomous system can easily collect all required information and correctly apply item retrieval policies. In the presented application scenarios in Section 2.1.4, we observed that in practice these retrieval policies are rarely implemented strictly.

3.6.2 Prior work

Retrieval strategies can be based on time like FIFO and LIFO, based on emptying remaining quantities, based on travel time or travel distance, RB or other strategies (e.g., based on energy consumption).

The batch first-in, first-out (BFIFO) strategy is a softened FIFO strategy that can be used if products should be removed from storage over time. However, it provides more flexibility in terms of which exact unit load is going to be picked and offers hence, additional room for optimization, e.g., LIFO can still be applied within lanes. Also, other combinations of item retrieval policies like always emptying residual quantities first and otherwise applying a strategy based on travel distance are possible. Sometimes also BBD/expiry dates are used for first-expiry, first-out (FEFO) in order to prevent the perishing of the goods.

As mentioned in the introduction of the problem, these rules are not always applied strictly in practice. Often, certain deviations are tolerated (e.g., BFIFO or arrival times within a time window). If goods are non-perishable it is even possible, that no time-based rules are applied. This degree of freedom allows to select one out of many unit loads that optimizes the defined objectives. Unit loads can for example be selected based on travel distance or travel time.

Strategies for unit-load selection have mainly been investigated in larger scenarios as part of simulation studies. For instance Seidmann [Sei88] developed a dynamic control approach for AS/RS that adapts to seasonal fluctuations, where he relaxes the strict FIFO policy.

3.7 Vehicle Positioning

In this section, we describe the vehicle positioning problem followed by a brief review of related literature.

3.7.1 Problem setting

If a block stacking storage system does not have any tasks that need to be fulfilled, AMRs are idle. Idle AMRs have to be parked somewhere until a new task is received. This parking position (also referred to as dwell point)

can be a charging station, the current position, a specific home position or a position that allows fast reaction times without blocking any other AMRs.

If an AMR stays at the current position the travel distance and energy consumption is minimized. Driving to a charging station would increase the availability later on since the battery levels will be higher. A clever distribution (e.g., in central positions) can improve the response time to any kind of incoming tasks. These waiting, battery charging and repositioning decisions are subproblems of vehicle dispatching. However, we want to highlight vehicle positioning with its own section in this overview as it differentiates autonomous from manual systems.

Decision-making for vehicle positioning requires information of the whole system state which is often not available in manual system. Especially in lengthy waiting times, humans are not always attentive and might follow other interests. Some may head to charging stations or parking positions, others drive for a break to the common rooms or visit colleagues at other workstations. For newly incoming tasks, human operators may be distracted for a while and not be immediately available.

We believe that autonomous block stacking storage systems are able to make better decisions for vehicle positioning as they are able to calculate probabilities for the occurrence of the next incoming tasks.

3.7.2 Prior work

Solution approaches for vehicle positioning can be classified into static and dynamic as well as central and distributed methods. A good overview of the most important rules for selecting a waiting position is provided by Le-Anh and De Koster [LD06] (see also Egbelu and Wu [EW93] for AS/RS and Meer [Mee00] for guided vehicles.

Rules for static vehicle positioning are the point of release positioning (similar to drop-off point positioning), central zone positioning, circulatory loop positioning, and distributed positioning. For point-of-release positioning a vehicle simply waits at the target position of the last transport task. The central zone positioning rule sends the idle vehicles to designated areas, where they do not block any other vehicles or might charge their batteries. Vehicle positioning is often investigated under the assumption of guideways composed of one or several loops (so-called loop layouts). In this case, the circulatory loop rule may be applied, where idle vehicles travel in one of these loops until they receive a new task. In distributed positioning a vehicle can be directed to one of multiple available parking positions. [LD06]

In block stacking storage systems generally, any position in aisle space and any unoccupied position may be a possible temporary parking position. Since the system state and therefore also the possible parking positions are changing over time, it is necessary to investigate dynamic vehicle positioning.

Hu and Egbelu [HE00] studied conventional layouts with unidirectional travel and changing pick-up demands. Many publications on vehicle positioning can be found in the context of larger factory setups, for instance in cell manufacturing systems [Vah14] or in large semiconductor fabrications [Lee+21]. Related research can also be found in connection with the VRP or taxi dispatching, where waiting strategies and vehicle positioning strategies are important as well ([Pil+13], [Li+11]). Waiting and positioning strategies are essentially based on the probability of new incoming tasks (e.g., in a certain region or at a target point). It may for example be sometimes better to not take and process immediately the first incoming task, but to wait for a short time and be able to build an improved sequence or get a closer task. Another possibility is the active relocation of vehicles to a position with a high likelihood for the appearance of the next task. Bent and Van Hentenryck [BV07] present waiting and re-positioning strategies in online and stochastic vehicle routing.

3.8 The autonomous block stacking warehouse problem

After presenting all necessary decisions and solution approaches of related systems, we introduce the ABSWP. The ABSWP is influenced by its environment and by its resources, which we assume as given from the outside. Part of the predefined system environment is the storage space, multiple I/O-points and adequate building characteristics with restrictions such as the load bearing capacity. As resources, the number of each vehicle type, and the number and the position of the charging stations are given. In the context of a block stacking warehouse, we consider vehicles, which are able to carry only one unit load at a time. The vehicles are not restricted to SC or DC cycles and able to process tasks continuously (MC cycles). As usual for warehousing, the ABSWP must cope with highly dynamic and stochastic input variables as well as heterogeneous available information (e.g., a mix of item and product information). Storage and retrieval tasks associated to in- and outbound flow of unit loads are not known for longer planning periods. They enter the system dynamically from outside throughout the day. Future exact arrival or departure times from production scheduling or notification of logistics service providers are valuable pieces of information. However, usually only product information based on historical data is available for the prediction of in- and outbound loads. In certain cases, no information might be available. For storage and retrieval tasks, we assume that the assignment and sequence of in- or outgoing unit loads to I/O-points is given.

The ABSWP contains the following decisions. Some of the decisions are optional and not necessarily required for warehouse operations.

- **Storage location assignment:** Incoming unit loads must be assigned to storage locations if they are not directly forwarded to an O-point. A typical objective is the minimization of travel time. Therefore, it is beneficial to consider future storage, relocation, and retrieval tasks within the warehouse. Especially relocations for unit-load retrievals are often time-consuming and should be avoided. To achieve this, the number of required relocations for unit-load retrievals must be minimized. In order to fully utilize the flexibility of an autonomous block stacking warehouse, the assignment decisions should be adapted to the current in- and out-bound flow, as well as, to changing product portfolios.
- **Internal layout design (optional):** Internal layout design decisions define the spatial distribution of storage space and aisles for vehicle movement. In contrast to infrastructure-based systems, in block stacking warehouses, temporarily free cells can be used for vehicle movement. Hence, no layout must be defined, it only needs to be assured that enough space is available to maintain the operability of the system and to avoid a deadlock situation. However, even if aisles reduce the amount of available storage space, they are beneficial to increase the material handling efficiency. Thus, also in block stacking warehouses there exists a trade-off between material handling efficiency and storage space utilization. Hence, defining an internal layout is probably beneficial. Nevertheless, it also is implicitly a part of each storage location assignment decision and an internal layout design can be incrementally updated to new requirements.
- **Vehicle dispatching:** Indispensable is the consideration of vehicle dispatching. It determines the assignment of tasks to vehicles and the definition of a sequence. Tasks for vehicles are transport tasks, but also battery charging, waiting or repositioning. Vehicle repositioning decisions are not mandatory for operations, but could improve the efficiency. In the context of larger systems with many vehicles, also conflict free routes need to be ensured. Objective functions for vehicle dispatching are for example the minimization of travel distances, travel times, makespan or tardiness.
- **Unit-load selection:** The selection of a unit load for retrieval is necessary, if the unit load is not determined by item retrieval rules. Then, a unit load can be selected minimizing material handling effort, for example to achieve DC cycles with pickup and deliveries.
- **Unit-load relocation and pre-marshalling (optional):** Another optional decision is to change the internal layout or storage location of unit loads over the course of time to improve the future system performance. The question is when and to which extent improvement relocations are initiated. Improvement relocations are an additional effort but can be realized in off-peak times. For example, the rearrangement of unit loads during the night based on orders of the next day is reducing the access times.

3.8.1 Major Challenges

Considering the decision problems in section 4.1, solving the ABSWP is challenging. A high computational complexity of the tightly coupled decision problems is already proven. Due to its magnitude, already the formulation of the ABSWP as an integrated mathematical model is difficult. Different planning horizons and planning hierarchies make the ABSWP more difficult. Some decisions like unit-load selection, storage location assignment or vehicle positioning usually consider shorter planning horizons than finding an internal layout. Today, internal layouts are determined on a tactical level while SLAPs are solved instantly for each storage or relocation task within the boundaries of a fixed internal layout. This classical hierarchical approach could be disrupted: Layouts must not be determined in advance but can evolve incrementally as part of solving SLAPs over time. This requires the extension of the planning horizon of the SLAP while keeping the planning frequency.

Solutions for the SLAP (e.g., storage policies) depend on many variables like the internal layout variant with its I/O-points, defined operation cycles, the product portfolio, arrival and departure times of goods, the degree of available information and the set objectives. Human operators often structure storage locations and form product classes. A known structure helps them to find unit loads and storage locations easier (especially in case of changing personnel). Hence, they stick to established rules and storage policies. In an autonomous system, a fixed structure would not be necessary anymore, because all the vehicles know about the current state and storage locations of the system. This allows using shared storage policies, which can utilize storage space more flexibly (better utilization of attractive locations leads to reduced travel time) and require less maximum space. Besides, the influencing variables to find a suitable storage policy are not constant. Storage policies should be adjusted continuously over time. Thereby, a challenge is delayed rewards. A storage decision is often made with goals like minimizing order retrieval time, tardiness, or total travel distance. The reward function for these goals includes elements of the future process steps (unit-load relocations and retrieval tasks) until the unit load has left the system.

After a certain period of time, it could be helpful to change a storage location or internal layout. In automated storage systems, concepts to reposition unit loads in off-peak times are already available and proven to be helpful (e.g., to prepare for known upcoming orders of the next day). In block stacking warehouses, however, the internal layout can also be changed. Human operators do not change these internal layouts frequently. It requires additional planning and material handling effort. As a result, while product portfolios evolve, the internal layout often stays the same. The possibility of incrementally adapting the internal layout could be a huge benefit of au-

tonomously organized block stacking warehouses. Vehicle dispatching plays a key role and major impact in the ABSWP, because all other decisions are strongly interrelated. Today, often simple dispatching rules are applied in AGV systems. They are able to cope with large systems in a dynamic and stochastic environment. However, online solution approaches to solve a VRP with look-aheads could possibly lead to better results. Human drivers are often good at looking ahead. An example of this is the anticipation of peak periods and the preparation by charging vehicles beforehand.

Finally, we see a large number of practical requirements, which have to be met. Some requirements like exception rules or the consideration of due dates are relatively easy to consider. Other requirements significantly increase the complexity of the problem. One example is the variety of sizes and shapes of load carriers. This has an impact on internal layout designs and increases the number of feasible configurations. A product-specific requirement is the stacking capability (height clearance), which depends on the specific material statics and height restrictions of a warehouse. For some products stacking on top of one another is not stable. In this case, a different kind of stacking pattern like a pyramidal structure shown in Fig. 2.5, can be highly beneficial. This implies that the consideration of weight and height restrictions would not be enough for SKU-mixed block stacking.

3.8.2 Conclusion

In this chapter, we formally introduced the operational decision problems including related literature for autonomous block stacking storage systems. Based on the strongly related decision problems, we defined the aggregated ABSWP. It integrates the most relevant operational decision problems of block stacking warehouses. Given the complexity of the sub-problems and the wide range of research, developing solution approaches for the ABSWP is a challenging. Yet, we believe that these solution approaches have a high practical relevance and will enable more efficient operations in block stacking storage systems.

Even though human operators are outstanding in terms of creativity and flexibility, AMRs allow the use of more sophisticated algorithms instead of simple rules for decision-making in a fixed warehouse structure. Reliable and permanently available information eliminate long search procedures and daily stocktaking. Further potential advantages are among others the continuous adaption of objectives, strategies, and configurations (e.g., layout) while the system environment and resources (e.g., the number of AMRs) can be easily and seamlessly adjusted. The use of AMRs may also lead to a higher degree of visibility, reliability and accident prevention.

As a major feature of AMRs, we identified the constant availability. Almost every storage system is subject to a fluctuating utilization. Our simulation

study in [Pfr+22] demonstrates based on a real use case that AMRs are only utilized for approximately 25% of the time. A reason is that storage systems are usually specified and dimensioned to withstand maximum peak times. Whereas human operators take a break or leave the warehouse in off-peak hours, AMRs are still there. The consistently available AMRs allow to sort and rearrange a storage system in off-peak hours (e.g., night shift) and prepare the storage system for future orders.

Thus, the subsequent chapters in this thesis are dedicated to solve the UPMP in block stacking warehouses. Further next steps to tackle the ABSWP are described in our outlook in Chapter 6.

Chapter 4
Solving the unit-load pre-marshalling problem

Block stacking storage systems are highly adaptable warehouse systems with low investment costs. With multiple, deep lanes they can achieve high storage densities, but accessing some unit loads can be time-consuming. The UPMP sorts the unit loads in a block stacking storage system in off-peak time periods to prepare for upcoming orders. The goal is to find a minimum number of unit-load moves needed to sequence a storage bay in ascending order based on the retrieval priority group of each unit load. In this paper, we present two solution approaches for determining the minimum number of unit-load moves. We show that for storage bays with one access direction, it is possible to adapt existing, exact tree search procedures and lower-bound heuristics from the CPMP. For multiple access directions, we develop a novel, two-step solution approach based on a network flow model and an A* algorithm with an adapted lower bound that is applicable for all possible access direction combinations. We further analyze the performance of the presented solutions in computational experiments for randomly generated problem instances and show that multiple access directions greatly reduce both the total access time of unit loads and the required sorting effort. The chapter is based on our publication [PMT22].

Warehouse systems are constantly adapted to dynamic material flows due to changing customer demand or evolving production capacities. Block stacking storage systems are especially capable of dealing with these challenges thanks to their flexibility, expandability, and ease of setup/breakdown. In block stacking storage systems, unit loads (e.g., pallets or boxes) are placed on the floor and stacked on top of each other in a grid-based layout. Block storage does not require the installation of technical infrastructure. This keeps investment costs low and allows companies to quickly react to changing business conditions. Types of block storage systems consisting of movable units on the floor include, for example, pallet-based block storage, RMFS, container yards, or buffer storage with material transport trolleys.

Fig. 2.1 in Chapter 2 shows the basic structure of a block stacking storage system. Each lane of a storage bay is filled with a sequence of stacks. Only the outermost stacks can be accessed by transport vehicles like robotic forklifts. In this type of storage system, unit loads can be accessed from multiple directions, i.e., from an adjacent aisle or cross-aisle. This provides immense flexibility compared to systems that can only be accessed from the front. However, unit loads can still be blocked from being retrieved if other unit loads are (i) stored on top of them or (ii) stored between the unit load and the direction of access (aisle/cross-aisle), and the blocking unit load has a later retrieval time than the one being removed.

AMRs (more specifically in this work, robotic forklifts) are being increasingly used to control material handling and storage systems [Fra+21]. However, most block stacking warehouses are currently still organized manually. Human operators generally compose warehouses with a fixed layout and storage zone structure, in which SKUs that represent product variants are always placed at defined storage locations (e.g., a single SKU is always assigned the same lane).

In autonomously organized block stacking warehouses, the advantage of robots is that they have exact knowledge of the system state, including the storage location of each unit load. Thus, they naturally support storage policies without a fixed allocation of SKUs to storage locations (shared storage policy) as well as extensive rearrangement operations. This increases the storage density by better utilizing the available storage space. Another advantage of AMRs is that they are operational around the clock. Hence, off-peak hours can be utilized to sort the system, improving future access and retrieval times. For example, the storage system could prepare for an expected demand sequence like the loading times of trucks or the production schedule over the next few days. We introduce this problem for block stacking storage systems as the UPMP. An overview of operational decision problems in autonomously organized block stacking warehouse is provided in Pfrommer and Meyer.

The pre-marshalling problem is a well-known problem in the container stacking [LK14]; [CSV11] and steel slab and coil shuffling literature [TZL12]; [Ge+20]. In the CPMP, all stacks are accessible from the top via a crane. Fig. 4.1 a) illustrates an example with two stacks containing four unit loads. The UPMP with side access from only one direction in Fig. 4.1 b) is similar to the CPMP. In a sequence of stacks per lane, the unit loads at the outermost stacks can be accessed directly. The only difference is that the sequence of the unit loads is derived from multiple stacks instead of a single stack. Both examples Fig. 4.1 a) and Fig. 4.1 b) allow direct access to only two of the eight unit loads (dark-gray). To reach all other unit loads, relocation of the blocking unit loads on top or in front is necessary. A common setup in warehouses is the use of pallets and forklift trucks, which allow access from at least two

opposite directions, in many cases even from four directions. Fig. 4.1 c) shows an example of the UPMP, where three access directions are available and only the back is blocked by a wall. This leads to direct access to every stack, which speeds up access times and reduces the number of reshuffling moves necessary to sort the warehouse. To our knowledge, a stacking problem with side access to bays from up to four access directions has not been studied yet.

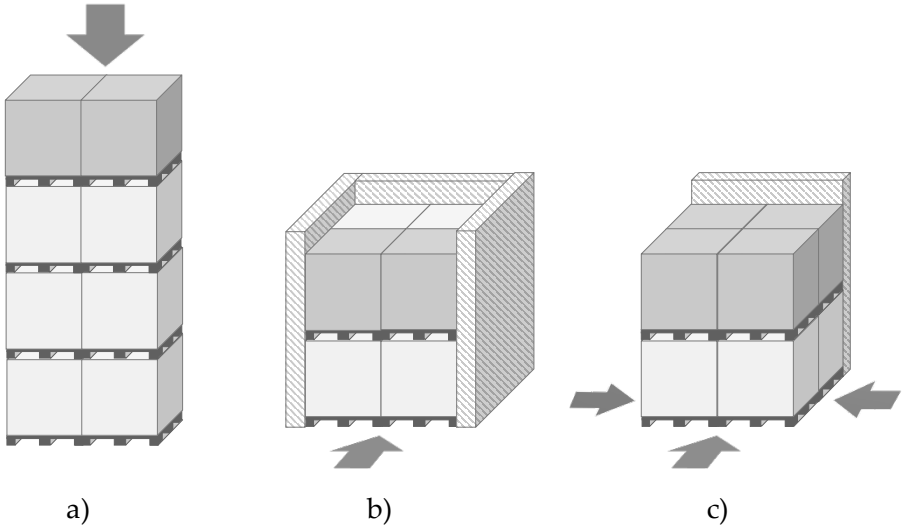


Fig. 4.1 Example storage bays with eight unit loads in the CPMP via crane in a) as well as side access in the UPMP via one direction in b) and three directions in c). The directly accessible unit loads are dark-gray.

As the main contributions in this chapter, we (1) formally introduce the UPMP and propose solution approaches for finding a minimum number of unit-load moves to sort a block stacking storage system. We (2) show that for the UPMP with one access direction, it is possible to transfer existing solution approaches from the CPMP. For cases with multiple access directions, we (3) develop a novel two-step solution approach based on a network flow model and an adapted search procedure that is applicable in all scenarios with up to four access directions. The proposed tree search methods and lower-bound heuristics based on algorithms from the CPMP [see TPV17] solve the UPMP to optimality. Finally, our contribution is complemented by (4) a publicly available benchmark set and experimental evaluation of the proposed tree search methods.

This chapter is organized as follows. We first introduce the UPMP in Section 4.1. This is followed by a literature overview in Section 4.2. Section 4.3 and Section 4.4 describe our solution approaches for one and up to four access directions, respectively. In Section 4.5, we present our UPMP benchmark dataset and provide the results of our computational experiments. The impact

of multiple access directions are discussed in Section 4.6. Finally, we conclude and point out future research directions in Section 4.7.

4.1 The unit-load pre-marshalling problem

The UPMP involves sorting a bay of a block stacking storage warehouse until all blockage is resolved. We assume that the current warehouse configuration with the position and retrieval priority group for each unit load is given and that no unit loads enter or leave the warehouse during planning/rearranging. The allocation of unit loads to a corresponding retrieval priority group is based on a known or predicted future retrieval time (see also the duration-of-stay storage strategy in Goetschalckx and Ratliff). Unit loads can only be accessed by AMRs at the outermost stacks if a free path from at least one of the four cardinal directions is available. Direct access may be blocked by further unit loads, walls or other obstacles, as well as due to limitations of the load carriers. Blockage occurs if unit loads of a higher retrieval group are placed in front of a unit load of a lower retrieval group (lower retrieval groups have to be removed earlier). We define such unit loads as *blocking*. The goal is to find a minimum number of unit-load moves to sort a storage bay such that all unit loads can be retrieved according to their retrieval group without any blockage. The resulting sequence of moves can be executed by one or multiple vehicles.

For the UPMP, we make the following assumptions with respect to the technical features of the block stacking warehouse. Since we only consider a single storage bay, the minimization of the number of moves is reasonable as an approximation of the sorting effort. We assume that process times for positioning and handling of AMRs dominate driving times. For larger warehouse setups with multiple bays, it would also be necessary to take into account travel times. Furthermore, we assume that robots can only carry a single unit load at a time and cannot reach over the top of a stack in front. If a stack has multiple tiers, only the top unit load can be accessed directly. This corresponds to the capabilities of standard industrial forklift trucks. Increasing the transport capacity to multiple items is a possible modification.

As shown in Fig. 4.2, a storage bay in the UPMP consists of a three-dimensional grid of storage locations with columns I, rows J and tiers T. Unit loads can only be placed above tier one if unit loads are stored underneath. Also, bays are filled up and sorted dependent on the access direction, preventing empty space or gaps in between. Whereas in the CPMP literature, a bay is usually considered as a single line of adjacent stacks [CSV11], the UPMP defines a bay as a configuration of $I \cdot J$ stacks.

The defined access directions have a major impact on the UPMP that will be discussed in this paper. Fig. 4.3 gives an overview of the possible access variants. For all access variants with more than one access direction, storage

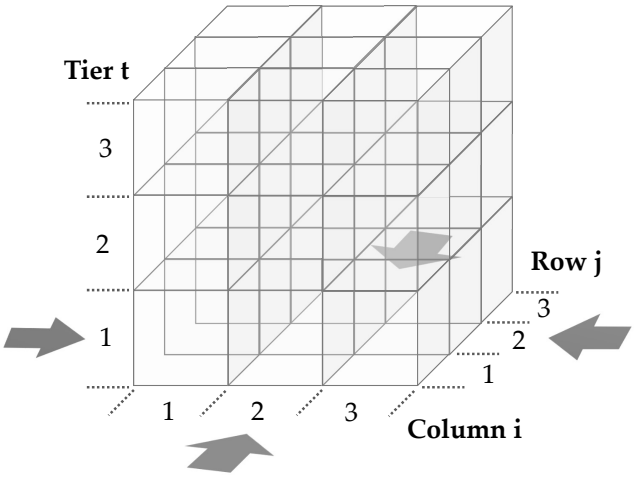


Fig. 4.2 State representation of the UPMP.

locations can theoretically be accessed from any available direction. Hence, it must be decided from which direction a unit load is picked and also from which direction it is placed on a storage location.

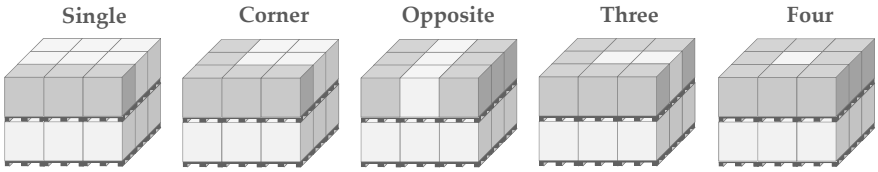


Fig. 4.3 Overview of the different access variants (excluding symmetrically similar combinations). The directly accessible unit loads are dark-gray.

Fig. 4.4 shows a simple example of a configuration with $I = 3$, $J = 2$ and $T = 2$ along with a possible solution of the UPMP with one access direction. Retrieval groups $g_{ijt} \in \{1,...,G\}$ from one to five are assigned to each unit load in the storage bay at their respective position ($i \in \{1,...,I\}$, $j \in \{1,...,J\}$, $t \in \{1,...,T\}$). The initial configuration in Fig. 4.4 a) is sorted according to the retrieval groups by a sequence of two consecutive moves. The unit load of group 3 cannot be moved in front of the groups 1 and 2 of the other two lanes. So in a first move, we need to free up a storage location for the blocking unit load of group 3 and reach the new configuration in Fig. 4.4 b). A second move leads to Fig. 4.4 c). This is the final, sorted configuration of the bay.

The main idea of unit-load pre-marshalling is to resolve blockages in idle time so that time-consuming relocation moves can be avoided later on in peak hours. Hence, the available space of block stacking storage systems can be used more efficiently. The UPMP is an operational problem, which may

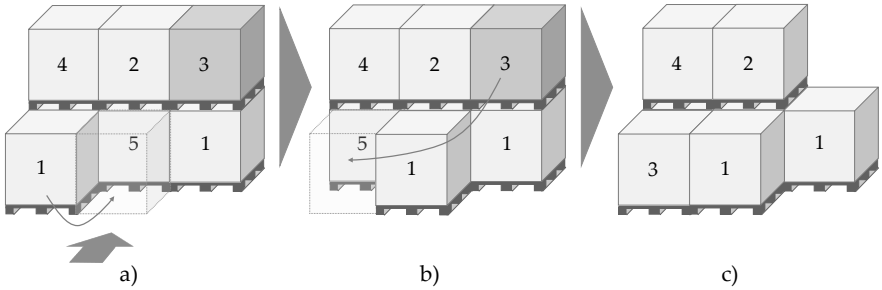


Fig. 4.4 Example configuration and solution of the UPMP.

arise several times throughout the day, overnight or less frequently (e.g., on the weekend). Given the proven complexity of the CPMP, which is NP-hard [see CSV11], the similar UPMP with one access direction is also NP-hard. The cases with more than one access direction are a generalization of the problem. Further investigation on the complexity of these problem variants is required.

4.2 Related work

Relocation problems in many types of storage systems have been widely studied. Well-known are container handling problems in maritime transportation and stacking problems in the steel industry. Besides relocation problems in maritime transportation and the steel industry, a variety of further problems have been addressed for many different storage systems. We give an overview of most relevant research related to our problem in AS/RS, RMFS, block stacking warehouses, general warehousing, puzzle-based storage systems and other problems. Caserta, Schwarze, and Voß [CSV11] specify the three types of post-stacking problems: the remarshalling problem (intra-block with moves between bays), the premarshalling problem (intra-bay), as well as the relocation problem (for retrieval). Lehnfeld and Knust [LK14] classify stacking problems into storage loading, unloading, pre-marshalling, combined and further problems. Most studied relocation problems are mainly related to storage unloading (retrieval) and pre-marshalling. However, an increasing amount of publications consider further problems like re-marshalling, integrated problems as well as online shuffling problems at runtime. In Section 3.4 we give an overview of the most relevant studied relocation problems in associated domains and classify them into storage loading problems, pre- and re-marshalling problems, (integrated) shuffling-at-runtime problems as well as storage retrieval problems. Storage loading in terms of storage location assignment is not necessarily a relocation problem. However, it is related since a storage system can already be sorted at the loading stage and it directly affects other relocation problems for marshalling and retrieval.

4.2.1 Maritime transportation

Most relevant for this work, also because of the similarity of the problem, is research related to container pre-marshalling problems. The CPMP is a very well studied problem concerning the reshuffling of a set of containers in a single storage bay during idle time. An overview and categorization of optimal pre-marshalling algorithms in this domain is presented by Tierney, Pacino, and Voß [TPV17] and Tanaka and Tierney [TT18]. Methods include integer programming, dynamic programming, constraint programming and several tree search methods. Lee and Hsu [LH07] present an integer programming model for solving the CPMP to optimality based on a formulation of a multi-commodity network flow problem. The size of the model and the unknown number of time periods, which must be specified in advance are problematic. Parreño-Torres, Alvarez-Valdes, and Ruiz [PAR19] develop several integer programming models that outperform previously presented mathematical models. Prandtstetter [Pra13] uses a branch and bound algorithm based on dynamic programming utilizing state memoization to avoid revisiting already explored nodes. More recently, Jiménez-Piqueras et al. [Jim+23] present several constraint programming models based on an iterative algorithm. Rendl and Prandtstetter [RP13] formulate a constraint programming model for the CPMP as well as the robust CPMP and introduce a search heuristic, where they try to identify the most promising moves. Expósito-Izquierdo, Melián-Batista, and Moreno-Vega [EMM12] introduce the application of an A* tree search algorithm for solving the CPMP using a simple lower bound that only counts misplaced items. Bortfeldt and Forster [BF12] present a heuristic tree search procedure that uses a novel lower bound for the number of moves. This bound also considers the available storage locations (supply) and required storage locations (demand) for each retrieval priority group. Tierney, Pacino, and Voß [TPV17], Tanaka and Tierney [TT18] and Tanaka et al. [Tan+19] build on this bound and extend it for many different situations. Tierney, Pacino, and Voß [TPV17] propose several branching and symmetry breaking rules combined with A* and IDA* algorithms. Tanaka and Tierney [TT18] developed an iterative deepening branch and bound algorithm, which is extended by Tanaka et al. [Tan+19] with an improved lower bound, new dominance rules as well as memoization and tie breaking algorithms. The number of optimally solved instances of the CPMP for known datasets increases significantly with each of their publications. Apart from optimal algorithms, a vast amount of heuristic solution procedures including deep-learning assisted tree search (Hottung, Tanaka, and Tierney[HTT20]) have been developed.

Besides the CPMP, the CRP or blocks relocation problem is also an important relocation problem in maritime transportation. In the CRP the objective is to find a minimum number of reshuffling moves for the retrieval of items.

The problem does not consider incoming goods and has a decreasing branching factor due to containers leaving the warehouse. For the CRP similar solution methods like IDA* algorithms (e.g., Zhang et al. [Zha+10], Zhu et al. [Zhu+12]) or branch and bound search with some shared properties (e.g., Tanaka and Takii [TT16]) are used. In comparison, research regarding the re-marshalling and stack loading problem is scarce. For intra-block re-marshalling (moving containers between multiple bays), we refer to the review of Caserta, Schwarze, and Voß [CSV11] and the publication of Covic [Cov17], who compares different features of existing re-marshalling algorithms with his own contribution. Lehnfeld and Knust [LK14] provide an overview of few publications regarding stack loading problems. Boysen and Emde [BE16] introduce the parallel stack loading problem, where incoming items need to be moved into parallel stacks minimizing misplaced items based on retrieval priorities.

4.2.2 Steel industry

Also in the steel industry shuffling problems need to be solved in warehouse operations. Several authors have analyzed different kinds of problems, which Tang, Zhao, and Liu [TZL12] categorize into plate shuffling problems and coil shuffling problems (CSP). Plate shuffling includes the slab stack shuffling problem, where requested slabs are selected out of a feasible set of slabs with the goal of minimizing shuffling moves (Tang et al. [Tan+02]). König et al. [Kön+07] describes a steel slab stacking problem, where incoming items have to be processed within a time window and moved to buffer stacks or target stacks, which represent the exact retrieval order sequence. If items cannot be directly placed in the correct position of a target stack, two additional types of moves are possible. Items can be moved between buffer stacks or from a buffer stack to a target stack. The goal is to find the minimum number of moves to build all target stacks given an initial configuration. The CSP (see [ZW06]; [TZL12]; [Tan+15]) deals with piled up steel coils in cylindrical form, which structurally support each other. Thus, one steel coil is placed on top of two coils in the row underneath. Ge et al. [Ge+20] introduce the slab pre-marshalling problem. In contrast to CPMP, the goal is to sort items and generate stacks of the same item group. This way fast access times can be guaranteed without considering the duration of stay. However, stacks containing only one item group are only possible, if the amount of item groups is smaller then the number of stacks. Therefore, final configurations may contain stacks with badly placed items of another item group. Ge et al. [Ge+21] integrate slab pre-marshalling and storage moves to find configurations with a minimum number of badly placed items requiring the least amount of stacks. The problem is defined as the integrated optimisation of storage and pre-marshalling moves.

4.2.3 Automated storage / retrieval systems

Shuttle-, Crane-, Vehicle- or Robot-based AS/RS are widely used in corporate practice and the associated decision problems have been investigated for many years. Linn and Wysk ([LW90a]; [LW90b]) introduce a system framework considering the most relevant decision problems. Among other control policies, a restoring policy for repositioning of loads in idle time is proposed. Thereby, items are removed and restored according to turnover-based rules. Muralidharan, Linn, and Pandit [MLP95] represent the shuffling problem in an AS/RS, where pallets are relocated in idle time, as a network model. They formulate the problem as a selective travelling salesman problem and present the two heuristics: shuffling with nearest neighbor and shuffling with insertion, where a tour route is calculated to maximize profit in terms of saved travel time until a time limit for shuffling is reached. Carlo and Giraldo [CG12] consider the problem of how an optimal storage assignment can be reached based on the current storage positions by incrementally updating the storage position when loads have been requested by a picker and are going to be restored in the AS/RS running dual command cycles. Since it would take a long time until all loads are requested by a picker and it is unlikely that the orders are in the right sequence to obtain an optimal storage assignment, they also present a scenario with one empty location where the storage/retrieval machine is idle and can perform any move. Reshuffling of items is even more important for AS/RS with multi-deep storage or for robot-based compact storage and retrieval systems with stacks like AutoStore, where each robot has also lifting capabilities (see [ZKX16]).

4.2.4 Robotic mobile fulfillment systems

Since the concept of RMFS has been introduced by Wurman, D'Andrea, and Mountz [WDM08] with Kiva systems, it has become standard in many e-commerce fulfillment centers. RMFS are automated parts-to-picker systems, where mobile racks (pods) are moved by robots and brought to the picking stations. The unique feature of RMFS is that the robots are able to drive under the racks. Merschformann [Mer18] classifies relocation problems in RMFS into active and passive repositioning. Passive repositioning is about updating a storage location when pods are returned to storage after picking. Under this strategy pods which are not requested will not be relocated and might block attractive storage locations. Hence, active repositioning allows to move pods from one to another storage location without visiting a picking station. Merschformann [Mer18] determines the attractiveness of storage location based on their shortest path to a picking station and calculates the difference between actual and desired storage location. Two scenarios are analyzed: Utilizing idle time at night or performing active repositioning in parallel to a running system. The results show that active repositioning in

idle time is especially helpful, if the system is not already performing passive repositioning. Not using passive repositioning allows the system to achieve a better performance by using a storage policy like nearest open location, when pods are brought back from a picking station. Passive repositioning shows negative effects for most cases in a scenario where the storage system is continuously well utilized. Krenzler, Xie, and Li [KXL18] define the deterministic Pod Repositioning Problem for passive repositioning and propose several algorithms. Relocation of pods in RMFS becomes especially important for layouts with multi-deep storage bays, where blockages are possible (see e.g., [JYD20]; [YJD21]). Yang, Jin, and Duan [YJD21] indicate that in multi-deep RMFS the time of moving obstructing shelves consumes a large amount of the total working time.

4.2.5 Block stacking warehouses

In terms of block stacking warehouses there are some recent publications about relocation problems. Lee et al. [Lee+06] introduces the dynamic block stacking problem with random demand where the optimal row depth is determined and changed over time by using a discrete-time infinite-horizon Markov decision process model. Thereby, unit loads are relocated, if the benefits like reduced retrieval time and improved space utilization exceed the relocation costs. One of the intentions is to reduce the honeycomb loss, which occurs when a storage lane is dedicated to a product and not completely full. The applied reward function considers floor space costs as well as travel costs for replenishment, retrieval and relocation. Maniezzo, Boschetti, and Gutjahr [MBG21] investigate the stochastic pre-marshalling problem in block stacking warehouses, where all stacks are directly accessed (e.g., via crane or shallow storage bays). Their research addresses the stochastic case, where the retrieval orders are not known in advance. Based on historical data the expected cost of the recourse action is calculated for any given configuration. The recourse costs are all costs for processing the orders of the previously unknown picking list. The authors develop a dynamic programming model and combine it with a corridor heuristic as well as a multiheuristic approach to solve larger instances. The results show that a small amount of pre-marshalling effort significantly reduces the number of relocation moves for retrieval operations.

4.2.6 General warehousing

Some publications address relocation problems for many types of warehouses and storage systems. Kofler et al. [Kof+11] differentiate between construction and improvement approaches to rearrange a warehouse. Construction approaches (called re-warehousing), determine the best storage location assignment from scratch and are used to rearrange the whole warehouse. They require a large amount of time and resources. However, over time seasonal

factors and an altering product mix may change the demand profile and affect warehouse efficiency. Improvement approaches (called healing) continuously improve and cleanup existing warehouse configurations to achieve a better total storage assignment.

One of the first publications concerning warehouse rearrangement by Christofides and Colloff [CC73] assumes a given and final arrangement of the warehouse as known and propose a two-stage algorithm based on moving cost to minimize relocation cost. The warehouse rearrangement process is performed in cycles for at least two items, that exchange positions and are sequentially moved. Pazour and Carlo [PC15] consider the same problem (called the warehouse reshuffling problem) and relax the assumptions regarding the handling of cycles. This reduces the reshuffling costs and allows open locations to be different in the final warehouse arrangement. Chabot, Coelho, and Renaud [CCR18] also investigate a similar problem as Christofides and Colloff [CC73] (called the warehouse reassignment problem) with the main difference, that unit loads can also be directly relocated to currently occupied locations. Unit loads are switched by dropping a new unit load on the floor next to the target storage location, removing the other unit load, moving the new unit load into the storage location and picking again the other unit load. Since this increases handling effort, a penalty is added. This way, unit loads are always moved directly from a current position to a target location without any intermediate moves. The goal is to minimize the total working time of the operators. As a solution technique, they present three different exact mathematical formulation methods. Kofler et al. [Kof+11] present both, construction and improvement approaches and compare them for picker-to-part scenarios. The authors propose local search and simulated annealing algorithms based on the pick frequency/part affinity score (item affinity measures whether items are often ordered together) as improvement approach for randomly generated warehouse instances.

4.2.7 Puzzle-based storage systems

Finding the minimum number of reshuffles for a given set of items to reach a goal state is a well known problem in artificial intelligence (e.g., blocks world planning, Tower of Hanoi puzzle or 8 and 15 puzzle problem). In blocks world, an initial state and final state of blocks on a table is given. The goal is to find the minimum number of moves to reach the final state. The problem is NP-hard (Gupta and Nau [GN92]). The Tower of Hanoi is a disk-based puzzle with disks of various diameters, which can be moved onto three different pegs. It starts with one pyramidal stack from which disks are moved to the other pegs with the goal of rebuilding the initial stack. Disks are never allowed to be placed on a disk with a smaller diameter (see e.g., Ball [Bal93]). Puzzle-based storage systems are inspired by the well-known 8

and 15 (slide) puzzle problem, where only one location within a grid-based storage system is empty. Items are moved to the currently empty location creating a new open location until a target configuration is found or an item can be retrieved. Gue and Kim [GK07] compare aisle-based and puzzle-based storage systems in different configurations. While the slide-puzzle concept achieves very dense storage configurations, the expected retrieval time as well as the moving costs are generally high, even for systems with multiple escorts (empty locations).

4.2.8 Other related problems

Furthermore, related problems in areas like sequencing and reshuffling of train wagons (mentioned by Caserta, Schwarze, and Voß [CSV11]), resequencing of assembly lines (mentioned by Boysen and Emde [BE16]) or sorting of vectors and disk fragmentation (mentioned by Carlo and Giraldo [CG12]) have been identified.

4.3 Solving the UPMP with a single access direction

We present a search procedure based on the A* algorithm to solve the UPMP with one allowed access direction to optimality, analogous to the approaches presented by Expósito-Izquierdo, Melián-Batista, and Moreno-Vega [EMM12] and Tierney, Pacino, and Voß [TPV17] for the CPMP. These tree-based solution approaches are known for their well performance for the similar CPMP. We first describe the A* algorithm implemented branching rules, followed by the adapted lower bound.

In the case of a single access direction, the columns represent lanes, rows refer to the depth of the bay (number of stacks per lane) and tiers refer to the height of the bay. The number of columns, rows, and tiers is given by I, J, and T respectively. Fig. 4.5 a) shows a small configuration with $I = 4$, $J = 2$ and $T = 2$. Each column i can be translated into an access sequence of unit loads in Fig. 4.5 b). The tiles represent a storage position with the assigned group $g_{ijt} \in \{1,...,G\}$ of each unit load. All empty positions are set to $g_{ijt} = 0$. All tiles of blocking unit loads that block other unit loads are colored in dark-gray.

For one access direction, a column with a length of two stacks $J = 2$ and two tiers $T = 2$ results in a sequence of four storage locations that can be retrieved one after the other. In the first column at $i = 1$ of Fig. 4.5 b), the unit load at position ($i = 1$, $j = 2$, $t = 2$) is blocking, since it has a higher priority group than the unit load on the storage location below ($i = 1$, $j = 2$, $t = 1$) with $g_{1,1,2} > g_{1,1,1}$. All unit loads that are placed at a storage location before a blocking unit load are also blocking, because they need to be moved for resolving all blockage. Hence, the unit loads at the positions ($i = 2$, $j = 1$,

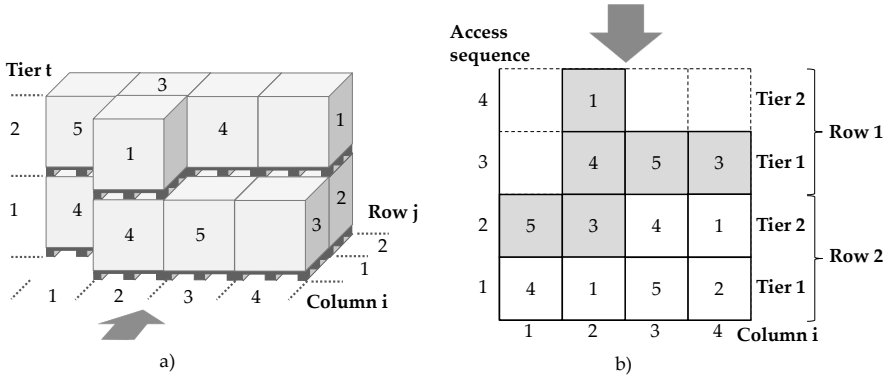


Fig. 4.5 A simple example of a configuration for one access direction with a three-dimensional view on the left a), and translated into an access sequence at column i on the right, b). Dark gray colored tiles represent blocking unit loads.

$t = 2$), ($i = 2$, $j = 1$, $t = 1$), ($i = 2$, $j = 2$, $t = 2$), ($i = 3$, $j = 1$, $t = 1$) and ($i = 4$, $j = 1$, $t = 1$) are blocking as well.

4.3.1 A* for the unit-load pre-marshalling problem

The A* algorithm uses a best first search guided by a heuristic to solve planning problems. Starting from an initial bay configuration all possible moves (branches) of each movable unit load are examined. Iteratively the most promising move that minimizes costs is extended. The cost estimation heuristic for branch selection is given by $f(n) = g(n) + h(n)$, where n represents a search node, $g(n)$ is the number of performed moves at node n, and $h(n)$ is a lower bound on the number of moves needed to reach a configuration from node n without any blocked unit loads.

Fig. 4.6 illustrates the search procedure with an example based on counting blocking items as a simple lower bound. Starting from an initial configuration with three blocking items, there are four possible moves, leading to four nodes all with a search depth of one. These newly generated nodes are called *open nodes*, and we calculate $f(n)$ for each one. We explore the node with the lowest value of $f(n)$, which in this example is the configuration with $f(n) = 3$. We apply tie breaking rules and symmetry breaking that is described in more detail later.

Algorithm 1 shows the A* algorithm for the UPMP. It starts with the initialization of an empty open list where all unexplored nodes are stored, an open_lookup dictionary mapping the unexplored nodes to f-values and a closed set collecting all previously visited nodes. Whereas all unexplored nodes are stored in the open list, the closed set stores all previously visited nodes. The open_lookup dictionary is used to improve the lookup of minimum f-values of already visited configurations. Initially, the open_lookup

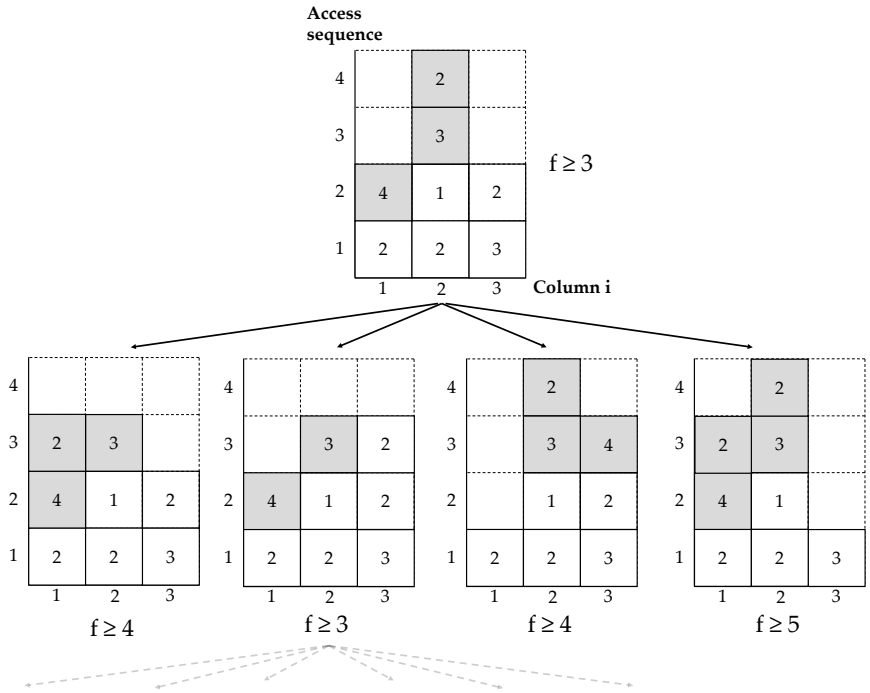


Fig. 4.6 Example of the tree search procedure with the cost function $f(n)$ (simple lower bound as blocking items in dark gray) similar to the illustration in Tierney, Pacino, and Voß [TPV17].

dictionary contains only the starting configuration (root node) of the storage bay. Unlike the root node, each node object contains an attribute that allows a link to its parent node object.

After initialization, the algorithm starts to search for a solution within the main loop and stops when an optimal solution without any blocking unit loads has been found or when the open list becomes empty, meaning no solution exists. While nodes are in the open list and the time limit has not been reached, a node n with the minimum cost estimation $f(n)$ is popped off the priority queue. If n has already been visited and is in the closed set, then the node is skipped, as we already know the shortest path to get to it. Otherwise, the node is added to the closed set and the successor nodes for every possible move of a unit load are calculated, as long as they are not pruned due to branching rules (see Section 4.3.2). Successor nodes are added to the priority queue and open_lookup dictionary if there is not already a node with the same configuration and a smaller cost in the closed set or open_lookup dictionary. This search procedure guarantees an optimal solution as long as the heuristic used in the cost estimation is an admissible lower bound on the true cost of the configuration.

Algorithm 1: A* algorithm for the UPMP adapted from Tierney, Pacino, and Voß [TPV17].

```
Function astar(root):
    open ← PRIORITY_QUEUE();
    open_lookup ← DICT();
    closed ← ∅;
    PUSH(open, root);
    open_lookup[root] ← f(root);
    while |open| > 0 and time < timeout do
        n ← POP(open);
        if n in closed then
            continue;
        end
        closed ← closed ⋃ {n};
        if blocking(n) = 0 then
            return n;
        end
        successors ← BRANCHING(n);
        for s ∈ successors do
            if s ∉ closed and (s ∉ open_lookup or open_lookup[s] > f(s)) then
                PUSH(open, s);
                open_lookup[s] ← f(s);
            end
        end
    end
```

4.3.2 Branching

For each iteration of the search, our algorithm calculates all feasible successors of the current node, i.e., configurations that can be reached by a single move of a unit load from the current configuration. Thereby only moves to the furthest location per lane are allowed in order to prevent any holes even though placing a unit load on other locations is technically possible. Holes are detrimental as they lead to less free storage locations in front of the items and, therefore, to infeasible configurations. Still some of these configurations make no sense to examine, for example given node n, we should not reverse the move that brought us to node n. Besides the avoidance of move reversals, we further use *memoization* to remember all seen configurations so that we do not explore the same configuration multiple times unless we find a cheaper way of reaching a configuration. Therefore, we use a dictionary for look-ups of the total costs f-value of already encountered configurations. Tierney, Pacino, and Voß [TPV17] and Tanaka and Tierney [TT18] present further symmetry breaking rules (e.g., transitive move avoidance or empty stack symmetries) and heuristics for branching (e.g., only perform moves that place containers in non-blocking positions). They may also be beneficial here, as there are

usually many nodes that have the same f value, thus one needs a way to differentiate between them.

4.3.3 Lower bound on the number of moves

In this section, we give an overview of the adapted lower bound from the CPMP and provide an example. The notation is summarized in Table 4.1. For mathematical proofs of correctness, we refer to the respective publications.

Table 4.1 Adopted notation for the lower-bound heuristics from Bortfeldt and Forster [BF12].

$n^b(i)$	Number of blocking (badly placed) unit loads of the lane at column i
$d(g)$	Number of blocking unit loads represents demand of priority group g, i.e., no. of blocking unit loads of group g
$D(g)$	Cumulative demand of a priority group g ($d(g) + d(g+1) + \cdots + d(G)$)
$s(g)$	Number of supply slots for priority group g, i.e., no. of empty storage locations for group g
$S(g)$	Cumulative supply of a priority group g ($s(g) + s(g+1) + \cdots + s(G) + T \cdot n_{l,\text{empty}}$; $n_{l,\text{empty}}$ˇnumber of empty lanes)
$DS(g)$	Cumulative demand surplus of priority group g ($D(g) - S(g)$)

We adopt the classification of possible moves introduced by Bortfeldt and Forster [BF12] into bad-good (BG), bad-bad (BB), good-good (GG) and good-bad (GB) moves. The notation indicates whether a unit load is blocking (bad) or well-placed (good) before and after a move. bad-good and bad-bad (BX) moves consist of BG and BB moves, whereas good-good and good-bad (GX) moves correspond to GG and GB moves.

Simple lower bound

The first valid lower bound for the minimum number of item moves in (4.1) simply counts the number of blocking unit loads of the current bay configuration (see [LC09]; [EMM12]). Since the unit load of higher priority will be retrieved first, at least all unit loads in front of it must be relocated. In the best case, all unit loads are relocated as BG moves from badly to well placed. In Fig. 4.5 b), the total count of blocking unit loads is $n_b = 6$.

$$n_b = \sum_{i \in I} n_b(i) \tag{4.1}$$

Extension by Bortfeldt and Forster [BF12]:

The lower bound for the number of moves from Bortfeldt and Forster [BF12] is defined as the sum of the number of BX and GX moves in (4.2) necessary to reach a final bay configuration without any blockages.

$$n_m = n_{BX} + n_{GX} \tag{4.2}$$

The first extension to the simple lower bound is that to carry out a BG move, it is necessary to have at least one lane without any badly placed unit loads. Otherwise, at least the lane with the smallest number of badly placed items has to be moved as a BB move (see (4.3)).

$$n_{min} = \min\{n_b(i)\} \tag{4.3}$$

Therefore as shown in (4.4), the number of BX moves is the sum of the blocking unit loads (simple lower bound n_b) and the minimum number of blocking unit loads of all stacks n_h.

$$n_{BX} = n_b + n_h \tag{4.4}$$

In Fig. 4.5 b), all lanes contain badly placed items. Three lanes contain only one badly placed item. Therefore, $n_h = 1$ and a single additional move is necessary.

The second extension refers to the number of GX moves, which involve moving well-placed unit loads. By calculating the required slots for badly placed unit loads (demand) as well as the available slots (potential supply) for each priority group, we derive a cumulative demand surplus. A slot is a storage location defined by the coordinates i, j and t. The required demand $d(g)$ is computed by counting the number of badly placed items for each priority group. Table 4.2 shows $d(g)$ for the example in Fig. 4.5. The cumulative demand $D(g)$ is calculated from the lowest to highest priority groups.

Once $D(g)$ has been determined, the supply slots $s(g)$ must be computed. The supply slots of a lane at column i consist of the slots before the last well-placed item that do not need to be moved. The available slots for the priority group g of the last well-placed item of each lane are added to the potential supply slots. For the example in Fig. 4.5 b) and Table 4.2, there are five potential supply slots, respectively, above $g = 4$ (lanes 1 and 3) and $g = 1$ (lanes 2 and 4). Again, the cumulative values $S(g)$ are calculated.

Finally, the resulting cumulative demand surplus is the difference of cumulative demand subtracted by the cumulative supply, since the unit loads with higher priority can always be stored on slots with lower priority. A positive cumulative demand surplus means that there are not enough available slots by just moving badly placed items. In this case, additional GX moves are necessary. The necessary GX moves are calculated by first deriving the maximum demand surplus for all priority groups $DS(g^*)$ via (4.5) and (4.6). In our example in Table 4.2 the maximum demand surplus $DS(g^*)$ is 2.

$$DS(g^*) = max\{DS(g)\} \tag{4.5}$$

$$g^* = argmax\{DS(g)\} \tag{4.6}$$

Item group g	Demand d(g)	Cumulative demand D(g)	Potential supply s(g)	Cumulative potential supply S(g)	Cumulative demand surplus DS(g)
5	2	2	0	0	**2**
4	1	3	5	5	-2
3	2	5	0	5	0
2	0	5	0	5	0
1	1	6	5	10	-4

Table 4.2 Calculating demand surplus based on the example in Fig. 4.5 b).

We use the cumulative demand surplus $DS(g^*)$ to calculate the potential GX lanes n_{GX}^{pot}, where additional slots for unit loads of priority groups $g \geq g^*$ can be created. Slots for these priority groups can be created by relocating correctly placed unit loads of a priority group $g < g^*$. This means a potential GX lane must contain correctly placed unit loads of a smaller priority group. The correctly placed unit load of the smallest priority group of each lane corresponds to the outermost correctly placed item. Empty lanes are not potential GX lanes, since their supply is already fully taken into account with the highest priority group and no additional slots can be created. The necessary number of GX lanes must be at least n_{GX}^{nec} according to (4.7) with a maximum of $J \cdot T$ available slots per lane, if a whole lane has to be cleared. This is a slight modification compared to the approach for the CPMP with only T slots per stack.

$$n_{GX}^{nec} = max(0, [DS(g^*)/(J \cdot T)]) \tag{4.7}$$

The number of potential GX lanes n_{GX}^{pot} must be greater or equal than the necessary number of GX lanes n_{GX}^{nec} to find a feasible solution. If this is the case, the lanes are sorted by $n^{g^*}(i)$ in ascending order. $n^{g^*}(i)$ is the number of correctly placed unit loads with priority group $g < g^*$ of a lane at column i. In our example of Fig. 4.5 b), lanes 1, 2 and 3 contain one correctly placed item and lane 4 contains two correctly placed items smaller than $g^* = 5$. This leads to the number of potential GX lanes $n_{GX}^{pot} = 4$ and we get the sorted list

of correctly placed items that have to be moved [1,1,1,2]. The lower bound for the number of GX moves n_{GX} is calculated in (4.8).

$$n_{GX} = \sum_{j=1}^{n_{GX}^{nec}} n^{g^*}(i) \tag{4.8}$$

For the example in Fig. 4.5 b), we round the result of $max(0,[2/(2*2)]$ up to $n_{GX}^{nec} = 1$ and derive $n_{GX} = \sum_{j=1}^{1} n^{g^*}(i) = 1$.

Further improvements for cases in which the lower bound can be increased have been proposed in [TT18] and [Tan+19]. These improvements can be transferred analogously, but due to their complexity, we omit them from this study.

4.4 Solving the UPMP with multiple access directions

In storage bays with multiple access directions, each stack can generally be accessed from any allowed access direction. Multiple access directions are beneficial and challenging at the same time. An advantage of multiple access directions is that potentially fewer relocation moves are required to sort the storage bay until each unit load can be accessed from one access direction without blockage. However, difficulties are a tremendous increase in the branching factor as well as the size of the solution space and a higher complexity for configuration assessment.

To overcome these challenges, we propose a solution approach in which we initially determine the access direction for each stack and fix this direction until the pre-marshalling process is finished (access direction fixing). This allows us to benefit from a significantly reduced number of required moves and, at the same time, achieve fast runtimes. Access direction fixing makes also sense from a practical point of view. We will discuss this together with the implications of our assumption at the end of the section.

For solving the UPMP with multiple access directions, we present a novel two-stage algorithm shown in Fig. 4.7. First, a network flow model is used to initially determine the access direction fixing for each stack that minimizes the number of blocking items. This configuration is transformed into a representation of *virtual lanes* similar to the single access direction case and is used throughout the search. A virtual lane consists of 1 to max(I, J) successive stacks in a row/column with a defined access direction. Since the length of each virtual lane can be different, we introduce an adapted lower bound with a MIP formulation to determine the minimum number of required reshuffling moves.

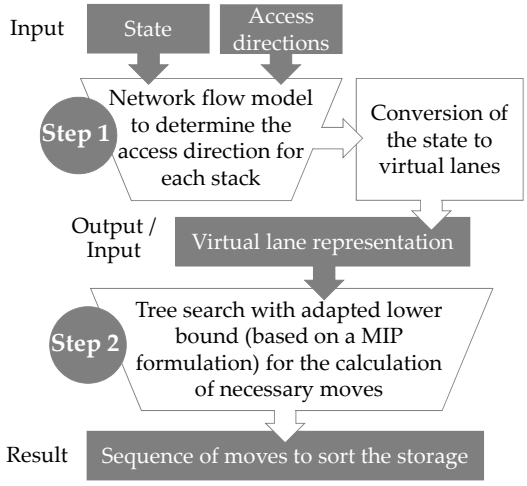


Fig. 4.7 Illustration of the main components of our solution approach with the respective input and output.

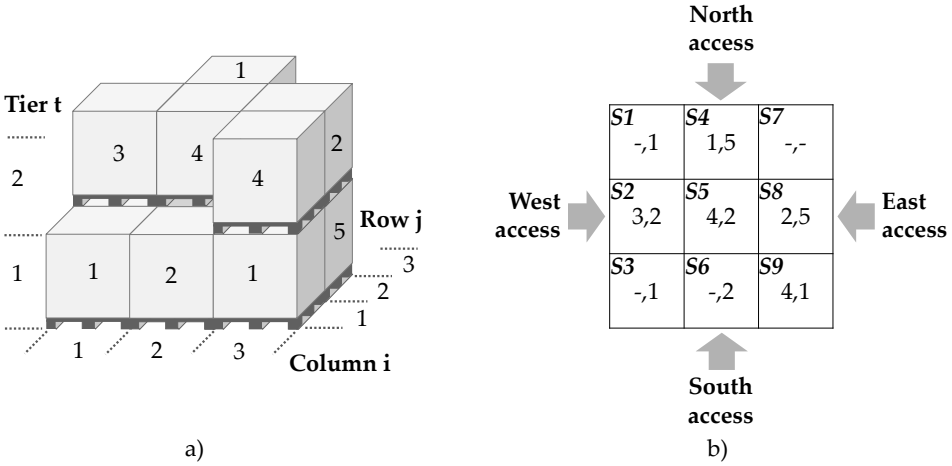


Fig. 4.8 Example of a bay configuration with four access directions in a). Translated into a top view in b), each tile in the grid represents a stack with a tuple of multiple tiers. Hyphens indicate empty storage locations.

4.4.1 Step 1: Access Direction Fixing

We first describe the components of the mathematical model for calculating the access direction for each stack and then provide an example based on Fig. 4.8. Our model captures the grid-based layout of a block stacking storage system and considers simple rules of feasible AMR movements. Table 4.3 shows the sets, parameters, and decision variables that we use throughout this section. Fig. 4.8 a) shows a simple example of the configuration with $I = 3$

columns, $J = 3$ rows and two tiers $T = 2$. Fig. 4.8 b) translates the example into the associated top view. Each tile of the grid represents a stack $S1$ to $S9$ with a tuple of stacked unit loads (tier 2, tier 1).

Table 4.3 Notation of the network flow model.

Sets and Indices	
s, t	Indices of network vertices
d	Index of the neighbor vertex for an access direction ($\{north, south, east, west\}$)
n	Index of an orthogonal neighbor vertex of $x_{d,s}$
E	Set of all edges
E^x	Set of edges excluding edges from the origin o
V^x	Set of vertices excluding the origin o
E^{origin}	Set of edges from origin o
V^{inner}	Set of inner vertices representing inner stacks that cannot be directly accessed by one of the access direction vertices
Parameters	
$c_{s,t}$	Cost of visiting vertex t from adjacent vertex s
$b_s \in \mathbb{N}$	Demand at vertex s (1 for each stack that has to be visited)
$P \in \mathbb{N}$	Supply provided at origin o (equal to the number of stacks)
Variables	
$f_{s,t}$, $f_{t,s}$	Flow from source s to destination t and vice versa
$x_{s,t} \in \{0,1\}$	Equals 1, if a stack is accessed via source s to destination t
$x_{d,s} \in \{0,1\}$	Equals 1, if stack s is accessed via the edge of access direction d
$x_{s,n} \in \{0,1\}$	Equals 1, if the outgoing edge from stacks s to an orthogonal neighbor n of s is used

In our model, each stack is represented by a vertex with a directed edge from and to the adjacent stack vertices for any allowed access direction. For each of the allowed access directions (up to four) a vertex is added with edges pointing to the vertices of stacks that are placed at the edge of the bay. Finally, a vertex representing the origin o is connected to vertices for each access direction. The capacity of edges is unlimited and the assigned costs represent the incremental difference of blocking items between accessing the previous and the current stack in a lane. To avoid holes, i.e., items in front of empty locations, we add penalty costs to edges that may create holes and set these costs off on the edge to the last empty position.

Supply is provided at origin o for all of the stacks' vertices. Each vertex representing a stack must be visited once, and thus has a demand of one. The intermediate vertices for the access directions have zero demand. Edges

between vertices that are directly accessible by an access direction are not necessary, since the minimum costs in terms of blocking items are guaranteed through access from the respective direction. The flow represents the number of accesses via the respective edges by AMRs. The optimization objective is to minimize the costs of accessing all stack vertices (demand) from supply vertex o via intermediate vertices for the available access directions. The result is a defined access configuration with a minimum number of blocking unit loads.

The mathematical formulation of our network flow model is:

$$min \sum_{(s,t)\in E} c_{s,t} x_{s,t} \tag{4.9}$$

$$f_{s,t} \leq P x_{s,t} \quad \forall (s,t) \in E \tag{4.10}$$

$$\sum_{(s,t)\in E^x} f_{s,t} - \sum_{(t,s)\in E^x} f_{t,s} = b_s \quad \forall s \in V^x \tag{4.11}$$

$$\sum_{(s,t)\in E^{origin}} f_{s,t} = P \tag{4.12}$$

$$\sum_{(d,s)\in E} x_{d,s} + \sum_{(s,n)\in E} x_{s,n} \leq 1 \quad \forall s \in V^{inner} \tag{4.13}$$

$$x_{s,t} \in \{0,1\}, \quad f_{s,t} \geq 0 \quad \forall (s,t) \in E \tag{4.14}$$

The objective in term (4.9) is to minimize the movement costs for accessing all stacks. We use the binary variable $x_{s,t}$ to indicate whether an edge is used or not. The idea is that we want to count the number of blocking items only once, even if the stacks behind are accessed through the same stack. Constraints (4.10) ensure no edge flow exceeds the total amount of supply P. Constraints (4.11) and (4.12) ensure flow conservation for all vertices with a demand of zero and above, as well as flow conservation for supply in the origin o. We require Constraints (4.13) to avoid the AMRs turning within the grid. The constraints say that flow on orthogonal neighbor edges is not allowed for all inner vertices. Inner stacks are all stacks that cannot be directly accessed by one of the access direction vertices. We note that the number of constraints to avoid turning increases with the grid size (up to eight per inner stack). If the AMRs are able to move around corners in the grid, these constraints can be neglected. Our mathematical model is similar to a min-cost flow formulation, however, we use binary variables instead

of flow variables in our objective function because the costs should only be considered once and add additional constraints in (4.13) for AMR movements. Finally, constraints (4.14) restrict the variables $x_{s,t}$ to be binary and ensure that the flow $f_{s,t}$ is positive.

Example: We now consider the example given in Fig. 4.8 and derive a representation for the network flow model. All stacks represented by tiles are translated into vertices S1 to S9 in the network flow representation in Fig. 4.9 a). For each access direction a vertex N, S, W and E is added, as well as a vertex for the origin o. Edges between the outermost stacks of the bay for any allowed access direction are not necessary. These stacks can be directly accessed from the respective direction. Each edge is assigned costs that are equal to the incremental cost of accessing the corresponding vertex. In our example, accessing stack $S2$ from W costs one due to one blocking item. Now accessing stack $S5$ from stack $S2$ would add another two blocking items, as both items of stack two would have to be moved plus the top item of stack $S5$. Supply is provided at origin o for all stacks (in our example: 9 vertices). Each stack must be visited once and therefore has a demand of 1. The intermediate vertices for the access directions have zero demand. Fig. 4.9 b) illustrates a solution for a lower bound of blocking unit loads showing only used edges.

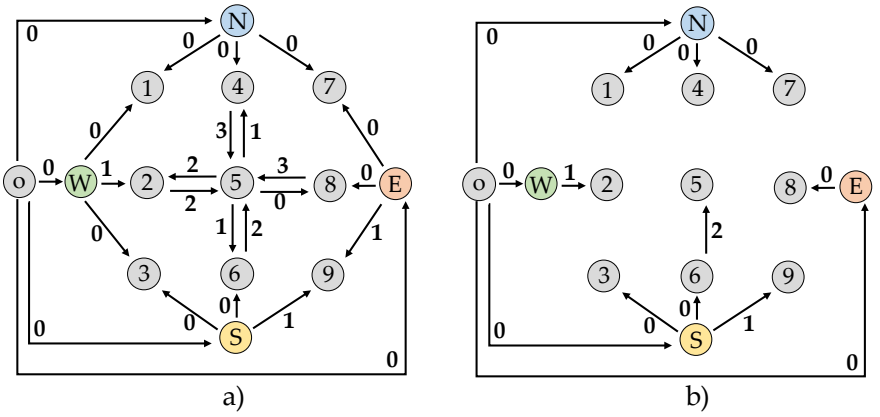


Fig. 4.9 Network flow scheme for the example with four access directions (each access direction vertex has a different color) of Fig. 4.8 in a) and a possible solution with four blocking items in b). Each arc is labeled with its cost.

After the access direction for each stack is determined, the configuration is represented as virtual lanes and fixed for the search procedure. Fig. 4.10 a) shows the grid representation with thick black lines to illustrate the virtual lanes. In Fig. 4.10 b) these lanes are transformed into a representation of the access direction for each lane. The corresponding access directions are specified underneath the stacks.

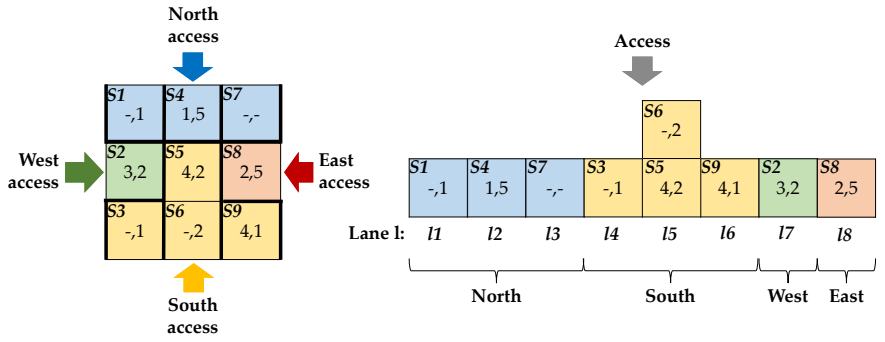


Fig. 4.10 Example from Fig. 4.8. Thick black lines illustrate the virtual lanes in a). In b) these lanes are transformed into a representation of one access direction.

4.4.2 Step 2 - Tree search in virtual lanes

In the second step of the algorithm with multiple access directions a tree search based on the former generated virtual lanes is performed which is similar to the tree search with a single access direction (see Section 4.3). For the lower-bound heuristics, we build on the adapted lower bound for one access direction:

We use the same procedure as for one direction to calculate the BX moves as well as to derive the demand, supply and the demand surplus $DS(g^*)$ and g^*. After the demand surplus $DS(g^*)$ has been calculated, the next step is to determine the necessary number of GX lanes n_{GX}^{nec}. This cannot be calculated in the UPMP via $n_{GX}^{nec} = max(0, [DS(g^*)/(J \cdot T)])$, since the length of each virtual lane can be different (e.g., l1 compared to l5 in Fig. 4.10). Also calculating the number of GX moves via $n_{GX} = \sum_{j=1}^{n_{GX}^{nec}} n^{g^*}(i)$ is problematic, since the potential GX lanes cannot be just sorted by the number of GX moves. It could make sense to select a lane that is more costly in terms of GX moves than another lane, but at the same time free up more space (e.g., moving two unit loads and getting six slots for the desired priority group compared to moving one unit load and getting only two slots). In fact, we need to select the lanes where a minimum number of GX moves is necessary to free up enough space for $DS(g^*)$. Therefore, we use a MIP formulation that minimizes the number of moves. The notation is shown in Table 4.4.

The MIP model to calculate the number of necessary GX moves is:

$$min \sum_{l \in L} k_l y_l \tag{4.15}$$

Table 4.4 Notation of the MIP model.

Set and Index	
L	Number of virtual lanes ($l \in \{1,...,L\}$) with well-placed items of a priority group $g < g^*$)
Parameters	
k_l	Cost of moving all well-placed items with a priority group $g < g^*$ of lane l
p_l	Potentially freed space of lane l calculated via the length of each lane l minus the well-placed items with $g \geq g^*$
Decision variable	
$y_l \in \{0,1\}$	Binary decision variable, if virtual lane l is selected to be emptied

$$\sum_{l \in L} y_l p_l \geq DS(g^*) \tag{4.16}$$

$$y_l \in \{0,1\} \quad \forall l \in L \tag{4.17}$$

The objective function in term (4.15) minimizes the sum of costs of the selected lanes. Constraint (4.16) ensures that the space requirement is fulfilled, which means that the sum of freed up storage slots must be equal or higher than the demand surplus $DS(g^*)$.

Applying the proposed lower bound on the example of Fig. 4.10, we get four BX moves. For the demand surplus, $DS(g^*) = 1$ and $g^* = 3$. Next, we calculate the costs k_l to empty each lane, as well as the potentially freed up space p_l of each lane l. Table 4.5 shows the results of k_l and p_l for each lane of our example. The lanes l with no well-placed items of a priority group $g < g^*$ are not considered because no additional space for g^* can be created. Therefore, lane $l3$ can be neglected.

Parameter type	$l1$	$l2$	$l4$	$l5$	$l6$	$l7$	$l8$
k_l	1	1	1	1	1	1	1
p_l	2	1	2	4	2	2	1

Table 4.5 k_l and p_l for each lane l of the example.

In this simple example, any lane besides $l3$ would free up the required one storage location for items with $g \geq g^*$. Since the costs for all lanes are the same, one of them can be chosen at random. We add one GX move to the lower bound and get an estimation of overall five moves to sort the storage.

4.4.3 Impact of access direction fixing

As indicated at the beginning of the section, in our problem definition we fix the access directions for each stack over the course of the search. Restricting the solution space might increase the required number of moves in certain situations. We show this effect based on two examples in Fig. 4.11 and Fig. 4.12.

The example in Fig. 4.11 a) shows the result of the network flow model for corner access from south and west. The unit load of priority group 2 at stack *S7* is blocked by two unit loads in the stacks *S8* and *S9*. However, this blockage could be resolved by a single move if the access directions are not fixed. Fig. 4.11 b) shows possible places to relocate the load in stack *S8* (light-gray crosshatch) when accessing it from the west. This allows access to the load in stack *S7* from the south without any blockage by passing over the empty stack *S8*. Thus, stack *S8* is accessed from two different access directions over the course of the pre-marshalling operation. Placing an item at *S2* or *S5* is, however, problematic, as this leads to a hole as the empty stack *S8* cannot be directly accessed anymore. Additional rules to prevent holes are necessary, but this limits the use of available storage locations.

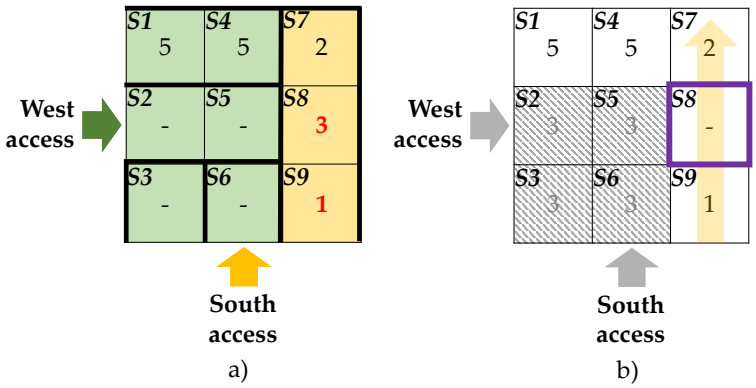


Fig. 4.11 The example in a) shows a configuration with access direction fixing illustrating two blocking unit loads in bold red. In b) all blockage is resolved by moving the unit load of stack *S8* (box with thick purple lines) to a possible location indicated with crosshatched tiles and a light gray number.

Fig. 4.12 a) is another example with three access directions, where the items at stack *S12* and at stack *S18* are blocked. The network flow model would propose to relocate two blocking items in front (e.g., *S7* and *S19*), but again, as shown in Fig. 4.12 b), a solution is possible via a single move in case there is no access direction fixing. The item from stack *S13* can be moved via west or south access to the light-gray crosshatched locations. Stacks *S12* and *S18* can be accessed via stack *S13* from the south and west. The difficulty is to maintain this access over time: When placing a unit load at stack *S13*, it

must be ensured that it does not block any other unit loads in both access directions.

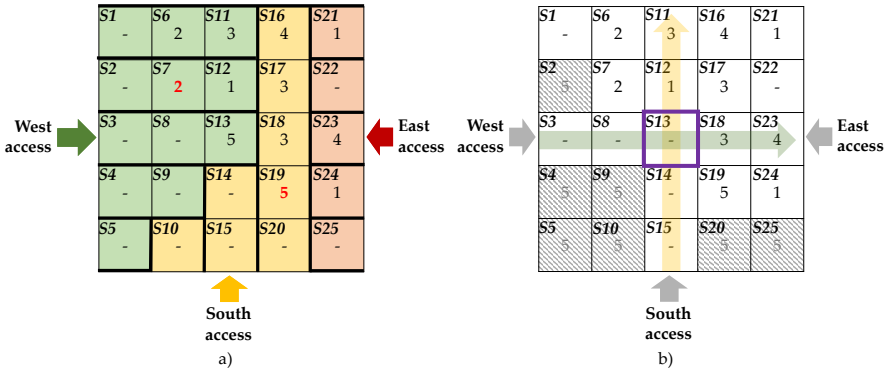


Fig. 4.12 Example in a) shows a configuration with access direction fixing illustrating two blocking unit loads in bold red. In b) all blockage is resolved by moving the unit load of stack *S*13 (box with thick purple lines) to a possible location indicated with a crosshatched tile and a light gray number.

These two examples show that fixing the access directions may require additional moves compared to solutions where a stack can be visited from several access directions over time. This means that the network flow model is not a valid lower bound without access direction fixing. However, real-life use-cases benefit from access direction fixing due to easier traffic management and a more stable and coherent configuration, especially for changes of priorities and incoming unit loads. To assess the share of instances, where additional moves are required compared to the network flow model further investigation is required.

4.5 Computational experiments

The goal of our analysis is to evaluate the effects of different access directions and assess the performance of the proposed solution approaches for the UPMP. Therefore, we introduce a publicly available extended dataset https://doi.org/10281/zenodo.7215494. and present the results in terms of the number of solved instances, runtime, and total moves. We further provide a root node analysis to investigate the gap of our lower bound as well as managerial insights.

We conduct our computational experiments on AMD EPYC 7401P Central Processing Unit (CPU)s at 2.45 GHz. Our algorithms are implemented in Python applying Gurobi 9.5 with the default parameters for the network flow model and the MIP. The execution time for each instance is limited to one hour of CPU time.

4.5.1 Instance generator and dataset

We develop an instance generator for the UPMP that places unit loads in a storage bay depending on the defined access directions. In addition to allowing up to four access directions (north, south, east and west), we vary the number of columns, rows and tiers of the storage bay as well as the fill percentage. We model five priority groups that can occur with uniform probability. We place unit loads in the storage bay successively at random and generate a new instance for each possible combination. During this procedure, it is necessary to consider the defined access directions to provide free storage locations that are directly accessible and avoid gaps. The generated instances may be infeasible, especially for high fill percentages. We generate an extended dataset with 3300 instances in which there are twenty instances (twice as many compared to our dataset in [PMT22]) for each possible combination of the parameters shown in Table 4.6.

The access variants (combinations of possible access directions) are:

- north (referred to as *single*)
- north and west (referred to as *corner*)
- north and south (referred to as *opposite*)
- north, south and west (referred to as *three*)
- all directions (referred to as *four*)

Parameter	Value
Access directions	{single, corner, opposite, three, four}
Size (I x J x T)	{3x3x1, 4x4x1, 5x5x1, 6x6x1, 7x7x1, 8x8x1, 9x9x1, 10x10x1, 3x3x2, 4x4x2 5x5x2}
Fill percentage	{40%, 60%, 80%}

Table 4.6 Varied parameters of the generated benchmark dataset.

Besides the sizes with a quadratic footprint shown in Table 4.6 other rectangular footprints are possible but omitted from this study due to limited space. The dimensions start at a 3x3x1 footprint (I x J x T) up to a size where we reach the limit of solvability due to timeouts for all access variants. A maximum 10x10x1 footprint for a single tier (overall 2400 instances: 20 instances for 5 access variants, 8 sizes and 3 fill levels) and a maximum 5x5x2 footprint for two tiers (overall 900 instances: 20 instances for 5 access variants, 3 sizes and 3 fill levels) results in up to 100 or respectively 50 storage locations for a single storage bay. This covers a large amount of real scenarios of small storage systems with a single bay.

4.5.2 Computational results

We start our analysis by presenting an overview of the solved instances and runtimes. This is followed by an analysis of the lower-bound heuristics' root node gap and a detailed comparison of the number of moves for all possible access variants for one and two tiers.

Runtime and number of instances solved

Table 4.7 provides an aggregated summary of the number of instances solved over all sizes and fill levels for a single tier. The detailed results are provided in Table A.1. The access variants are ranked in ascending order according to the number of solved instances, which represent all feasible solutions. Unsolved instances occur for the two reasons infeasibility or timeouts. In terms of infeasible instances, we are only able to verify infeasibility when the search runs completely through for instances of a single access direction and very small sizes at a fill level of 80%. We anticipate that there are some infeasible instances for other access variants as well, but we are unable to prove this within the timeout. Timeouts occur when we reach the limits of solvability for each access direction. With the access variants ordered according to the number of solved instances also the mean moves, mean visited nodes, and mean runtime improve. Exceptions only occur if not the same number of instances could be solved.

Access variant	No. solved	No. infeasible	No. timeout
single	326	20	134
corner	403	0	77
opposite	434	0	46
three	463	0	17
four	476	0	4

Table 4.7 Results summary for a single tier showing the number of instances solved, proved infeasible and unsolved (in total 480 instances each: 20 instances for 8 sizes and 3 fill levels).

Fig. 4.13 shows a barplot with the amount of up to twenty solved instances and provides the results for all access variants (legend) for each bay size of the dataset (y-axis). The amount of solved instances goes down with increasing size and fill levels. Whereas single access could already not solve all instances at a fill level of 40% (infeasible and timeout), four access directions solve all instances up to a fill level of 80% and size 10x10x1.

A detailed overview for 80% fill level is shown in Fig. 4.14. The fill levels of 40% and 60% are provided in Fig. A.1 and Fig. A.2 respectively.

Fig. 4.15 presents box plots (without outliers) illustrating the runtime in seconds. The peaks in runtime are especially observed in situations, where the

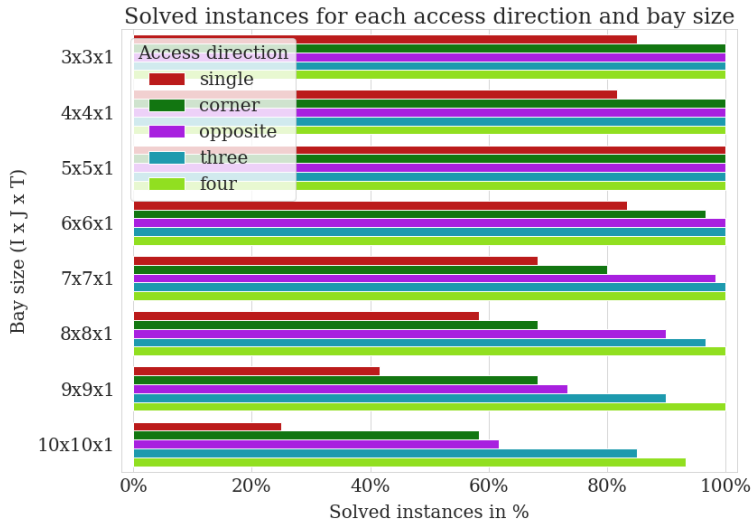


Fig. 4.13 Overview of solved instances for a single tier over all fill levels.

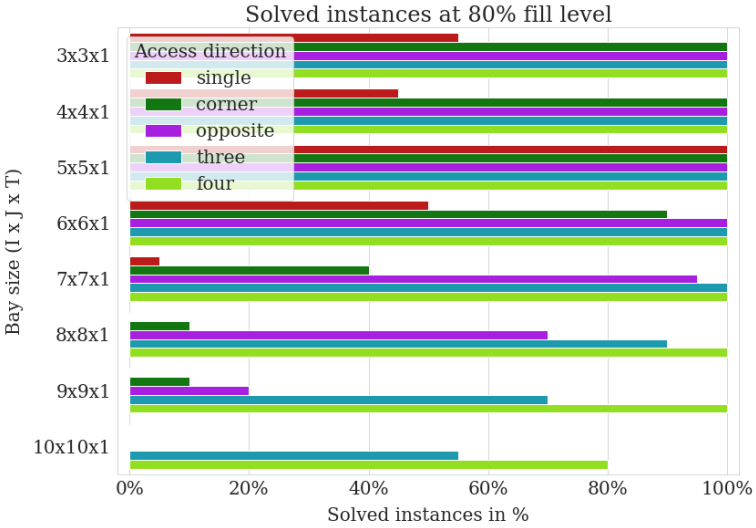


Fig. 4.14 Overview of solved instances for a single tier at a fill level of 80%.

limits of solvability are reached (e.g., single access at 8x8x1). At a certain point only a few instances are solved relatively fast, but the number of timeouts increases. Compared to single access the runtimes for the access variants with multiple access directions stay at a very low level. The overall mean runtime decreases from 120.17 seconds for single access, to 49.09 seconds for corner, to 19.89 seconds for three, to 16.58 seconds for opposite and 8.39 seconds for four access directions (Note that the number of solved instances including larger sizes increases for multiple access directions). The maximum runtime

for four access directions up to a size of 9x9x1 is 30.43 seconds. Even for 10x10x1 where not all instances could be solved anymore, we do not exceed 306.45 seconds.

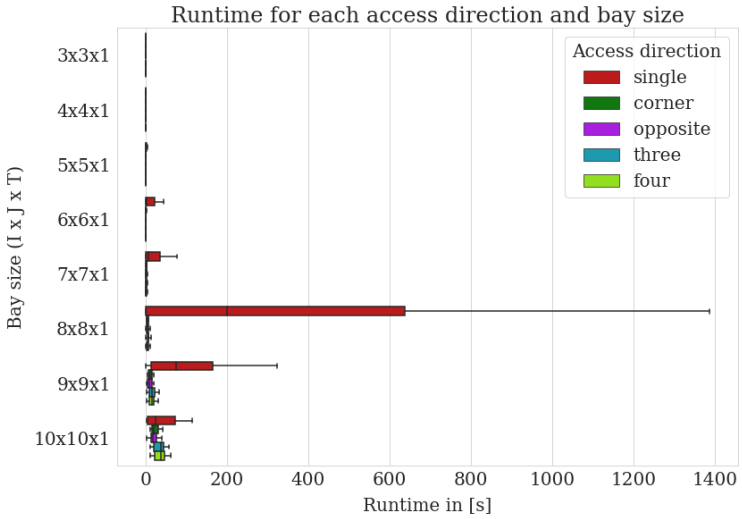


Fig. 4.15 Box plots (without outliers) of the runtimes for a single tier over all fill levels.

Only for a size and fill level, where single access could not solve any instances an significant increase in runtime (for instance corner access at size 7x7x1) is noticed. In general, we see an increase in runtime mainly right before or when we encounter the first unsolved instances. If not all instances have been solved, a longer runtime or infeasible instances (especially at 80% fill level) can be expected. We also come across a few unsolved instances for access variants, where the mean runtime is generally at a very low level (for example corner access at size 9x9x1 and 10x10x1). Reason can be an outlier that is exceptionally difficult or an infeasible instance.

As shown in Fig. 4.16, at 80% fill level finding a solution gets more difficult (A detailed analysis for 40% and 60% fill level is provided in Fig. A.3 and Fig. A.4): More infeasible configurations appear due to a limited amount of empty storage locations and on average more required moves are necessary to sort the block storage. For a single access direction we encounter four infeasible instances at size 3x3x1 and 4x4x1 (see also Table A.1). The size 5x5x1 is the only size, where every possible access variant is able to solve all instances. The reason for this is that 80% of 25 storage locations are exactly 20 unit loads (no rounding of unit loads necessary) leaving five empty storage locations for pre-marshalling. Five empty storage locations allow to clear a full lane for size 5x5x1, enabling to find feasible solutions.

At size 10x10x1 three access directions are still able to solve six instances and four access directions do not solve all instances for the first time. This

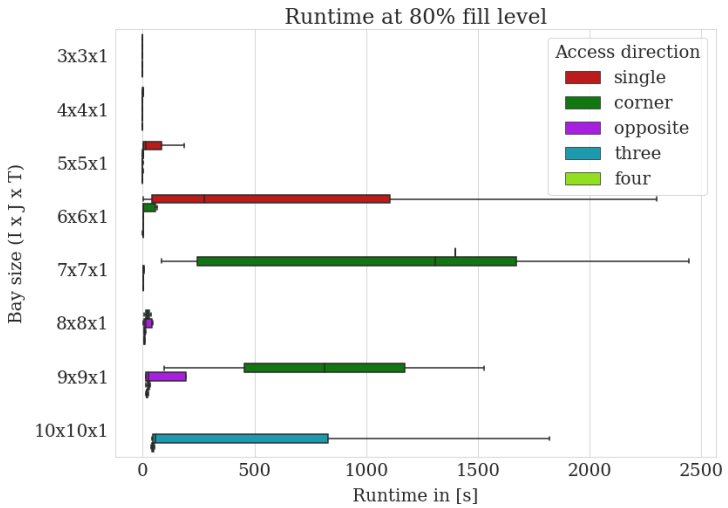


Fig. 4.16 Box plots (without outliers) of the runtimes for a single tier at a fill level of 80%.

overview shows particularly that multiple access directions solve much more instances than single access and at the same time achieve comparatively very low runtimes. Obvious is also the advantage of being able to solve instances with a high fill level of 80%. Interesting would be to increase the fill level even further.

Access variant	No. solved	No. infeasible	No. timeout
single	109	20	51
corner	129	0	51
opposite	168	0	12
three	170	0	10
four	175	0	5

Table 4.8 Results summary for two tiers showing the number of solved, infeasible and unsolved instances (in total 180 instances each: 20 instances for 3 sizes and 3 fill levels).

Table 4.8 presents a short summary of the results for two tiers. Detailed results are again appended at the end of the paper in Table A.2. Our experiments show that adding additional tiers in a scenario with a fully shared storage strategy is very demanding. Already at a size limit of 5x5x2, we are not able to solve all instances with any access variant. Nevertheless, multiple access directions perform much better. The number of solved instances increases from 109 for a single access direction to 175 out of 180 for four access directions.

Fig. 4.17 shows the barplots for solved instances.

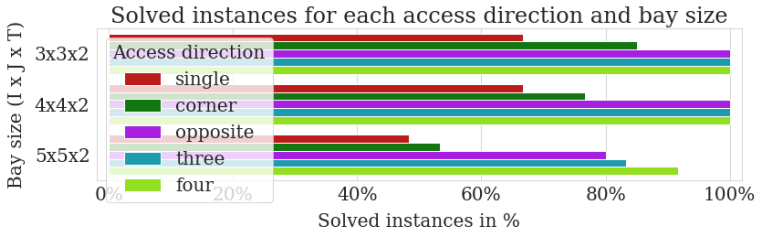


Fig. 4.17 Overview of solved instances with two tiers over all fill levels.

For two tiers, we are able to solve all instances up to size 4x4x2 and 80% fill level with two opposite, three and four access directions. Corner access is already struggling for very small instances, since an unsorted stack in the back corner may lead to an infeasible instance that cannot be solved if not enough empty storage slots are available. Even though we are only able to solve rather small instances the improvement of multiple to a single access direction is evident as a single direction fails to solve any instance with 80% fill level (see Fig. 4.18).

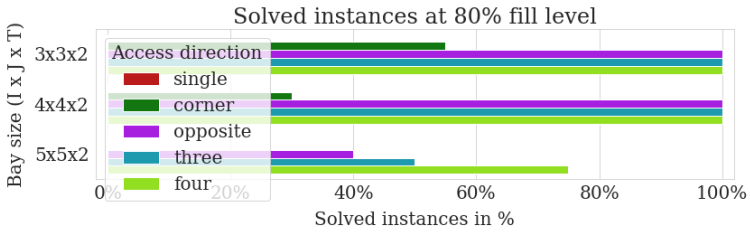


Fig. 4.18 Overview of solved instances with two tiers at a fill level of 80%

Details for 40% and 60% fill level are provided in Fig. A.5 and Fig. A.6.

Fig. 4.19 shows the plots for solved instances and their runtime for two tiers. The overall mean runtimes improve from 106.12 seconds for opposite access, to 64.53 seconds for single access, to 62.39 seconds for corner, to 57.64 seconds for three to 12.23 seconds for four access directions. The high mean runtime for opposite directions compared to single and corner access can be explained by much more solved instances at higher fill levels. For a fill level of 40% all instances are solved with overall very low runtimes. The runtimes are even slightly increasing with the amount of access directions due to an increased branching factor (more virtual lanes).

Fig. 4.20 shows that for higher fill levels the runtime is decreasing with an increasing amount of access directions due to fewer moves that are required. Again a detailed analysis for 40% and 60% fill level are provided in Fig. A.7 and Fig. A.8.

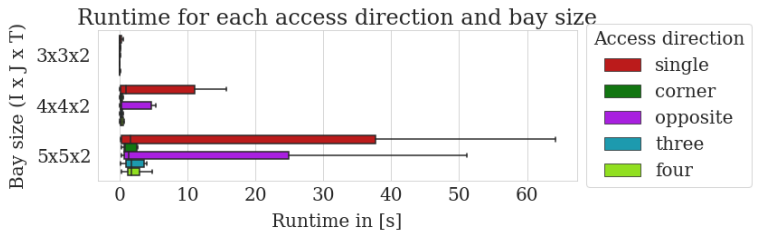


Fig. 4.19 Box plots (without outliers) of the runtimes with two tiers over all fill levels.

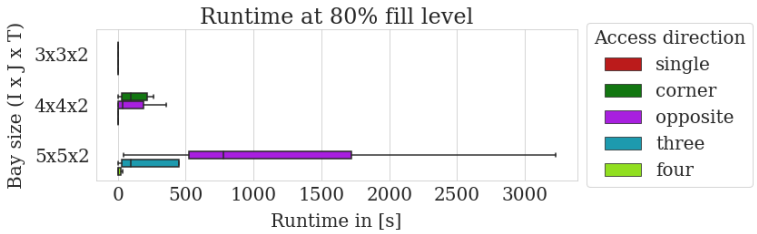


Fig. 4.20 Box plots (without outliers) of the runtimes with two tiers at a fill level of 80%.

Root node analysis

In the root node analysis shown in Table 4.9, we compare the lower-bound heuristics for the root node with the exact result of required moves for all solved instances. Interestingly, the results differ when considering additional access directions. For a single tier the root node gap decreases from 7.89% for a single access direction to 0.00% for three and four access directions. In case of two tiers, the root node gap decreases from 20.95% down to 0.07% for four access directions. The reduction of the root node gap from a single to multiple access directions is remarkable. We believe that generally fewer required moves, a larger action and solution space as well as shallow virtual lanes lead to a huge improvement of the lower-bound root node gap. In case of two access directions, the lower bound for opposite directions is slightly more accurate than for the corner case. Reason are generally less moves and also more shallow virtual lanes for opposite directions.

Access Directions	Mean Root Node Gap 1-tier in %	Mean Root Node Gap 2-tier in %
single	8.31	20.20
corner	0.79	3.38
opposite	0.19	3.32
three	0.00	1.20
four	0.00	0.15

Table 4.9 Root node analysis for all access directions.

Required number of moves

Fig. 4.21 shows box plots for the number of required moves for each combination of access directions and all considered sizes over all fill levels with a single tier. The box plots clearly illustrate the big advantage of multiple access directions compared to a single access direction. Logically, the number of required moves goes down for each additional access direction that is available. As an example the mean moves for size 5x5x1 are 9.47 for single, 4.03 for corner, 2.20 for opposite, 1.43 for three and 0.68 for four access directions. Corner access is a big improvement, but not as much as opposite, three and four access directions. Reason is that reaching the back corner always requires a deep lane.

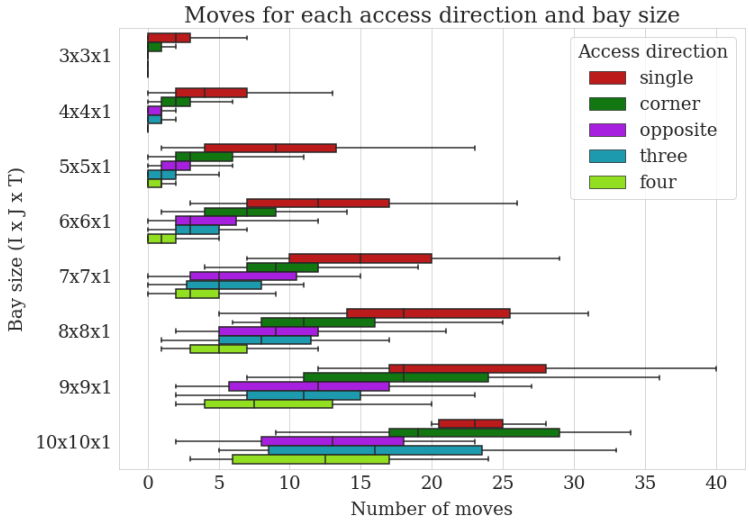


Fig. 4.21 Box plots (without outliers) for a single tier showing the number of moves for each access variant and each bay size. This is based on the solved instances shown in Fig. 4.15.

High fill levels are especially challenging as they require more moves while fewer free storage locations are available to sort the storage bay. Primarily at 80% fill level shown in Fig. 4.22, some access variants do not solve all instances as presented in the previous paragraph. Thus, even though opposite directions seems to perform better than three access directions for bay size 10x10x1, it did just not solve as many instances. The detailed graphs for 40% and 60% fill levels are shown in Fig. A.9 and Fig. A.10.

The box plots for two tiers in Fig. 4.23, and for each fill level respectively in Fig. A.11, Fig. A.12, and Fig. A.13, show in general a similar picture as a single tier. However, for all sizes a much higher number of moves is necessary compared to a single tier (at least twice as high mean for all sizes). The main problem for several tiers is that the probability of an unsorted stack increases

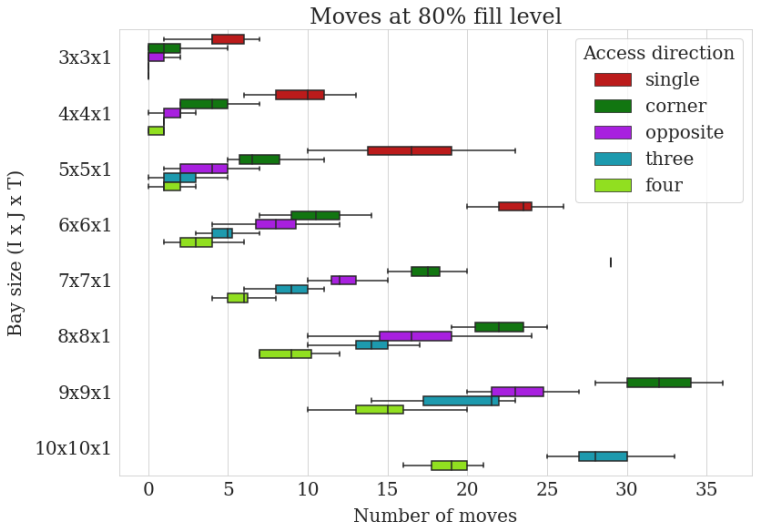


Fig. 4.22 Box plots (without outliers) for a single tier showing the number of moves for each access variant and each bay size at the fill level of 80%.

with the number of tiers and requires to relocate all items that are placed on top and in front. Therefore, generally a high number of items is blocking and must be relocated.

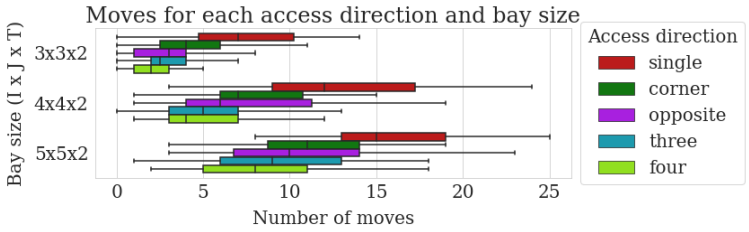


Fig. 4.23 Box plots (without outliers) for two tiers showing the number of moves for each access variant and each bay size. This is based on the solved instances shown in Fig. 4.19.

4.6 Insights for operations management

The main advantage of block stacking storage systems under a fully shared storage policy is to achieve high storage densities. Howhttps://keep.google.com/ever, high storage density comes at the cost of decreased accessibility due to blockage. Our results clearly show that this disadvantage can be significantly mitigated if up to four access directions are supported both technically by the load carriers and algorithmically in the control system. As discussed in the following, the number of access directions immensely impacts the system in three dimensions of (1) efficiency and feasibility, (2) layout decisions, and (3) traffic management.

Efficiency and feasibility

Our results show that the required number of moves to sort a storage bay drop considerably with an increasing number of access directions. This becomes especially visible at high fill levels. The main reasons are (1) an increased number of directly accessible unit loads, (2) a decrease in average lane depth (less blocking unit loads and better free slots), and (3) variable determination of lane depth and configuration (increased flexibility to access unit loads).

Our experiments show clearly that only using a single access leads to a large number of infeasible instances at high fill rates. Thus, single access is more suitable for scenarios where no sorting is necessary. Increasing the number of tiers also leads to a much higher number of moves, as well as many infeasible instances. Hence, we recommend to allow mixing of SKUs in lanes but not within a stack (pure stacks).

Layout decisions

Access directions have a direct impact on layout decisions. Indeed, our study results clearly suggest that when planning an aisle configuration, planners should consider the use of multiple access directions. Of course, this results in a trade-off between aisle and storage space (fast access and travel times versus a high space utilization). Further access directions might require additional aisle space but can enable deeper bays without suffering bad access times. Another aspect to be considered during layout decisions are the different results for two sided access: If possible, an access from two opposite directions is preferable to corner access as the pre-marshalling results in less moves. Reason is that in the opposite setup lanes are always divided in two parts and, hence, are shorter on average. Furthermore, in an opposite access direction setup for vehicle operations no access from crossing directions needs to be considered and most standard load carriers support opposite access (e.g., pallets).

Operational control

Arrival and retrieval times of unit loads can be stochastic. For scenarios with high uncertainty for the in- and outbound flow sequence, the higher number of directly accessible unit loads and the higher number of access direction options in a multiple access scenario is advantageous as less sorting effort results.

Furthermore, the concept of virtual lanes is beneficial for the vehicle management. As we build only contiguous virtual lanes, no crossing paths of vehicles as illustrated in the examples of Section 4.4.3 can occur. Hence, vehicle management does not need to consider collision avoidance within the bay. Also, additional access directions lead to a potentially better spatial distribution of transport tasks/AMRs. Many existing setups could benefit by not simply splitting a storage bay in the middle into two halves and utilizing the already available additional access directions.

An overview of the key aspects is provided in Table 4.10. Single access performs worst in all categories and requires a higher sorting effort expressed by far more moves than the other access variants. Also the share of infeasible and unsolved instances is problematic. No high fill levels are possible without additional space. Therefore, we do not recommend a shared storage policy for single access. It is more suitable for scenarios where no sorting is necessary like dedicated lanes for each SKU and a LIFO retrieval strategy.

Corner access is already a big improvement compared to single access. However, the other variants with multiple access directions perform better. Compared to two opposite access directions we observed a higher number of required moves and more unsolved instances. Also, we observed that corner access is prone to produce holes of empty storage locations that cannot be accessed anymore during operations. When placing an item via one access direction, it must be checked that no empty location or item is blocked via the other direction. This requires additional rules. Another downside is that load carriers are required that can be accessed from two orthogonal directions (assuming virtual lanes are not fixed forever). We recommend to use corner access only for specific situations (e.g., due to walls) where no other variant with multiple access directions is possible.

Opposite access brings an additional significant reduction of required moves and solved instances. For a single tier all instances up to 7x7x1 can be solved instead of 5x5x1 for corner access. Compared to corner access the lanes are more shallow as they are always divided in two parts. Opposite access is particularly relevant for practitioners and is the most simple to implement as many standardized load carriers like pallets allow already access from two opposite directions. Also traffic management and deadlock prevention are easier without any crossing directions. With an increasing size and fill level further access directions become more and more helpful.

Three access directions are a further slight improvement in terms of required moves as well as solved instances. However, the sizes that could be fully solved are the same as for opposite access. The required load carriers with three access directions are most likely also accessible from four directions. Therefore, it is rather something for situations where only three access directions are possible, because otherwise all four access directions could be used right away.

Four access directions perform best with the smallest number of mean moves for all sizes of instances and the most instance that could be solved. Even though load carriers need to be accessible from four directions, this can already be realized in many storage systems (e.g., pods in RMFS or material transport trolleys). Four access directions are also most flexible to adapt to any changes due to uncertainty (changing retrieval groups) or the storage of incoming items.

In general, our results indicate that multiple access directions can greatly increase sorting efficiency for high-density systems, and we suspect that this will hold true also when for storage and retrieval operations. However, further investigations are needed to confirm the value of multiple access directions in a more general intralogistics setting.

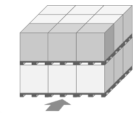
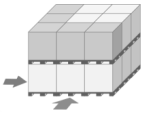
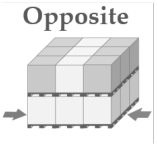
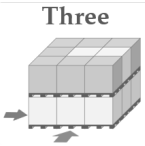
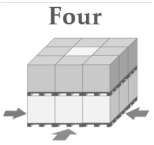
Access	Observation	Recommendation
Single 	• Largest number of required moves. • Highest share of unsolved and infeasible instances.	• We recommend it for scenarios where sorting is not necessary: Lanes dedicated to SKUs and a LIFO retrieval strategy.
Corner 	• Not as good as other variants with multiple access directions. • Requires rules to prevent holes due to crossing directions.	• Should only be used, if no other option with multiple access directions is possible. • Loads carriers need to be accessible from two orthogonal directions.
Opposite 	• Big improvement of required moves compared to single and corner access. • Also less unsolved instances.	• Easy to operate due to no crossing access directions. • Instantly realizable for many types of storage systems and load carriers (e.g., pallets).
Three 	• Slight improvement compared to opposite access directions. • However, same size limits for fully solved instances.	• Loads carriers need to be accessible from minimum three directions. • We recommend to use it only, if all four directions are not possible.
Four 	• Smallest number of mean moves for all sizes of instances and most instances solved.	• Best accessibility of items and most adaptable to any upcoming changes. • Recommended if access from four directions is feasible.

Table 4.10 Overview of the main observations for the different access variants.

4.7 Conclusion

We introduce the UPMP for sorting block stacking storage systems in idle time. Solving the UPMP allows organizations to increase the storage density of their block stacking storage, while still ensuring fast access times. This enables new kinds of compact layouts for many types of storage systems that consist of movable floor units, like pallet-based block storage, RMFS, container yards, material transport trolleys and many more.

Our solution approach sorts storage bays with any combination of up to four access directions to minimize the number of moves. In the case of the UPMP with a single access direction, we show that the solution approach from the CPMP can be directly applied. For any combination of multiple access directions, we present a novel two-step approach that fixes the access directions for each stack throughout the pre-marshalling process and allows for solving real-world sized instances with up to 81 storage locations without any timeouts within a limit of one hour.

The experiments show that multiple access directions significantly reduce the required number of moves and allow solving much more instances with faster runtimes, especially for high fill levels. Our detailed comparison of access directions and tiers provides valuable insights for the development and layout design of highly dense block storage systems. In a next step, several future research directions are possible. Our search procedure and adapted lower-bound heuristic proves to be strong with a very low root node gap for multiple access directions. However, further improvements to accelerate the search are possible (see e.g., [TPV17]; [TT18]; [Tan+19]).

Furthermore, future work is required to increase the applicability of our approach to a variety of use cases. Extending the scope to large, multi-bay warehouses would require adjusting the objective of minimizing the number of moves to also consider travel time or distance (e.g., [Par+20]). Finally, heuristic approaches have had significant success for the CPMP, and could be also useful for solving large instances of the UPMP.

Chapter 5
Sorting multibay block stacking storage systems

In Chapter 4 we considered the UPMP for a single bay. Besides the small storage capacity, one of the main limitations was that the fill level could be raised only as much as empty storage locations are needed to be able to sort the bay. Increasing the number of bays resolves both. The storage capacity can be increased by setting up a warehouse layout with multiple bays. Here the capacity can be scaled independently of the size of the bays. Also, the necessary number of empty storage locations to find a feasible solution can be distributed cross all bays, which allows much higher fill levels.

In this chapter, we introduce a solution approach for the MUPMP that extends pre-marshalling from a single bay to larger setups with multiple bays, thereby minimizing the sorting time. We use our introduced two-step solution approach based on a network flow model and an A* algorithm from Chapter 4, extending it with an upper bound and additional tie-breaking rules. Unlike the small single-bay storage systems in the UPMP, larger layouts in the MUPMP require considering not only the number of moves but also the move time/distance of the AMRs. At this point, we consider only the move distances/time (loaded travel) and not the whole travel distance/time (incorporating unloaded travel). Including the travel distance from a delivery point to the next pick location (empty travel) would require knowledge of the exact position of the AMRs as well as dispatching decisions (e.g., solving a VRP or traveling salesperson problem (TSP)). Even though we expect the total move distance/time to be a good approximation of the total travel distance/time, there is certainly room for improvement.

Extending the scope to multiple bays also requires a highly efficient solution approach. We further improve our implementation with an incremental lower-bound calculation and present an algorithm that splits the resulting move sequence into independent sequences for multi-agent scenarios. The results show that multiple access directions are able to significantly improve

the access times of unit loads as well as the sorting effort for block stacking warehouses with multiple bays.

Fig. 5.1 shows an example of a multibay block stacking warehouse with four bays separated by aisles for traveling. Similar to the example of a single bay in Chapter 2 (see Fig. 2.1), each bay contains unit loads that are placed in lanes. Dependent on the available access directions and the properties of the unit load, the outermost stacks are accessible from up to four cardinal directions. The use of AMRs for material handling allows increased storage density by utilizing a shared storage policy and rearrangement operations that are difficult for human operators to implement due to the high availability of AMRs, also during off-peak hours. AMRs can use the idle time to rearrange the storage and prepare it for a future demand sequence that may be derived from a production schedule or truck-loading time slots.

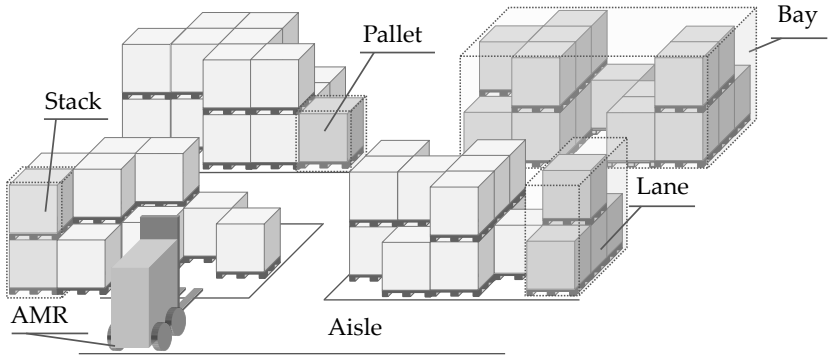


Fig. 5.1 Example of a multibay block stacking warehouse with a robotic forklift.

We commence this chapter with an introduction of the MUPMP followed by the description of the multibay unit-load pre-marshalling problem Section 5.1 and a brief literature review in Section 5.2. Subsequently, we introduce the extended solution approach in Section 5.3 and conduct a series of experiments presented in Section 5.4. This is complemented with a summary and outlook in Section 5.6.

5.1 The multibay unit-load pre-marshalling problem

The MUPMP is about sorting a block stacking warehouse that may contain multiple bays until the resolution of all blockage based on the future retrieval time of each unit load. Similar to the UPMP for a single bay, we assume that the AMRs are idle and that no unit loads are going in or out of the storage system. The warehouse consists of N given bays that are interconnected via aisle space for traveling. Each bay $n \in \{1,...,N\}$ can be accessed from one or up to four access directions and is filled with a set of unit loads with a known position and retrieval priority group. Because a fully shared storage policy is applied, SKUs can be placed anywhere and mixed in lanes as well as stacks. The retrieval priority group is derived based on a known (e.g., planned truck arrivals or production schedule) or predicted (e.g., based on material supply precedence graph or demand forecast) future retrieval time. We define unit loads as *blocking* when they have a higher retrieval group compared to the unit loads that are placed underneath or behind. These unit loads with a lower retrieval group will be retrieved before the unit load with a higher retrieval group. Whereas in case of the UPMP, the goal was solely to find a minimum number of unit-load moves to sort a bay such that there is no blockage, in case of the MUPMP, the move time or distance must also be considered due to the much greater time/distance differences between the moves. Minimizing the total move time allows pre-marshalling to be conducted in short time intervals, thereby ensuring long operational availability. Hence, our goal for solving the MUPMP is to minimize the move time for the pre-marshalling process that consists of the moves as well as the handling times (e.g., pick-up and fine-positioning), but not the unloaded travel time. Because multibay block stacking warehouses are usually operated by multiple vehicles, it is necessary to assign and sequence the resulting moves to the AMRs. In terms of AMR specifications, we assume that they cannot reach over a stack in front and can carry only a single unit load.

Fig. 5.2 shows the state representation of a block stacking warehouse with four bays arranged as a square. Within a bay, the storage locations are represented as a three-dimensional grid with columns $i \in \{1,...,I\}$, rows $j \in \{1,...,J\}$, and tiers $t \in \{1,...,T\}$ that is filled with unit loads of priority groups $g_{ijt} \in \{1,...,G\}$. Assuming rectangular warehouse footprints with length L and width W, we also use a two-dimensional global x-y coordination system to determine the position of bays, storage locations, and access points in the warehouse. Access points $p \in \{1,...,P\}$ are located in the aisle space in front of the bays and specify the stacks that can be directly accessed. The multibay UPMP is fully defined by a list of bays, where each bay contains $(I, J, T, G, g_{ijt} \in G^{IxJxT}, P)$.

The example in Fig. 5.3 (a) shows a warehouse configuration containing two square bays of the same size with $I = 3$, $J = 3$ and $T = 1$ that can both

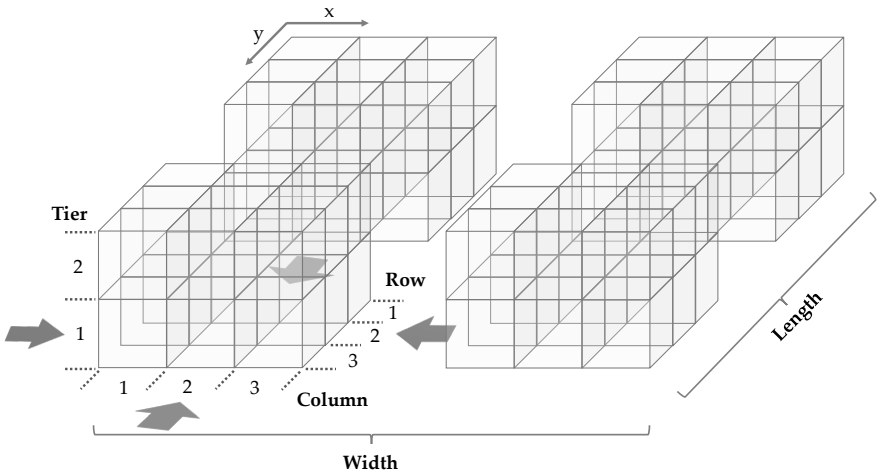


Fig. 5.2 State representation with four bays of the multibay block stacking warehouse.

be accessed from all four directions. It is filled with unit loads of retrieval groups g_{ijt} from one to five that are as depicted at each unit load. Fig. 5.3 (b) switches the perspective to top view with white tiles for empty/aisle space and dark gray tiles for unit loads. The bold red digits are the unit loads that are blocking the center unit load of the bottom bay. A red-dotted arrow indicates a possible move to resolve all blockage. Relocating one of the other blocking unit loads would also be possible, but requires a longer transport distance. Fig. 5.3 (c) shows the resulting sorted configuration of the MUPMP.

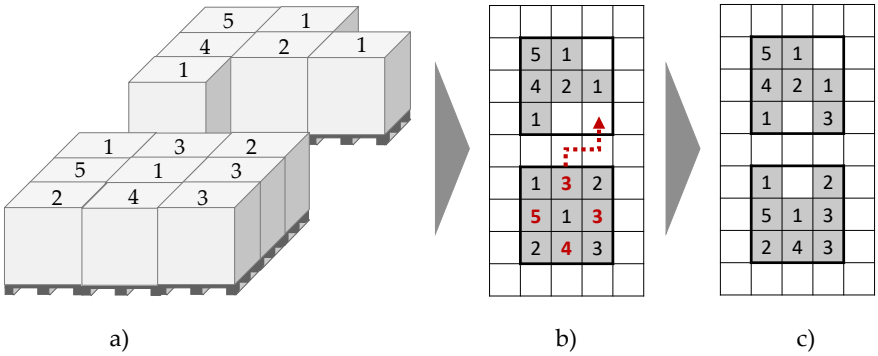


Fig. 5.3 Example configuration in (a), top view of the grid-based layout and red arrow for a possible move in (b) and solution of the MUPMP in (c).

5.2 Related work

Our literature review presents related publications focusing on rearrangement activities for multibay setups as well as the minimization of move time. For an overview of single-bay pre-marshalling literature, we refer to Section 4.2. A literature review of more general rearrangement problems is provided in Section 3.4.

5.2.1 Multiple bays

In maritime transportation, the rearrangement of containers within multiple bays is known as the remarshalling problem (RMP). Similar to pre-marshalling, the goal is to sort the scattered containers based on their retrieval priority with a minimum number of moves until all blockings are resolved [CSV11]. Different problem definitions exist for RMP, which the following literature review shows.

An early publication by Kim and Bae [KB98] deals with the RMP and converting the configuration of a current bay into a target configuration. Instead of the exact position of each container, the bay configuration is defined simply as the number of containers per group that are located within each bay. The group is a collection of containers going to the same destination on the same vessel. The problem is divided into two sub-problems: (1) bay matching and move planning and (2) sequencing the moving tasks. In the first step, dynamic programming and an approach referred to as "transportation algorithm" are used to determine the flow and movement between bays to reach the target configuration. Second, the task sequencing problem is modeled as a TSP with precedence constraints to consider space availability at the destination bay, with the goal of minimizing travel time. Because the exact positions of the containers are neglected in the bay configuration, the rehandling of containers within a bay is not addressed in this work.

Later publications take exact container positions into account and investigate the intra-block RMP where containers are sorted based on categories and retrieval priorities. In successive publications, Kang et al. [Kan+06] and Choe et al. [Cho+11] investigate the RMP where, starting from a source, bay containers are relocated to an empty target bay so that no rehandling is necessary either during the rearrangement process or later in the ship loading process. Interference between multiple cranes is taken into account. Both publications present a similar two-phase solution approach that couples a simulated annealing algorithm for finding a target configuration and partial order graph with crane scheduling to determine a minimum-time re-marshalling plan. Park, Park, and Ryu [PPR09] study the RMP in a container terminal with two cranes. Whereas one crane is used for loading containers from the storage yard on the vessel (seaside), the second crane is used to carry in containers from external trucks (landside). The goal of their solution is to bring con-

tainers from the landside of the storage yard to the seaside where the ship loading happens and, at the same time, sort the containers based on their retrieval priority. Therefore, the authors propose a two-stage algorithm: First, potential stacks where the containers will be moved are selected heuristically. Second, an algorithm is used to conduct an alternating search to solve the two sub-problems of determining target slots that are assigned to each container as well as the order of container movements.

Further approaches deal with rearrangement activities to improve the location of containers whenever one of the cranes is temporarily idle (called "housekeeping" [Kem12, p 148] or "iterative rescheduling"), where the re-marshalling jobs that most reduce future loading times are selected with the objective of minimizing current delay times as well as the overall makespan [Cho+15]. Besides re-marshalling before a ship is loaded (export terminal), re-marshalling activities may also be performed after unloading a ship (import terminal) before containers are retrieved (e.g., to external trucks). Asperen, Borgman, and Dekker [ABD13] and Covic [Cov17] investigate this type of re-marshalling problem where container departure times are reported by a truck announcement system.

The presented research for multiple bays deals with a variety of use-case-specific problems (e.g., moving containers to an empty target bay) using relaxing assumptions to simplify the problems. We could not find an extension of the CPMP to the RMP, possibly because cranes are typically installed permanently for a single block (here: bay) and cannot move between them.

5.2.2 Minimization of crane times

Besides publications that specifically address rearrangement operations between multiple bays, there are also solution approaches of the CPMP considering crane times. Instead of minimizing moves, Parreño-Torres et al. [Par+20] address the minimization of crane times for the pre-marshalling of containers. The authors propose an integer linear model that builds upon [PAR19] as well as a branch and bound algorithm, which is set up as an iterative deepening search. Besides the lower bound from [Tan+19], a novel lower bound incorporating the travel time, as well as a heuristic algorithm for an upper bound for the number of moves, are also introduced. The heuristic algorithm is executed at each iteration of the search, and the algorithm stops when the depth level reaches the upper bound. The novel time-based lower bound extends the lower bound for the number of moves. Each move is multiplied by a minimum time that consists of the travel time unloaded and loaded, crane lowering and lifting times, and twistlock time. Further variants of the time-based lower bound also consider the position of the blocking container and the highest positions they can be moved to. The time-based lower bound is used as a secondary tie-breaking criterion. Overall, the branch

and bound algorithm outperforms the integer programming models in terms of solved instances and runtime. In a recent publication, Parreño-Torres, Alvarez-Valdes, and Parreño [PAP22] propose a beam search algorithm that is able to find all previously found optimal and more improved solutions (not necessarily optimal). For the tree search procedure, the authors use several tie-breaking criteria based on the move type (BB, BG, GB and GG; see Section 4.3.3), develop a global evaluation heuristic to obtain the most promising nodes, and present new dominance rules for eliminating nodes that cannot result in a better solution.

The research on the minimization of crane times by Parreño-Torres, Alvarez-Valdes, and Parreño [PAP22] uses a lower bound based on minimum travel times multiplied by the number of moves. In single-bay scenarios, this is a feasible approach. However, in larger multibay warehouses, using a similar lower bound based on the minimum move time would not be an effective approach.

5.3 Adapted solution approach for the MUPMP

We adapt the approach proposed in Chapter 4 for the multibay scenario. Besides a pre-processing step for a new warehouse representation with multiple bays and access points, the main extensions are several tie-breaking rules that include the total move distance, an optional additional upper bound, the improved incremental calculation of the lower bound, as well as a post-processing step to split the resulting move sequence into independent sequences to enable the execution of pre-marshalling in setups with more than one AMR.

We start this section by giving an overview and briefly describing the steps of our solution approach based on a network flow model and a tree search procedure that are directly transferred from Chapter 4. The process flow of our two-step approach is shown in Fig. 5.4. Beforehand, a pre-processing step converts the warehouse representation into a list of bays with access points that indicate from which aisle locations stacks are accessible. Based on a graph that is placed in the grid-based layout, we calculate a static distance matrix with the distances between all access points.

In the first step, we use the network flow model to determine the access directions for each stack. The objective is to minimize the blocking unit loads. Based on the access directions, we are able to transform the configuration into a state representation of virtual lanes. Each virtual lane consists of a sequence of 1 to $\max(I, J)$ stacks in a row/column that can be accessed one after the other. The access direction for each virtual lane is defined by an access point, and the length of the lane may be different.

In the second step, our tree search procedure uses the virtual lane representation to find a solution for sorting the warehouse with a minimum number

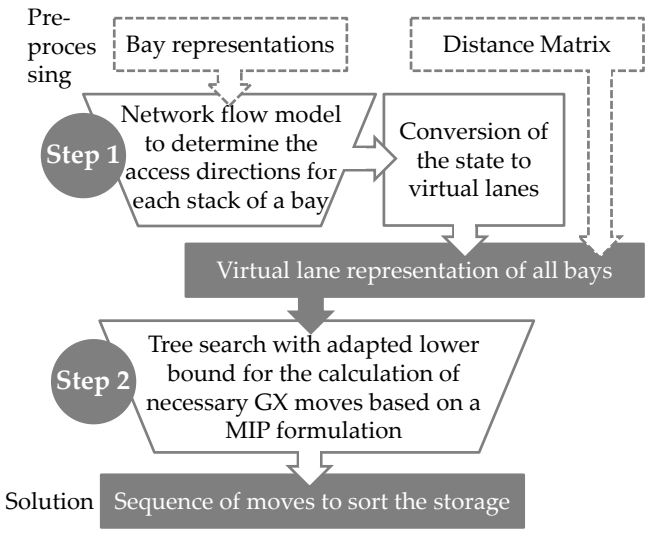


Fig. 5.4 Illustration of the main components of our extended solution approach based on Chapter 4.

of moves. The centerpiece is our adapted lower-bound heuristics with a MIP formulation (see Section 4.4). The resulting sequence of moves is then split into as many independent moves sequences as possible. Independent move sequences allow us to easily assign relocation tasks to multiple AMRs.

In the following subsections, we explain our solution approach in detail. We focus on the newly added adaptions as well as the additional pre- and post-processing steps.

5.3.1 Input

In the beginning, we add the required inputs. A grid-based warehouse is represented as a list of bays including the access points and directions to the respective bay, as shown in Fig. 5.5. The purple dots indicate possible access points from aisle space to the bay. This example allows access from all four cardinal directions. Next, we lay a graph onto the aisle space tiles and calculate all distances between the access points of all bays via breadth-first search. A downside of using such a static distance matrix between all access points is that possible shortcuts through storage space in case of low fill levels are not considered.

5.3.2 Step 1: Access direction fixing

In this step, we want to define the access direction for each stack. This is necessary to make the application of our lower bound valid and also because we want to avoid crossing AMRs when accessing a bay. Fixing the access

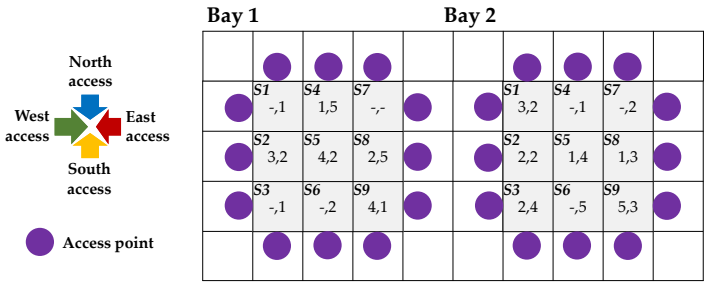


Fig. 5.5 Example of a warehouse configuration with two bays and possible access points.

directions at the beginning of our approach may lead to additional moves, as discussed in Section 4.4.3. To calculate the access direction for each stack that minimizes the number of misplaced unit loads, we use the network flow model introduced in Section 4.4.1 (Equations (4.9) to (4.14)). The result is used to determine a virtual lane representation that defines how each stack is accessed via a defined access point. In our previous version in Section 4.4, we obtained a single optimal solution in case many symmetrical solutions were available. Now, we want to utilize situations in which many symmetrical solutions are available by returning up to 10 optimal solutions for each bay via the solution pool feature of the Gurobi MIP solver. We then compare the solutions with our more advanced lower-bound heuristic that also considers the priority groups of blocking unit loads and open locations. The details of the lower-bound heuristics based on the demand/supply-concept to estimate the number of moves with the MIP extension are explained in Section 4.3.3 and Section 4.4.2 (Equations (4.15) to (4.17)).

Fig. 5.6 gives a short example with two bays that produce a list of virtual lanes. The two bays on top of the figure show an exemplary result of the network flow model for the example from Fig. 5.5. Thick black lines and the colors illustrate the virtual lanes with the respective access direction and access point. On the bottom of the figure, we show the final virtual lane representation that is used throughout the following procedure to determine the pre-marshalling moves.

5.3.3 Step 2: Tree search

The new implementation of the A* algorithm differs in some details from our previous work in Section 4.3.1. We introduce tie-breaking rules that consider the travel distance for popping nodes out of our priority queue, an optional upper bound to reduce memory consumption, and a faster incremental calculation of the lower-bound heuristics (which is not visible in the algorithm). Algorithm 2 shows the modified algorithm. We initialize the search procedure with a root node and optional upper bound. The main search loop runs until a solution with no blocking unit loads is found, until the open

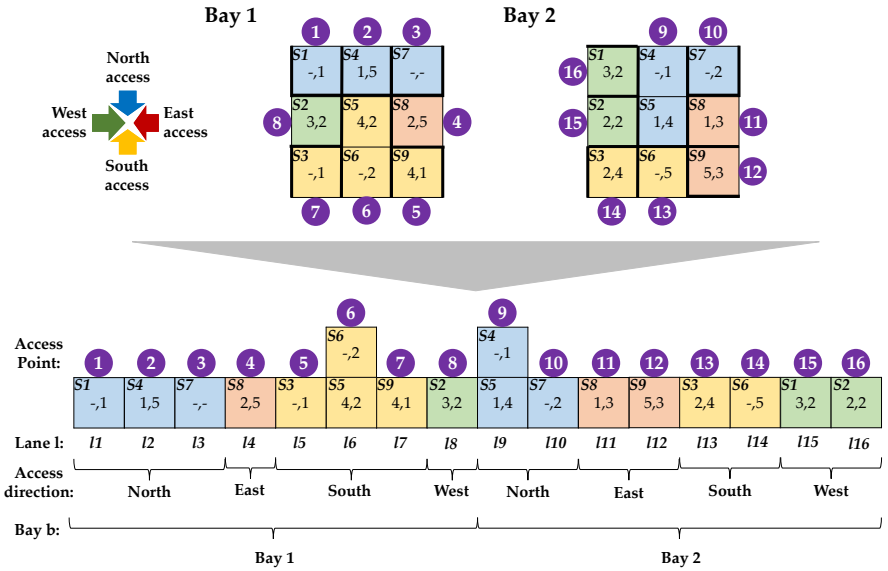


Fig. 5.6 Resulting virtual lanes for the example from Fig. 5.5. Thick black lines in bay 1 and bay 2 illustrate the virtual lanes. These lanes are transformed into a representation of one access direction.

list is empty and no solution exists, or until the time limit has been reached. In each iteration of the loop, a node n that minimizes the cost estimation $f(n) = g(n) + h(n)$ is popped off the open list, where $g(n)$ is the number of moves already performed at node n and $h(n)$ the lower bound estimating the number of moves until a blockage-free configuration is reached. If there are many nodes with the same cost estimation $f(n)$, we use several tie-breaking rules for node selection that are explained in the subsequent paragraphs. The selected node n is skipped if it has already been visited, and if not, it is added to the closed set. As long as we have not reached the final state, the successor nodes are calculated. The only difference for calculating successor nodes is that we incrementally update a total move distance. We use the same branching rules as in Section 4.3.2. Each successor node with a cost estimation $f(n)$ higher than the upper bound is skipped. A successor node is pushed to the open list and added to an f- and distance-dictionary, if there is not already a node with the same configuration in the f-dictionary that has a smaller cost estimation, or the same cost estimation but a smaller total move distance. Our procedure guarantees an exact solution for the minimization of moves. Because the travel distance is considered only for tie breaking, the resulting total move distance is not necessarily optimal. As long as the processing time for handling exceeds the travel time for each move, we expect good results for the total move processing time as well.

Algorithm 2: Modified A* algorithm from Chapter 4, Algorithm 1.

```
Function astar(root, upper_bound):
    open ← PRIORITY_QUEUE();                          ▷ new tie-breaking rules
    f_lookup ← DICT();
    distance_lookup ← DICT();                            ▷ new distance lookup
    closed ← ∅;
    PUSH(open, root);
    f_lookup[root] ← f(root);
    distance_lookup[root] ← 0;
    while open > 0 and time < timeout do
        n ← POP(open);
        closed ← closed ⋃ {n};
        if blocking(n) = 0 then
            return n;
        end
        successors ← BRANCHING(n);      ▷ extended branching over all bays
        for s ∈ successors do
            if f(s) > upper_bound then
                skip;                                          ▷ new upped bound
            end
            if s ∉ closed and (s ∉ f_lookup or f_lookup[s] > f(s) or (f_lookup[s] = f(s)
              and distance_lookup[s] > total_distance(s))) then
                PUSH(open, s);
                f_lookup[s] ← f(s);
                distance_lookup[s] ← total_distance(s);
            end
        end
    end
```

tie-breaking rules with travel distance: When selecting the next move, our A* algorithm in Algorithm 2 compares the different rules in a lexicographic order. The applied rules for tie breaking are the f-value (total cost estimation), h-value (lower-bound heuristics), and the total move distance *dist* (aggregated from the root to this node). The distance between two virtual lanes is defined as the length of the shortest path between their respective access points. The travel time within each virtual lane is neglected (it can be added by counting the empty tiles). We build different combinations based on these rules. In our approach, we always select the possible moves with the lowest f-value first. Because in many cases (especially large warehouses) there are still plenty of options with the same minimum f-value, we use the h-value and/or the total move distance *dist* as the first or second tie breaker. The variants include the following combinations: f_h_dist, f_h, f_dist, and f_dist_h.

Upper bound: Before running our main A* algorithm for determining the sequence of moves, we try to find an upper bound. To find an upper bound for the number of moves, we use a greedy depth-first search algorithm shown in

Algorithm 3. This step is optional. Although it entails additional computing time and effort, the main benefit is that it allows significant reduction of memory consumption when a solution is found, because all variants with a lower bound that is higher than the upper bound can be pruned. This may be helpful due to a large number of possible moves in bigger warehouses.

Starting from a root node based on an initial configuration of the warehouse, the algorithm generates all possible successor moves and picks one that minimizes the lower bound for the number of required moves. It then repeats the procedure until there are no blocking unit loads in the warehouse. The number of moves taken by this algorithm is always more than or equal to the optimal (minimum) number of moves. Therefore, it can be considered as an upper bound for our A* algorithm. The upper bound allows us to prune away unnecessary moves and avoid storing warehouse configurations resulting from these moves.

Algorithm 3: Greedy depth-first-search for finding an upper bound.

```
Function greedy(root):
    visited ← ∅;
    n ← root;
    visited ← root;
    while blocking(n)> 0 and time < timeout do
        successors ← BRANCHING(n);
        candidates ← successors \ visited;
        if |candidates| = 0 then
            return −1;
        end
        n ← min(candidates);
        visited ← visited ⋃ {n};
    end
    return f(n);
```

Incremental lower bound calculation: The A* search is further optimized by incremental lower-bound calculation, which is not visible in the algorithm. If the configuration and lower bound of a given node are known, calculating the lower bound of its successor can be simplified. The estimated number of required moves based on the BX and the GX moves can be computed faster if information about its predecessor is known. The number of misplaced unit loads can be changed only in the lanes impacted by a move. Therefore, the difference in misplaced unit loads within these lanes is also the total difference in misplaced unit loads in the entire warehouse. The BX moves based on the minimum number of misplaced unit loads per virtual lane can also be updated simply by comparing it to the number of misplaced unit loads in the virtual lanes affected by the most recent move. The calculation of additional GX moves uses a supply-and-demand model. The change of supply and demand is dependent only on the change in the virtual lanes

affected by the most recent move. To calculate the difference, the supply and demand of the two virtual lanes affected by the move must be determined for the parent node and for the successor. The difference in supply and demand is calculated by subtracting the parent's supply and demand from the successor's supply and demand. Based on the supply and demand differences, we calculate the cumulative incremental supply and demand, deriving the demand surplus difference. We then add the demand surplus difference to demand surplus of the parent node to obtain the new demand surplus of the successor. The following steps to derive the GX moves are the same as in Section 4.4.

5.3.4 Post-processing: Move sequence split algorithm

Block stacking warehouses with multiple bays may use multiple AMRs for pre-marshalling. Therefore, it is beneficial to split the moves into several sequences without dependencies that can be executed simultaneously. Such a partition requires information about which moves must be done sequentially.

We propose a solution that represents the move dependencies as a precedence graph. The nodes represent moves and the edges represent dependencies. The connected components of a resulting graph represent independent move sequences.

The moves are considered dependent if any of the following conditions are satisfied, as shown in Fig. 5.7:

a An earlier move places an item into the lane, while a latter move takes this item.
b An earlier move takes an item from the lane, while a latter move places a new item into the same lane.
c An earlier move takes an item from the lane, while a latter move takes the next item with a different priority group from the same lane.
d An earlier move places an item into the lane, while a latter move places the next item with a different priority group into the same lane.

Note that the chosen approach may create dependencies between the moves that are, in fact, independent. For example, under condition *a*, the lane might already contain a unit load of the same priority group, so the moves can be made in any order. Thus, we provide only a first set of rules that can be used to calculate an upper bound of independent sequences based on the precedence graph. There are surely further rules that would be able to split the precedence graph into more sequences. Also, dependencies should be updated over time. There may, for example, be a dependency in the very beginning that later splits up into several independent move sequences. Fig. 5.8 shows a simple example with four moves and the resulting configuration, as well as the move sequence. The first move must also be executed at first because the second and third moves go into the same virtual

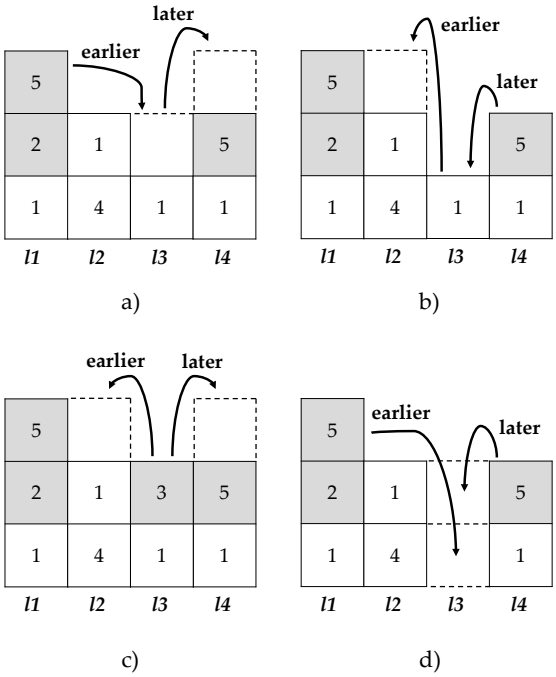


Fig. 5.7 Illustrations for the four different dependency conditions. Columns represent virtual lanes *l*, while arrows represent moves. Blocking unit loads are colored dark gray, and empty positions are illustrated with a dotted line.

lane. The sequence of the second and third moves is not relevant because both have the same priority group and both must be executed in order to perform the final, fourth move. In this simple example, only one independent move sequence is generated.

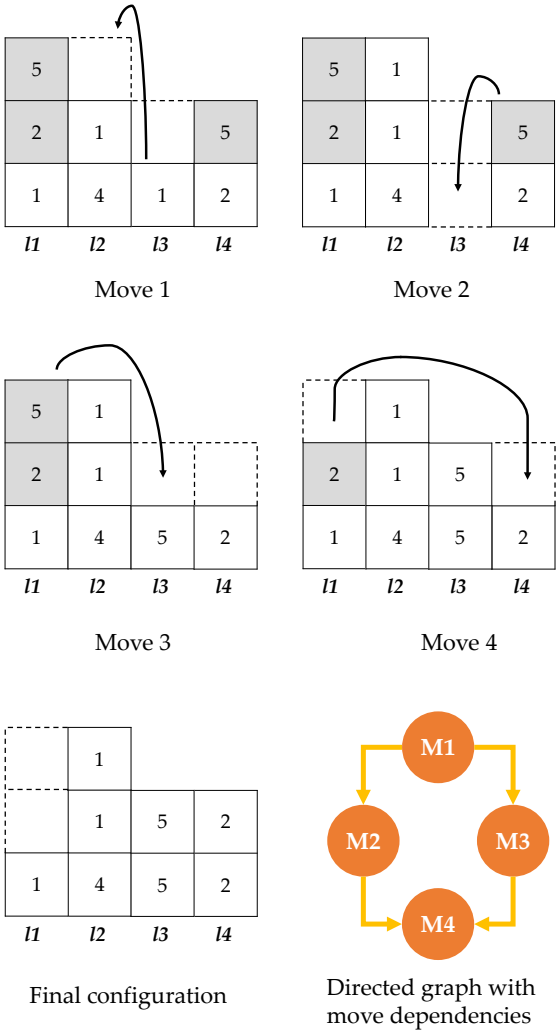


Fig. 5.8 Example of a resulting move sequence graphs with four moves. Columns represent virtual lanes l, while arrows represent moves. Blocking unit loads are colored dark gray, and empty positions are illustrated with a dotted line.

5.4 Computational experiments

In our experiment, we aim to investigate the performance of the proposed solution approach for the MUPMP in a large set of instances. We investigate the following three experiments based on separate datasets:

- Experiment and dataset *A*:
  - Description: Instances with a single tier and four access directions.
  - Goal: Investigation of different layout sizes, number of priority groups, fill levels as well as tie-breaking configurations for the search procedure.
- Experiment and dataset *B*:
  - Description: Instances with a variation of the defined access directions (single, opposite, and four access directions).
  - Goal: Investigation of different access directions based on two warehouse sizes and different fill levels.
- Experiment and dataset *C*:
  - Description: Instances with two tiers and four access directions.
  - Goal: Investigation of different layout sizes and fill levels.

Our analysis of the solved instances comprises runtime, total moves, total move distance and total move processing time. The total move processing time of the pre-marshalling procedure is calculated by a fixed handling time for each move and the average speed. In the interest of simplicity, we do not consider detailed movement times such as turnings and acceleration. Furthermore, we analyze the gap of our lower and upper bounds at the root node. The presented datasets are publicly available at https://doi.org/10.5281/zenodo.7272060. We execute our computational experiments on AMD EPYC 7401P CPUs at 2.45 GHz with 64 gigabytes of random-access memory (RAM) running Ubuntu 21.04. The algorithms are implemented in Python applying Gurobi 9.5 with the default parameters for both the network flow model and the MIP.

5.4.1 Instance generator

We develop a new instance generator for the MUPMP that places unit loads in a storage bay depending on the defined access directions. In addition to allowing up to four access directions (north, south, east, and west), we vary the number of columns, rows, and tiers of the storage bay, as well as the fill level. The instance sizes are inspired by real-world block stacking storage systems in production and logistics (e.g., buffer storage presented in Section 2.1.4). We model five priority groups that can occur with uniform probability. We place unit loads in the storage bay successively at random

and generate a new instance for each possible combination. During this procedure, it is necessary to consider the defined access directions to provide free storage locations that are directly accessible and to avoid gaps. The generated instances may be infeasible, especially for high fill levels.

The instance generator creates random stacks in a given warehouse; the resulting bays should have no *holes*. A *hole* is defined as an empty storage location that is inaccessible from the outside. The resulting number of unit loads should be as close as possible to the given *fill level*. The *fill level* is a ratio of generated unit loads to the total number of storage locations within a warehouse.

Step 1: To avoid holes, each bay is first split into random lanes. Each lane starts from an access point and ends at some stack within the bay. Thus, there will be no holes in the resulting unit load assignment if it is done lane by lane. The splitting process is iterative: In each iteration, the generator creates a random permutation of lanes and adds stacks to lanes in the order of permutation. If a lane reaches another lane or bay edge, it is skipped.

Step 2: After the lanes are fixed, the generator uses binomial distribution to obtain the number of unit loads in each lane. The parameters of the distribution are the number of storage locations in each lane and the given fill level.

Step 3: Once the number of unit loads is assigned to each lane, the generator randomly assigns priority groups to all unit loads based on a uniform distribution.

Because the resulting number of unit loads is random, post-processing is necessary to get as close as possible to the required fill level. The desired number of unit loads is calculated as a product of total storage space and fill level rounded to the nearest integer. Then, while the generated number of unit loads is higher or lower than desired, unit loads are removed or added respectively from randomly chosen lanes.

5.4.2 Datasets

We generate corresponding datasets for our three experiments. The first dataset A contains only warehouse layouts with bays that are accessible from all four access direction. The second dataset B provides instances to compare different access direction variants. Finally, the third dataset C includes instances with two tiers. Again, all four access directions are available for different bay and layout sizes. Due to the huge amount of possible sizes and shapes of rectangular bays and layouts, in our experiments we consider only quadratic footprints with a tile width and length of $1.4m$. Whereas we investigate various numbers of priority groups in dataset A, we keep five priority groups in dataset B and C.

Dataset *A***:** In dataset *A*, the dimensions start at a 3x3x1 (I x J x T) footprint for bays and a 2x2 layout of the warehouse (L x W). We scale the sizes until we reach timeouts for all variants. Besides the bay and layout size, we also vary the fill level and the number of priority groups. Table 5.1 shows all varied parameters that can be combined into different configurations. Each of the configurations is used to generate 10 instances based on distinct randomness seed values. Overall, dataset *A* contains 3.120 different instances based on four bay sizes with up to 12 layout sizes, four fill level percentages, three priority group amounts and 10 seed values.

Parameter	Bay 3x3x1	Bay 4x4x1	Bay 5x5x1	Bay 6x6x1
Access directions	{four}			
Layout Size (L x W)	{2x2, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9, 10x10, 11x11, 12x12}	{2x2, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8}	{2x2, 3x3, 4x4, 5x5, 6x6}	{2x2, 3x3, 4x4}
Fill level	{40%, 60%, 80% 90%}			
Priority groups	{5, 10, 20}			

Table 5.1 Varied parameters of the generated benchmark dataset *A* per bay size (I x J x T).

Dataset *B***:** We generate a second dataset *B* to compare the three different access direction variants shown in Fig. 5.9 for a 4x4x1 bay: single access, opposite access, and four access directions. For single access shown in Fig. 5.9 (a), which is commonly used in warehouses, the lane is simply split in the middle. Hence, lane depth and access direction for each stack are fixed without the need to apply the first step of our solution approach shown in Fig. 5.4 (in Fig. 5.9 (a): two stacks per lane in yellow and blue color for each access direction). In case of opposite (b) or multiple (c) access directions, access direction fixing for each stack via the network flow model must be applied (in Fig. 5.9 (b) and (c): resulting lane depth between one and three stacks per lane).

We generate random instances based on the parameters shown in Table 5.2. Dataset *B* contains 240 instances based on three access directions, two bay sizes, one layout size, four fill levels, one priority group, and 10 seed values. We use only even bay sizes (to obtain symmetrical lane depths) and the layout size for this experiment is small, being able to obtain a sufficient amount of solved instances for each access direction variant.

Parameter	Bay 4x4x1	Bay 4x4x2
Access directions	{single, opposite, four}	
Layout Size (L x W)	{2x2}	
Fill level	{40%, 60%, 80% 90%}	
Priority groups	{5}	

Table 5.2 Varied parameters of the generated benchmark dataset *B*.

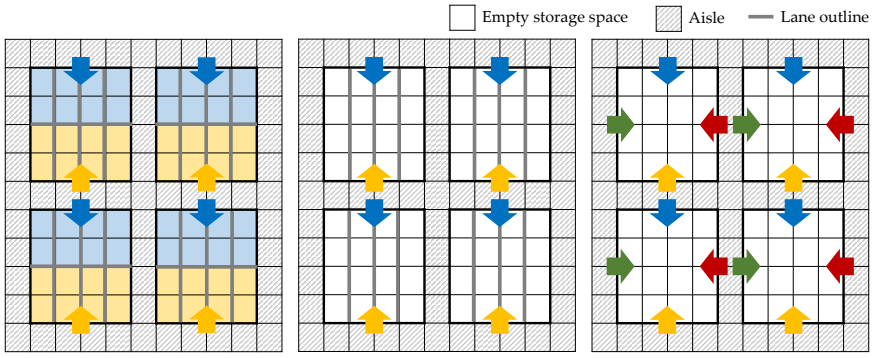


Fig. 5.9 Access direction variants (a) single, (b) opposite, and (c) four, of our second experiment based on dataset *B*.

Dataset *C*: The third dataset *C* contains instances with two tiers. Table 5.3 shows the parameters of dataset *C*. The instances are based on three bay sizes and all layout sizes until we reach the limits of solvability. The total number of instances comes down to 280 based on: Seven layout-bay combinations, four fill levels, a single priority group, and 10 seed values.

Parameter	Bay 3x3x2	Bay 4x4x2	Bay 5x5x2
Access directions		{four}	
Layout Size (L x W)	{2x2, 3x3, 4x4, 5x5}	{2x2, 3x3}	{2x2}
Fill level		{40%, 60%, 80% 90%}	
Priority groups		{5}	

Table 5.3 Varied parameters of the generated benchmark dataset *C* per bay size (I x J x T).

5.4.3 Results for experiment *A*

In this experiment, we use the 3,120 instances from dataset *A* and run each instance four times based on different tie-breaking rules for our A*-search. In all four variants, a successor node is picked by first minimizing the f-value. As a primary and secondary tie-breaking rule, we use variants based on the minimization of our lower-bound heuristics h and the minimization of the total move distance *dist*, which results in the four rules: f_h, f_h_dist, f_dist, and f_dist_h. The tie-breaking rule f_h is our baseline from Chapter 4. The time limit of the search procedure for all 12,480 runs is set to 10 minutes. In the following, we analyze the results, starting with an analysis of the tie-breaking rules (*A*1). We then present detailed results of the best-performing tie-breaking rule and give an overview for the number of solved instances and the runtime (*A*2), followed by an evaluation of the number of moves as well as total move distance and total move processing time (*A*3). Finally, we

also investigate the performance of our lower and upper bound ($A4$) and the resulting move sequence based on our post-processing algorithm ($A5$).

$A1$: Tie-breaking rules

Table 5.4 provides an aggregated summary and Fig. 5.10 an overview with respect to bay size for the number of solved instances for each tie-breaking rule. Both show that the number of solved instances with total move distance as the first tie-breaking rule f_dist and f_dist_h are outperformed by the other variants using the lower-bound heuristics h. With the minimization of h as a first tie-breaking rule in f_h and f_h_dist, almost twice as many instances could be solved within the time limit. Regarding the second tie-breaking rule, adding h has only a small positive effect for f_dist_h versus f_dist. However, adding the distance as a second tie-breaking rule has no effect on the number of solved instances (with the exception of one instance).

A detailed overview of the results for all variants is provided in the Appendix Table A.3.

tie-breaking rule	No. solved	No. infeasible	No. timeout
f_dist	1711	0	1409
f_dist_h	1723	0	1397
f_h	3005	0	115
f_h_dist	3006	0	114

Table 5.4 Results summary for a single tier showing the number of solved, infeasible, and unsolved instances (in 12,480 total runs with 3,120 instances for each tie-breaking rule).

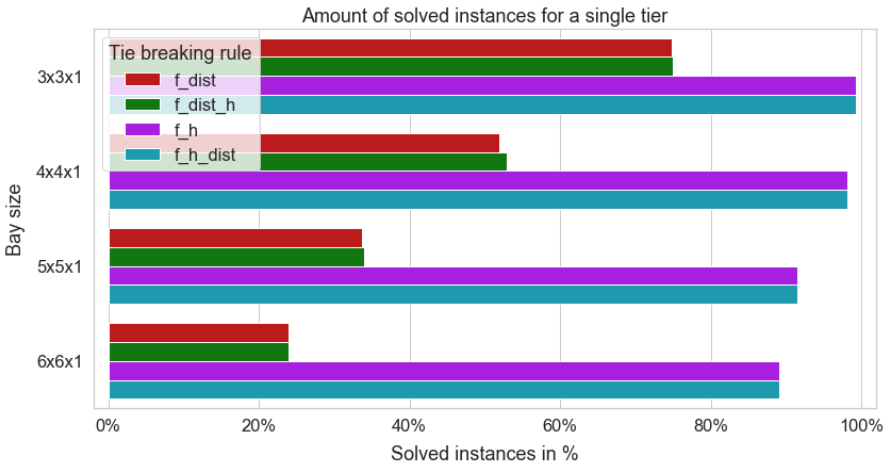


Fig. 5.10 Solved instances per bay size for each tie-breaking rule.

Fig. 5.11 and Fig. 5.12 show the results in terms of runtime and travel distance for the instances that have been solved by all tie-breaking rules. In accordance with the presented number of timeouts f_dist shortly followed by f_dist_h have much higher runtimes than f_h and f_h_dist. Between f_h and f_h_dist the difference in runtime is marginal (<0.01 seconds in mean). In terms of distance f_h performs worst, since distance is not taken into account. The other three variants are on the same level with the exception of f_h_dist being marginally higher (mean value difference of approx. $0.1m$; $5.0542m$ versus $5.1557m$).

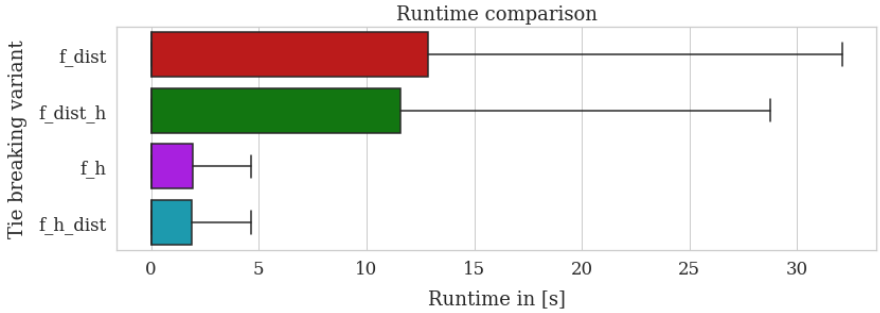


Fig. 5.11 Box plots (without outliers) of the runtime for each tie-breaking rule based on instances that have been solved by all.

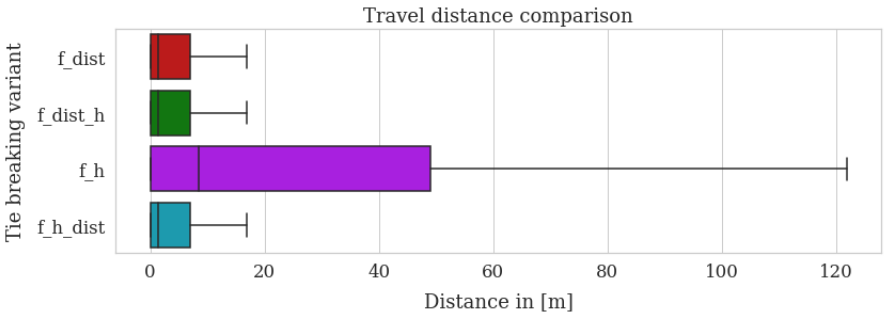


Fig. 5.12 Box plots (without outliers) of the total move distance for each tie-breaking rule based on instances that have been solved by all.

Based on the analysis of the tie-breaking rules, we conclude that f_h_dist is most suitable for multibay pre-marshalling because although we solve as many instances as the f_h variant, we are able to cut the total move distance by a factor of more than five in mean. The tie-breaking rules with distance as the first tie breaker f_dist and f_dist_h fail to solve a sufficient number of instances. Hence, the detailed results in the following sections are based on tie-breaking rule f_h_dist.

*A*2: Runtime and number of instances solved

Table 5.5 gives an overview of the number of solved, infeasible, and unsolved (timeout) instances for all fill levels and numbers of priority groups. High fill levels are especially challenging, which show 73 out of 114 timeouts for a fill level of 90%. The other fill levels with timeouts are 60% with eight timeouts and 80% with 33 timeouts. Infeasible instances could not be verified because even for 90% fill levels, a sufficient number of empty slots are available to resolve any situation in all considered multibay scenarios. Increasing the number of priority groups may be necessary for larger warehouse setups. Our experiment shows that a greater number of priority groups leads to more unsolved instances. An increase from five, 10, and up to 20 priority groups leads to an increase in timeouts of 14, 44, and up to 56. Increasing the priority groups leads to more misplaced unit loads in our randomly generated instances.

Fill level	Priority groups	No. solved	No. infeasible	No. timeout
40%	5	260	0	0
40%	10	260	0	0
40%	20	260	0	0
60%	5	260	0	0
60%	10	258	0	2
60%	20	254	0	6
80%	5	255	0	5
80%	10	248	0	12
80%	20	244	0	16
90%	5	251	0	9
90%	10	230	0	30
90%	20	226	0	34

Table 5.5 Results summary for a single tier showing the number of solved, infeasible, and unsolved instances for tie-breaking rule f_h_dist (in total 3,000).

Fig. 5.13 provides an overview of the solved instances per bay size for each layout size. Naturally, the maximum layout size that we are able to solve diminishes with increasing bay size. In terms of the total warehouse capacity, we are able to solve larger warehouses as the bay size becomes smaller (e.g., bay size 3x3x1 up to a layout size of 11x11: 1,089 storage locations; bay size 4x4x1 up to a layout size of 7x7: 784 storage locations). This is because smaller bay sizes provide better accessibility to storage locations. However, we note that even though the warehouse capacity is the same, the footprint is different, because for a higher number of small bays, more aisle space is required for traveling. Figures with the solved instances per bay and warehouse size for

each fill level are provided in Appendix A.1: Fig. A.15, Fig. A.16, Fig. A.17, and Fig. A.18.

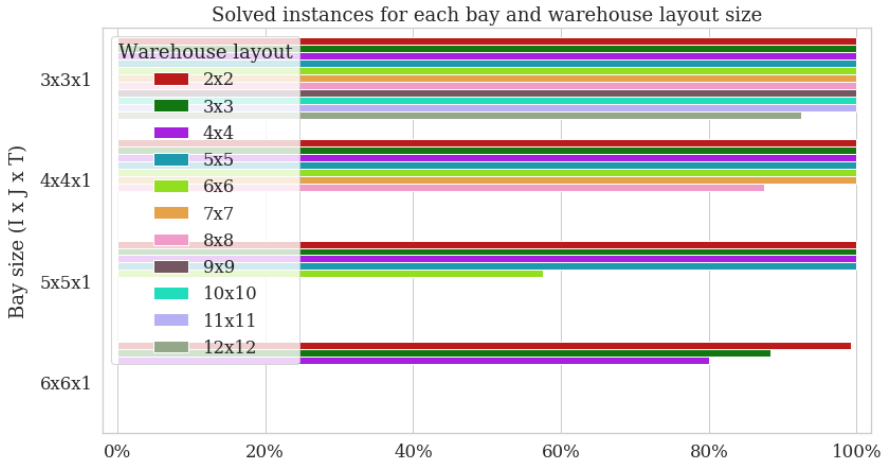


Fig. 5.13 Solved instances per bay size for each layout size.

Fig. 5.14 presents the results of the runtime of the solved instances in seconds per bay and layout size. Because we set a runtime limit of 10 minutes for the search loop, the limit is surpassed in only a few cases before the loop reaches the time constraint. For the largest warehouse layout sizes that we were able to solve for each bay size, we see a much wider range of runtimes. Bay sizes 3x3x1 and 4x4x1 yield the shortest runtimes, remaining close to zero. The runtimes increase with the instance size.

Fig. 5.15 and Fig. 5.16 show the box plots for the runtimes of each fill level and each number of priority groups, respectively. Unsurprisingly, a higher fill level leads to longer runtimes. The lower third quartile and maximum value of 90% fill level, compared to 80% fill level, can be explained by a smaller number of solved instances due to timeouts for the most difficult instances. A larger number of priority groups leads to a relatively moderate increase in runtimes.

*A*3**: Number of moves, total move distance and total move processing time**

The 3,006 solved instances for tie-breaking rule f_h_dist require a mean of 6.81 moves for the whole pre-marshalling operation. Fig. 5.17 shows the number of moves for each bay and layout size. We were surprised by the relatively low amount of sorting effort in terms of required moves even for large warehouses. Within our time limit of 10 minutes, we are able to solve instances with up to 58 moves. It is clear that the number of moves is increasing more slowly for small bay sizes than for larger ones, which also

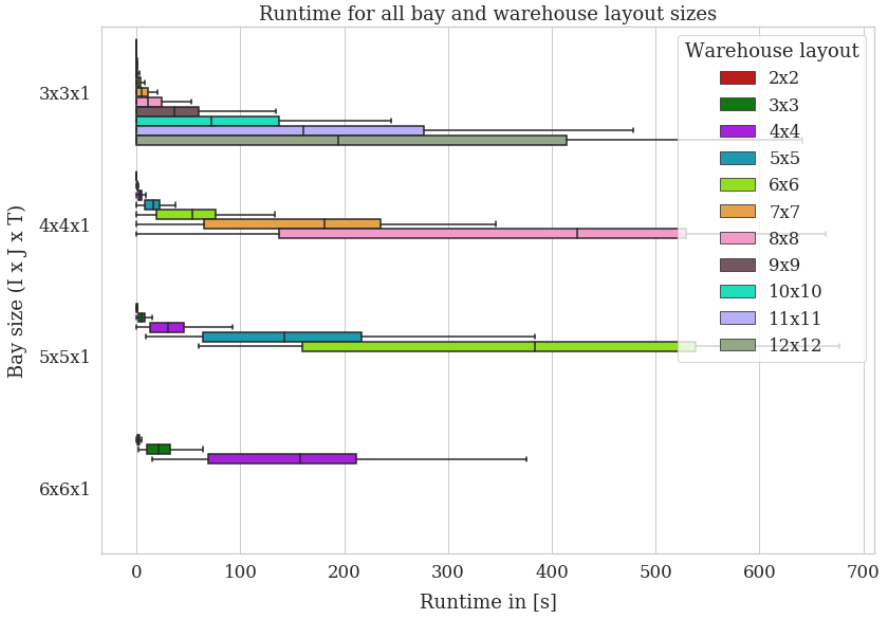


Fig. 5.14 Box plots (without outliers) for the runtime of solved instances in seconds per bay size for each layout size.

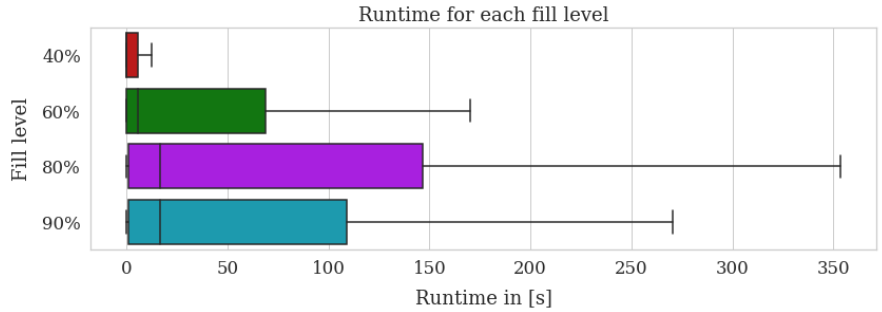


Fig. 5.15 Box plots (without outliers) for the runtimes of solved instances for each fill level.

makes sense when looking at the proportionally higher growth in warehouse capacity.

In Fig. 5.18, we compare different combinations of bay and layout sizes that result in the same warehouse capacity. The graph shows that a small bay size helps to reduce the sorting effort. The greater the difference in the bay sizes, the greater the increase in the number of moves (trade-off between aisle space and sorting effort).

Fig. 5.19 shows the box plots for the number of moves for each fill level. The mean number of moves increase with each fill level from 1.07 for 40% fill level, to 4.16 for 60% to 10.06 for 80% up to 12.61 moves for 90%. The increase

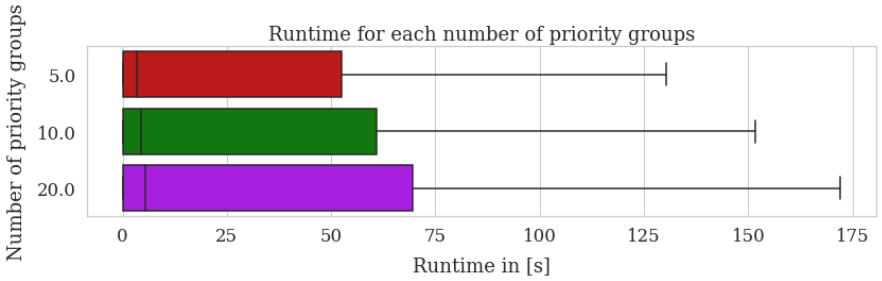


Fig. 5.16 Box plots (without outliers) for the runtimes of solved instances for each amount of priority groups.

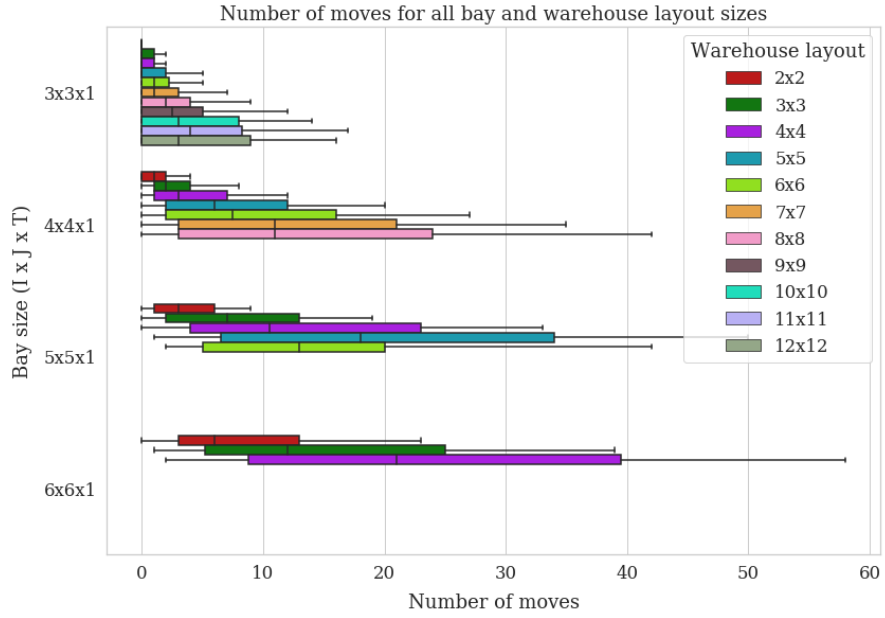


Fig. 5.17 Box plots (without outliers) of the number of moves per bay size for each layout size.

is clearly visible, but still moderate even from 80% to 90%. We expect a more pronounced increase when only few open storage locations are available for rearrangement operations. It appears that 90% fill level is not the limit for our multibay instances.

Another observation concerns the number of priority groups. In Fig. 5.20, we draw the curve progressions for the mean number of moves for each amount of considered priority groups five, 10 and 20. We only consider instances that have been solved by all three priority group variants. The increases of all three curves over the bay sizes are quite linear, with the exception of a small bend of the curves at bay sizes 5x5x1 to 6x6x1, which is

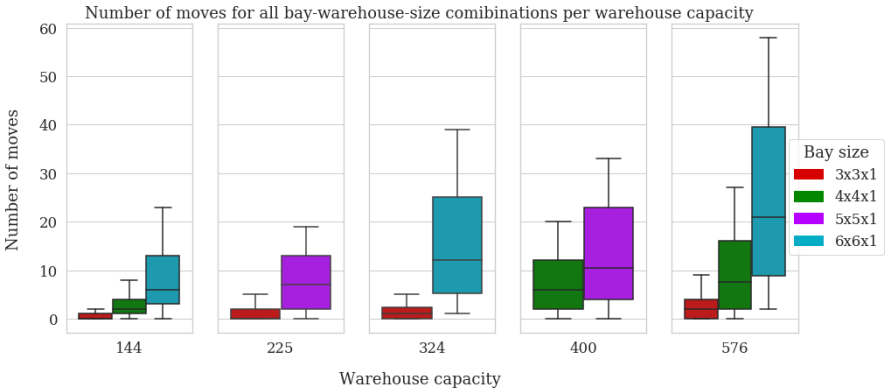


Fig. 5.18 Box plots (without outliers) of the number of moves for all warehouse capacities with several combinations of bay and layout sizes.

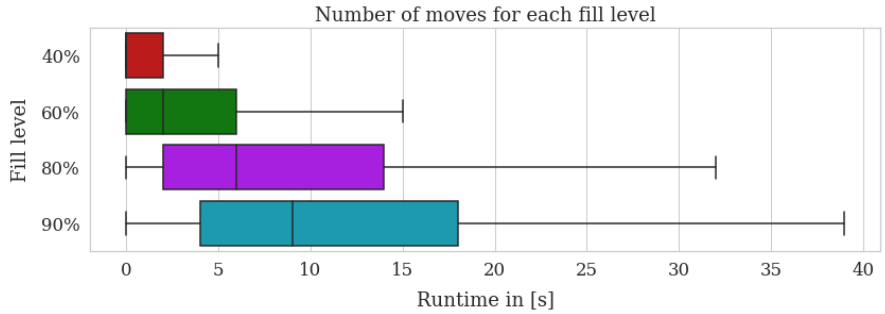


Fig. 5.19 Box plots (without outliers) for the number of moves for each fill level.

likely caused by a smaller amount of solved instances at bay size 6x6x1 with a high fill level. The differences between the curves are the starting points and the slopes. The gap between curves is growing with each enlargement of bay size. It appears logical to conclude that larger/deeper bays suffer more in sorting effort when handling a larger number of priority groups compared to only few priority groups. Doubling up the the amount of priority groups two times, from five to 10 and from 10 to 20, also has a different impact on the number of moves. Doubling the priority groups from 5 to 10 increases the mean moves from 5.31 to 6.50. A quadrupling to 20 priority groups leads to a mean of 7.12 moves. Hence, the impact/difference in mean number of moves decreases as the number of priority groups increases: The difference in mean number of moves is much lower when going up from 10 to 20 compared to from 5 to 10. As a supplement, the box plots of the number of moves for each priority group is shown in Fig. A.20.

The total move distance for the pre-marshalling moves ranges from zero to 765.80m with an overall mean of 41.01m. Fig. 5.21 shows box plots and a linear regression model for the total move distance over the number of moves.

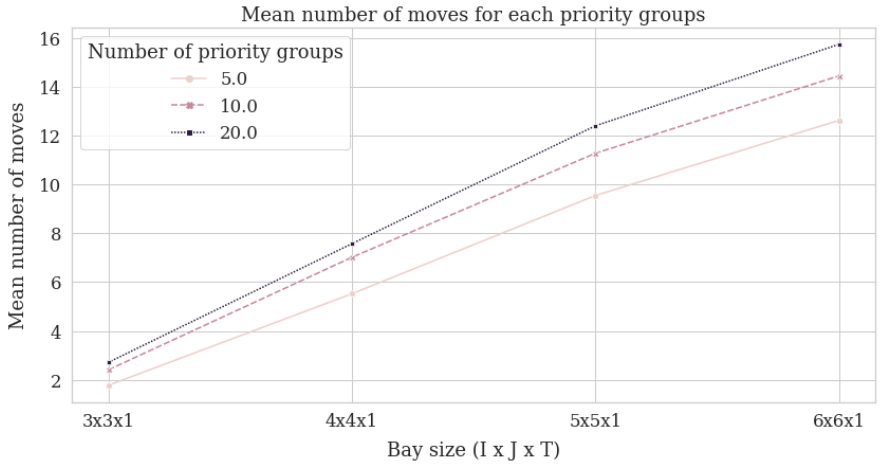


Fig. 5.20 Mean number of moves for each number of priority groups per bay size (tie-breaking rule f_h_dist). Only instances that are solved by all three variants are included.

The variance in distance increases alongside an increasing number of moves. The increased distances and increased number of moves are caused mainly by the larger bay sizes but, of course, also by the larger warehouse layout and overall instance sizes. Fig. A.19 in Appendix A.1 shows the detailed results as box plots of the total move distance for each bay and warehouse layout size.

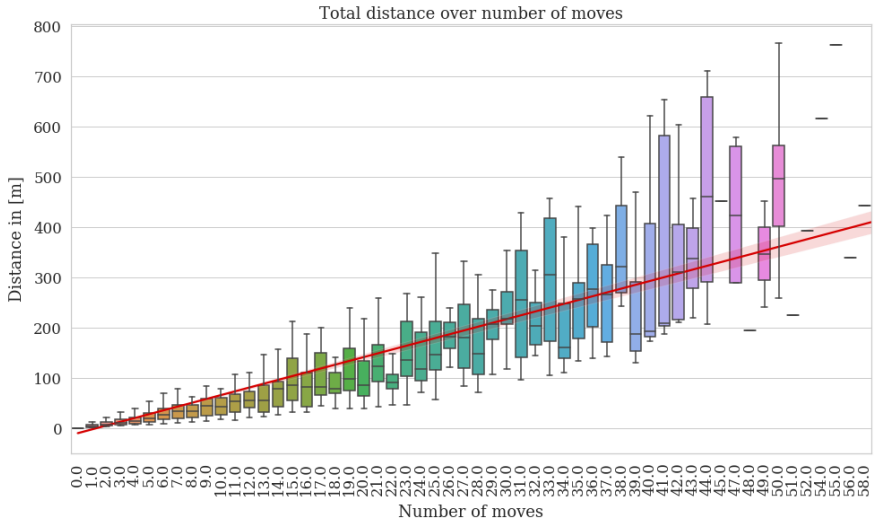


Fig. 5.21 Box plots (without outliers) and a linear regression model of the total move distance (in meters) over each number of moves.

When again comparing the bay–warehouse layout size combinations with the same warehouse capacity for the total move distance in Fig. 5.22, we can see a similar effect as for the number of moves. Warehouses with smaller bays achieve shorter travel distances. The effect is slightly less pronounced than for the number of moves because it is partially cannibalized by the larger footprint necessary for the additional aisle space in warehouses with smaller bays.

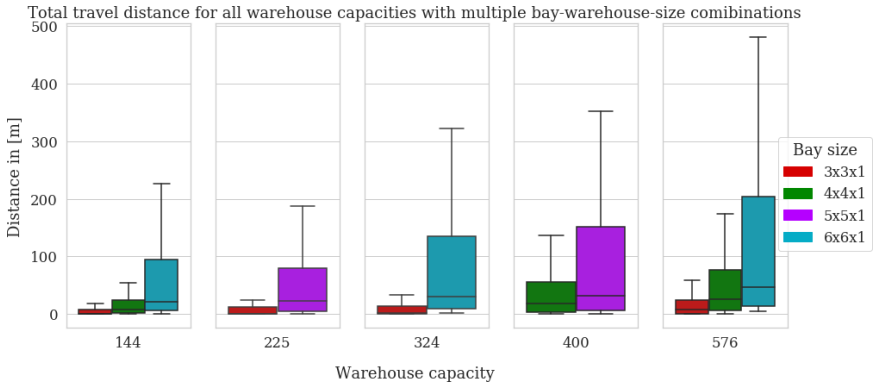


Fig. 5.22 Box plots (without outliers) of the total move distance (in meters) for all warehouse capacities with several combinations of bay and layout sizes.

Looking at the box plots of the different fill levels in Fig. 5.23, the mean and variance of the total move distance increase strongly with each fill level. The mean total move distance increases from 1.82m for 40% fill level, to 10.63m for 60%, to 46.76m for 80%, and up to 111.36m for 90%. Whereas the mean moves increase from 80% to 90% fill levels by just 25.35%, the total move distance increases by 138.15%. This means that with a higher fill level, the effort in terms of required travel distance increases much more than in terms of mean moves. Also, for higher numbers of priority groups, the total move distance increases slightly. Fig. A.21 and Fig. A.22 shows the box plots and curve progressions of the total move distance for each number of priority groups.

The sum of handling and traveling time resulting in the total move processing time is shown in Fig. 5.24. We assume an average speed of 1.4m per second and pick-up/drop-off times, including fine positioning, adding up to a constant of 60 seconds per move. The required total move processing time to conduct the moves does not exceed 65 minutes. The mean processing time over all instances is 437.96 seconds. Even when considering additional times for empty travels that are not yet included, we do not expect an unreasonable time span for the pre-marshalling of all instances.

The evaluation of the fill levels, the priority groups, and bay-layout combinations of the same warehouse capacities lead to similar conclusions as

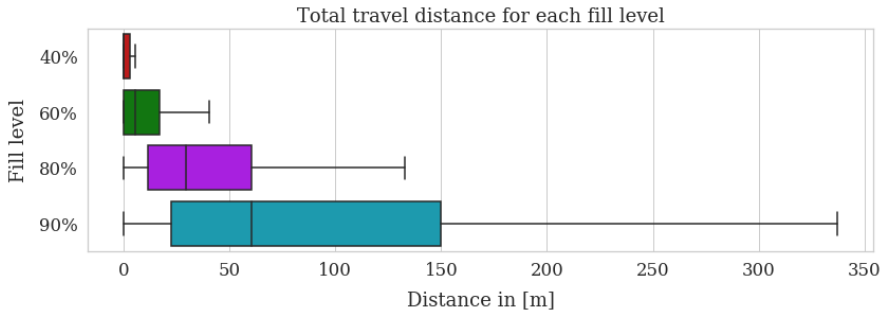


Fig. 5.23 Box plots (without outliers) of the travel distances (in meters) for each fill level.

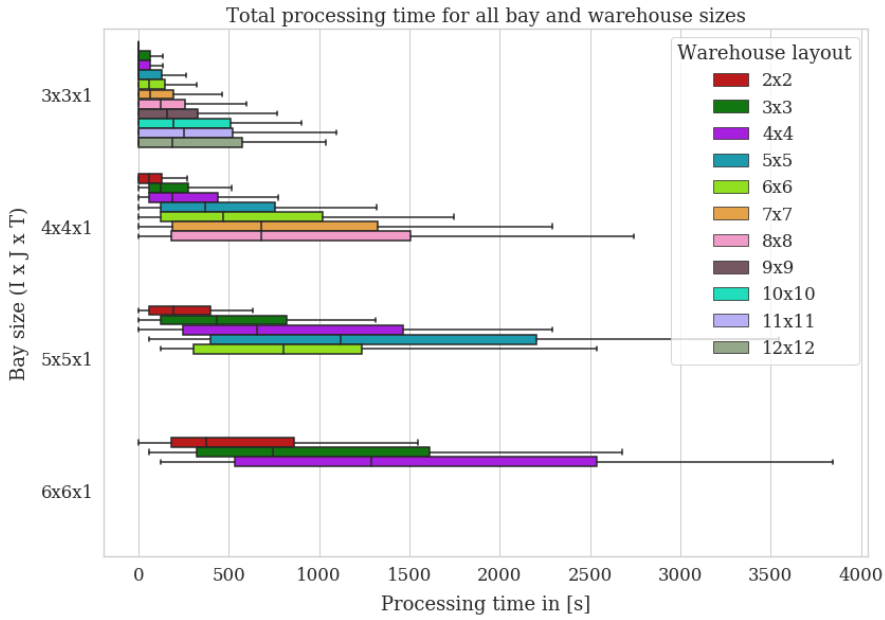


Fig. 5.24 Box plots (without outliers) of the total move processing time in seconds per bay size for each layout size.

previously discussed for the travel distance and number of moves. We refer to the respective sections and provide the figures in Appendix A.1; see Fig. A.23, Fig. A.24, Fig. A.25, and Fig. A.26.

*A*4**: Lower and upper bound**

The lower bound proves to be very accurate. The gap between the lower bound in the root node and the resulting final moves is 0%, which means that the lower bound is 100% accurate in all 3,006 solved instances. The main reason for this extremely strong lower bound may be that the large number

of possible moves for multibay scenarios leads to fewer BB and GX moves. BB and GX are harder to predict for our lower-bound heuristics. Ultimately, this theory requires further investigation.

The greedy search with a time limit of 2 minutes found a valid upper bound for 2,467 instances (see for each bay and warehouse layout size Fig. A.27 and for the different number of priority groups and bay sizes in Fig. A.28). However, in the case of challenging instances with a high number of required moves (> 35), it was not able to find a solution in the given time. Whereas the upper bound was very strong compared to the final required number of moves for fill levels up to 80% (0% gap), for the 90% fill level we encountered a few differences of up to six moves (see Fig. A.29). Fig. 5.25 shows two histograms of the instances with and without an upper bound, plotted over the required number of moves. It can be seen already that for instances we solved with a very low amount of moves, an upper bound could not always be found. (Box plots for the distribution for each bay and warehouse layout size are provided in Fig. A.30.) Although a relaxation of the time limit would most likely help us find more solutions, we doubt that a reliable output can be achieved for all instances. Because the greedy-search approach for finding an upper bound helps us to save memory only in some instances, we do not think that spending the extra time calculating the upper bound is worthwhile. It may be worth considering other methods to save memory, such as IDA*-search [TPV17].

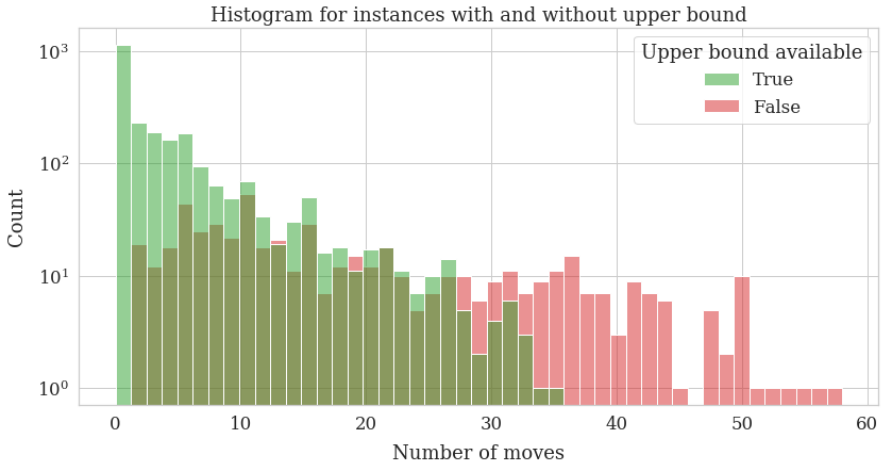


Fig. 5.25 Histogram of the instances with and without upper bounds that could be found, over the number of moves.

*A*5: Move sequences

Our proposed algorithm for post-processing splits the generated move sequence into as many as 49 independent move sequences. Overall, the mean number of independent move sequences is 6.27. The number of move sequences increases with bay and layout size (see Fig. A.31). Of the independent move sequences, the longest sequence contains 19 moves and the mean is 0.98, due to a number of instances with zero moves that are already sorted. Fig. 5.26 gives an overview of the longest move sequence with box plots for all bay and layout sizes. For a bay size of 3x3x1 and 4x4x1, the longest sequence length is predominantly one move. Hence, a parallelization of premarshalling can be achieved. For deeper bays, the longest sequence increases further as more dependencies between moves are existent.

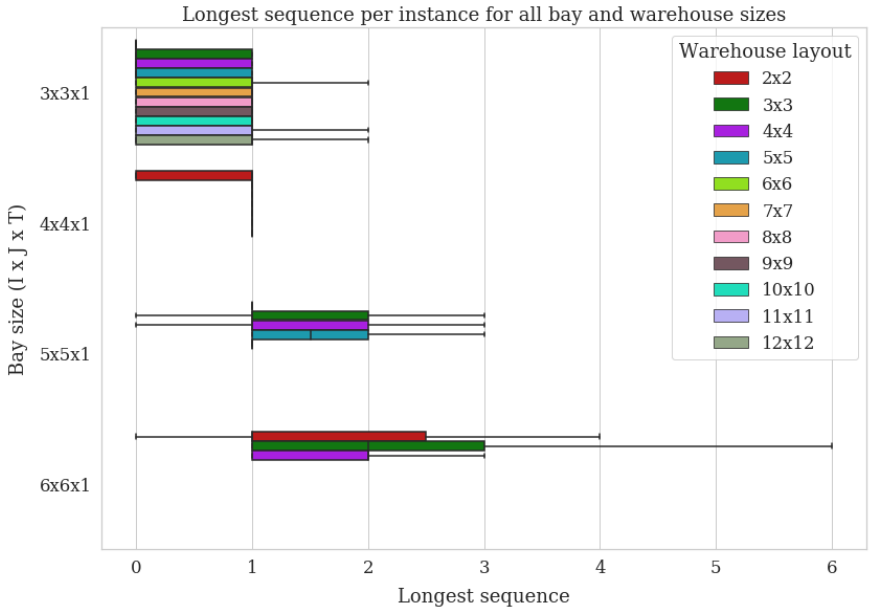


Fig. 5.26 Box plots (without outliers) of the longest sequence per instance for all bay and layout sizes.

5.4.4 Results for experiment *B*

Experiment *B* is based on a smaller dataset with 240 instances with layout size 2x2, because a fair comparison requires as many solved instances as possible for all of the access directions. Similar to the results of experiment *A*, we give an overview for solved instances and runtime *B*1 and analyze the results for the number of moves, total move distance, as well as total move

processing time $B2$. All runs are based on a search configuration utilizing tie-breaking rule f_h_dist.

$B1$: Runtime and number of instances solved

Table 5.6 shows the number of solved, infeasible, and timeout instances. All instances with four access directions could be solved, followed by the instances with opposite directions (11 timeouts) and the instances with a single access direction (17 timeouts). Again, infeasibility could not be proven.

Access direction	No. solved	No. infeasible	No. timeout
Single	63	0	17
Opposite	69	0	11
Four	80	0	0

Table 5.6 Results summary showing the number of solved, infeasible, and unsolved instances (80 per access direction; 240 in total).

In Fig. 5.27 we see a detailed breakdown of the percentage of solved instances per access direction for each bay size. At a bay size of 4x4x1, all instances could be solved. Increasing the bay size to 6x6x1 leads to multiple timeouts for single and opposite access, especially at higher fill levels.

The mean runtimes at bay size 4x4x1, where all instances got solved, are 0.03 seconds for four access directions, 0.34 seconds for two opposite access directions, and 0.64 seconds for a single access direction. Looking at the instances with bay size 6x6x1, the picture is changing as we achieve higher mean and maximum values for two opposite access directions compared to a single direction. This is also visible in Fig. 5.28, showing the runtime box plots for each access directions per bay size. This is because for two opposite directions, we simply found more feasible solutions for instances with higher fill levels that require longer computation times.

$B2$: Number of moves, total move distance and total move processing time

Fig. 5.29 shows the required number of moves for each access direction and both bay sizes. More access directions lead to fewer required moves and a smaller variance of the distribution to sort the multibay warehouse. At bay size 4x4x1, the mean number of moves increases from 0.88 for four access directions, to 3.90 for two opposite access directions, up to 6.95 for a single access direction. Whereas in case of four access directions, all instances could be solved with a maximum of four moves, the maximum increases to 10 for opposite access directions and to 20 for a single access direction.

The results for the total move distance in Fig. 5.30 also show an increase in the total move distance as well as a growing variance for fewer access directions. At bay size 6x6x1, the differences between the access directions

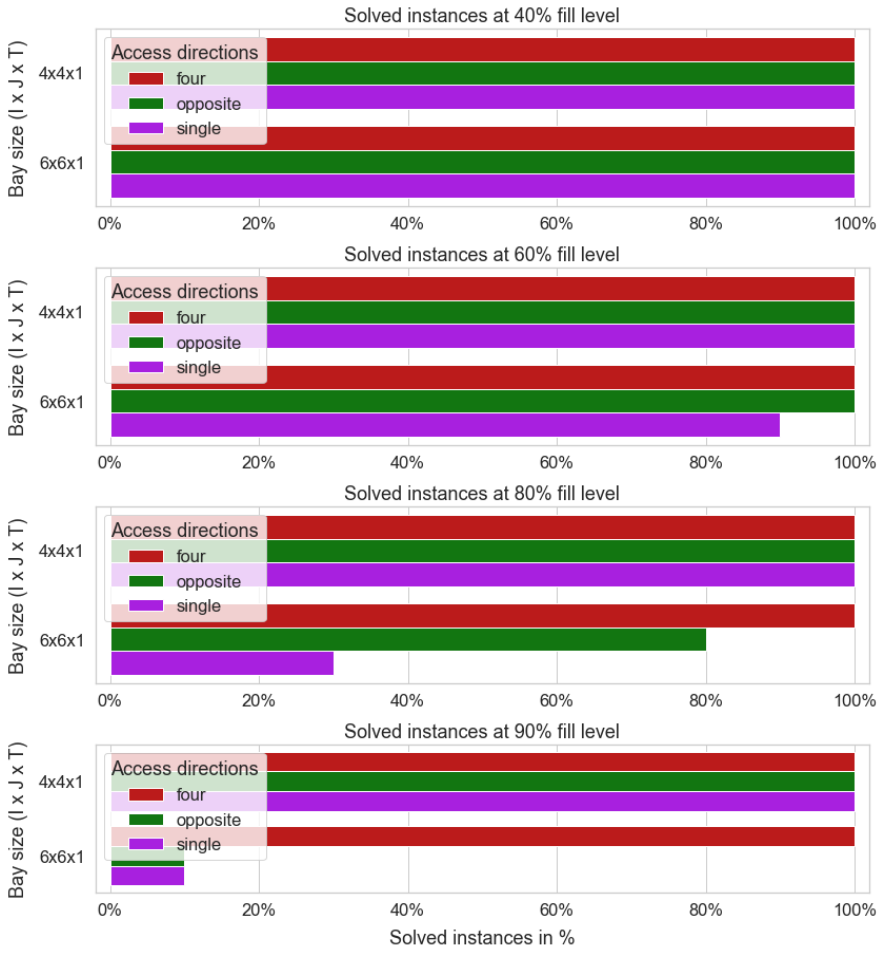


Fig. 5.27 Overview of solved instances for the multibay scenario with single, two opposite, and four access directions for each fill level over the bay sizes at a layout size of 2x2.

are less distinct due to the greater number of unsolved instances. For a bay size of 4x4x1, the mean total move distance is reduced from 42.95m (single access direction) to 24.19m (opposite access directions) and 4.94m (four access directions), and the maximum travel distance is cut by 78% from 177.8m to 89.6m and 39.2m.

The results for total move processing time are similar to the number of moves and the total move distance (see Fig. A.32). The maximum processing times are 50.88min (single access direction), 35.02min (opposite access directions), and 21.65min (four access directions): a reduction of 57%.

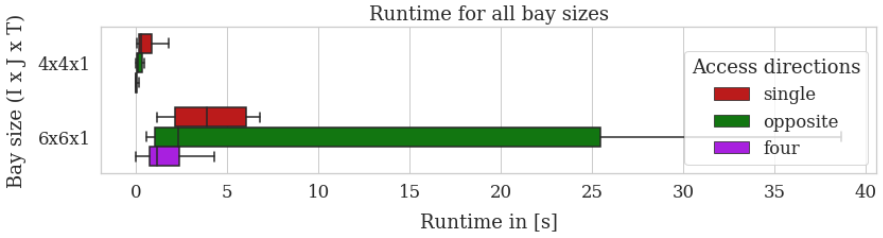


Fig. 5.28 Box plots (without outliers) of the runtime in seconds per bay size for each access direction.

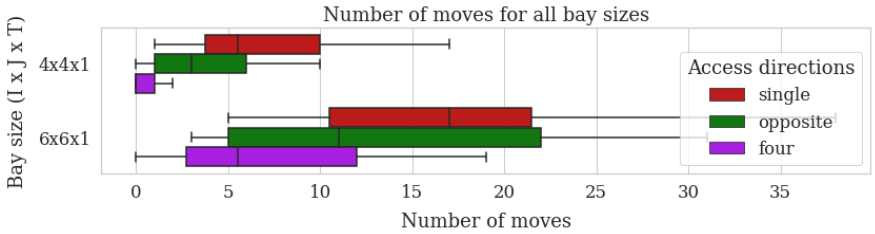


Fig. 5.29 Box plots (without outliers) of the number of moves per bay size for each access direction.

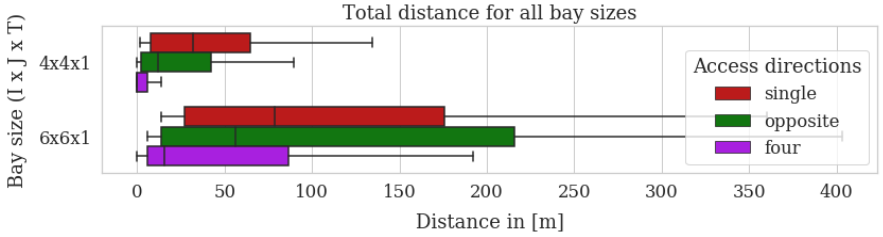


Fig. 5.30 Box plots (without outliers) of the total move distance in meters per bay size for each access direction.

*B*3: Lower and upper bound

The lower-bound heuristics are tight for any of the three access directions, achieving a gap of 0% in the root node compared to the final result. This comes as a bit of a surprise, because we encountered significant differences between the access directions for a single bay. As already discussed in experiment *A*, the greater number of possible moves may be beneficial. Perhaps some of the instances that could not be solved also lack tightness of the lower-bound heuristics. However, this needs further investigation with long runtimes.

The greedy search for an upper bound was able to find a feasible solution for all 112 solved instances. A difference between upper bound and required number of moves could be found most often for the instances with two opposite directions at high fill levels of 80% and 90% (see Fig. A.33).

*B*4: Move sequences

The number of independent move sequences increases with each additional access direction. Starting with a mean of 3.08 for four access directions, it goes up to 5.07 for opposite directions and to 6.74 for a single access direction. Fig. 5.31 shows box plots for the number of independent move sequences over all bay sizes and access directions.

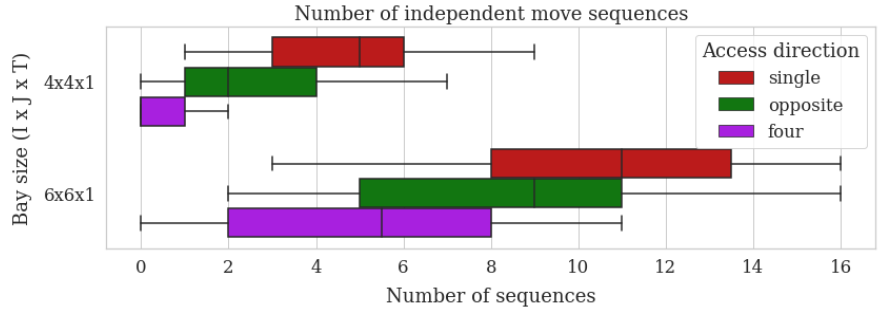


Fig. 5.31 Box plots (without outliers) of the number of independent sequences for all bay sizes and access directions.

Although the number of independent move sequences increases for fewer access directions, the length of theses sequences also increases, as illustrated in Fig. 5.32. This is mainly due to a much higher number of moves that are necessary to solve these instances, caused by deeper average lanes. The longest sequence per instance increases from an overall maximum of nine (for four access directions), to 17 (for opposite access directions) and to 27 for a single access direction (maximums are not shown in Fig. 5.32).

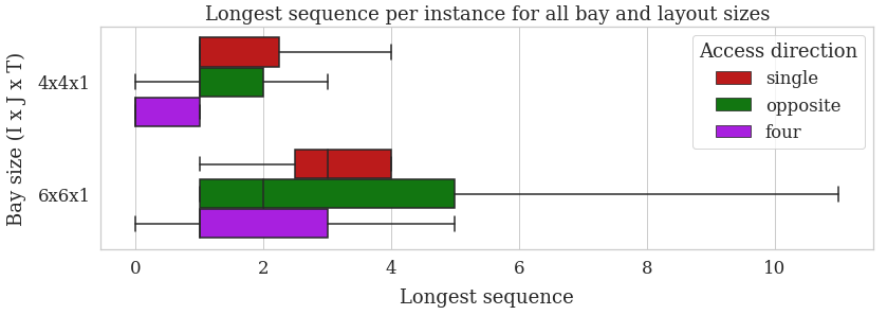


Fig. 5.32 Box plots (without outliers) of the longest sequence per instance for all bay sizes and access directions.

5.4.5 Results for experiment *C*

In the third experiment, we analyze the results for instances with two tiers. The dataset contains 280 instances based on several bay and warehouse layout sizes. We increase the sizes until the limits and no sufficient amount of solutions can be found. Our analysis follows the known structure of the previous sections, and it is split into the overview for solved instances and runtime *C*1 and the investigation of number of moves, total move distance, as well as total move processing time *C*2. We briefly summarize the results for the lower and upper bound *C*3 as well as the move sequence analysis *C*4. All runs are utilizing tie-breaking rule f_h_dist. In the case of a single bay Section 4.5.2, we have already seen that two tiers make it much more complicated to find feasible solutions. The chance of an unsorted stack at the very beginning is already high, and all unit loads that are placed in front of this stack within the same lane must be relocated. Our results for multibay warehouses correspond to the previous findings.

*C*1**: Runtime and number of instances solved**

For the instances with two tiers, we are able to solve 194 instances; we find no infeasible instances and obtain 86 instances with timeouts. In Table 5.7, the numbers of solved, infeasible, and timeout instances are listed based on the four fill levels. For fill levels of 40% and 60%, all instances could be solved. At a fill level of 80%, we obtain 31 timeouts. This increases further to 55 timeouts at 90% fill level, which represents 78.6% of the instances.

Fill level	No. solved	No. infeasible	No. timeout
40%	70	0	0
60%	70	0	0
80%	39	0	31
90%	15	0	55

Table 5.7 Results summary showing the number of solved, infeasible, and unsolved instances (70 per access direction; 280 in total).

Taking a closer look at the different bay and layout sizes in Fig. 5.33 reveals that only bay size 3x3x2 at layout size 2x2 could be solved at all fill levels. Bay size 4x4x2 was barely solved at 90% fill level, and we were unable to solve any instances within the time limit for 5x5x2. When we look at the warehouse capacities, the decline compared to a single tier is substantial. For a bay size of 3x3x1 with a single tier, we solve all instances with warehouse layout sizes up to 11x11, resulting in a warehouse capacity of 1,089 storage locations. For two tiers, we solve a warehouse with only 72 storage locations. Already at

warehouse capacities of 128 (4x4x2, layout size 2x2) and 162 (3x3x2, layout size 3x3), not all instances could be solved.

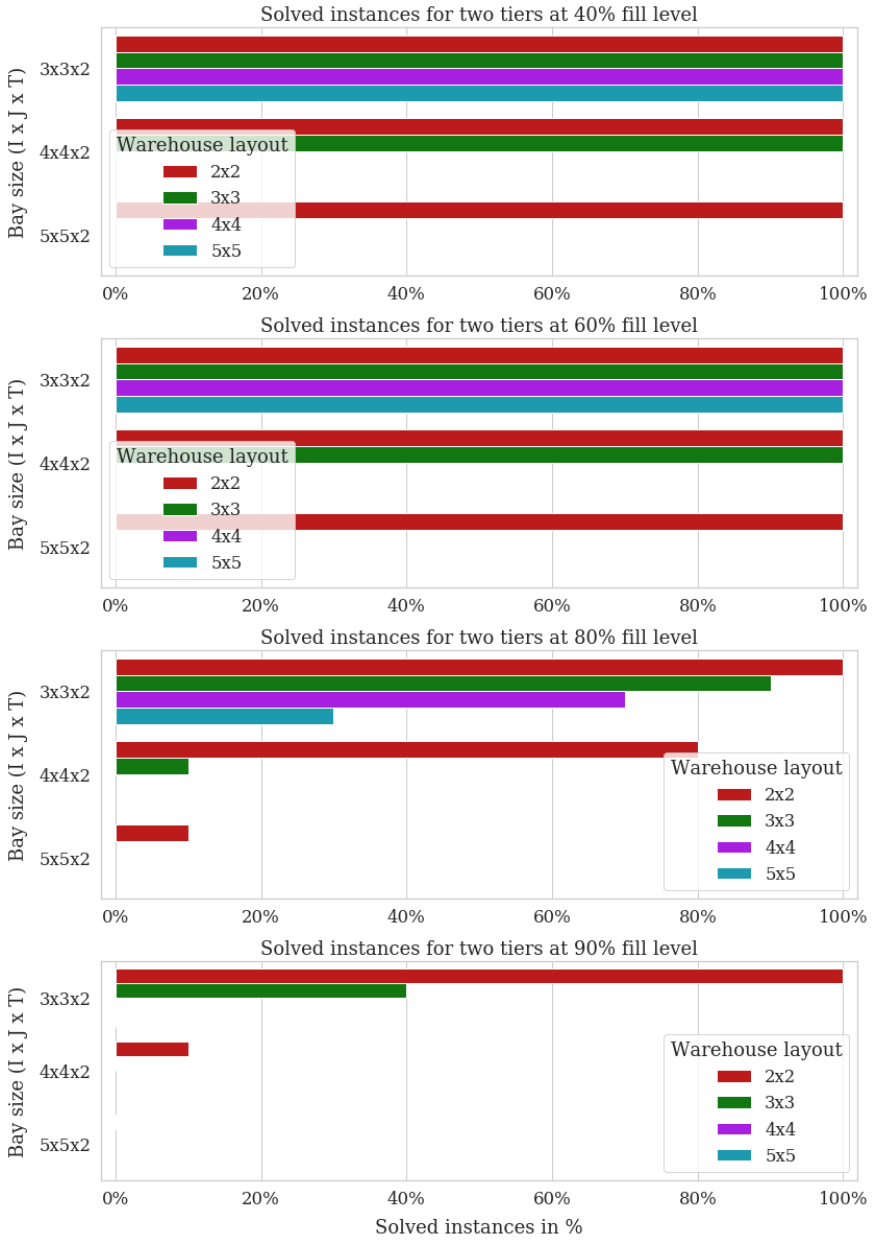


Fig. 5.33 Overview of solved instances for the multibay scenario with two tiers for each fill level over the bay and layout sizes.

Even though only a few solutions could be found for bay sizes 4x4x2 and 5x5x2, the runtimes presented in Fig. 5.34 are rather low. Also, the warehouse layout sizes of 3x3x2 that were solved have low runtimes in comparison to the results of experiment *A*, where all instances have been solved (see Fig. 5.14). An explanation may be that the runtime for two tiers has a higher number of outliers. However, another possibility is that in some of these cases, the lower-bound heuristic is not as strong. A definite answer needs further investigation with longer runtimes in order to ensure that potential instances with weaker lower bounds are solved.

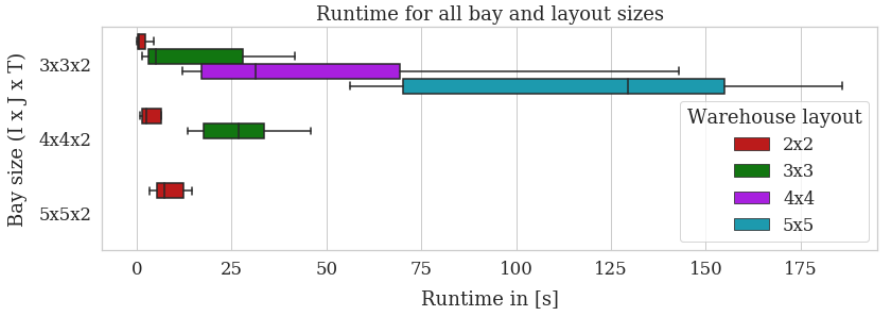


Fig. 5.34 Box plots (without outliers) of the runtime in seconds per bay size for each layout size.

*C*2: Number of moves, total move distance and total move processing time

When looking at the number of moves in Fig. 5.35, it becomes clear why there are rather many unsolved instances. The overall mean number of moves is 19.54, which is almost three times higher than for a single tier, where we solve warehouses with much larger capacities. Similar to a single tier, the number of moves increases with bay and warehouse layout size. We were able to solve instances with a maximum number of up to 70 moves.

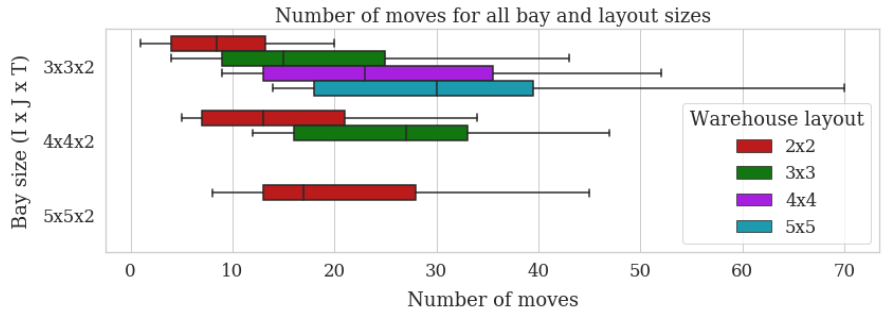


Fig. 5.35 Box plots (without outliers) of the number of moves per bay size for each layout size.

Also, the total move distance increases over bay and layout size (see Fig. 5.36). Even though the layout sizes are quite small, the travel distance increases to 611.8m and has a mean of 104.11m, which is more than twice as high as the mean for a single tier (which also contains instances with larger warehouse perimeter). Due to the high number of moves and the accompanying long total move distances, the total move processing times increase to a maximum of 77.28min and a mean of 20.78min, compared to 7.04min for a single tier (more details in Fig. A.34).

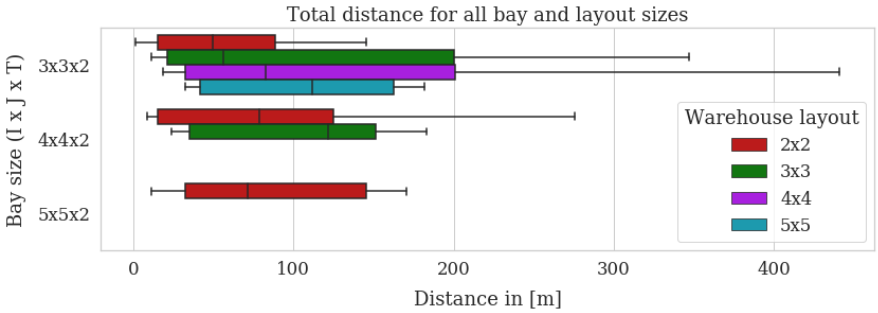


Fig. 5.36 Box plots (without outliers) of the total move distance in meters per number of moves.

*C*3**: Lower and upper bound**

The results for the lower and upper bounds lead to similar findings as in the case of a single tier. The gap between the lower bound in the root node and the resulting final moves is again 0% for all 194 solved instances. An upper bound was found for 176 instances. The detailed number of upper bounds found for each bay and warehouse layout size is provided in Fig. A.35. Only for bay size 3x3x2 at a warehouse size of 2x2 was an upper bound determined for all instances. The distribution of the final number of moves for instances with and without upper bound in Fig. 5.37 shows that an upper bound was not found for instances with more than 30 moves.

*C*4**: Move sequences**

For the experiments with two tiers, the derived numbers of independent move sequences are between one and 47. Whereas the minimum and maximum are quite similar to those of a single tier, the mean number of sequences is more than twice as high, at 13.55. The box plots for the number of independent sequences for all bay sizes and access directions is shown in Fig. 5.38. The number of independent move sequences increases much faster with each bay and warehouse layout size than for a single tier.

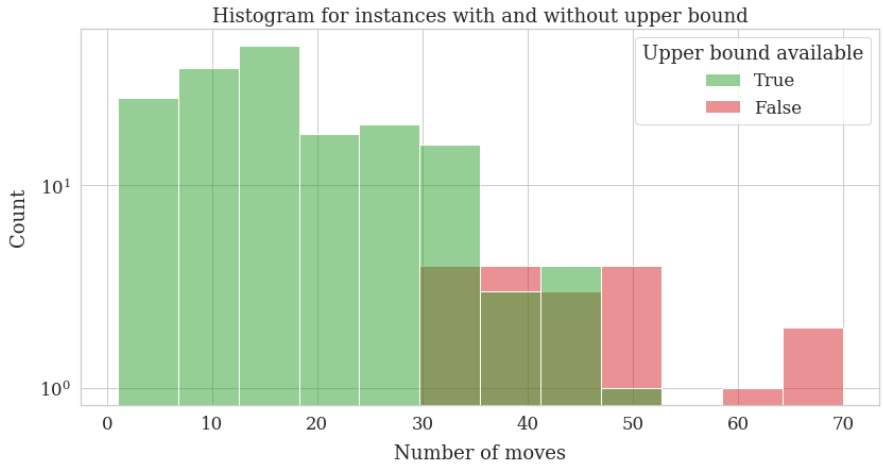


Fig. 5.37 Histogram of instances where upper bounds could and could not be found, over the number of moves for experiment *B*.

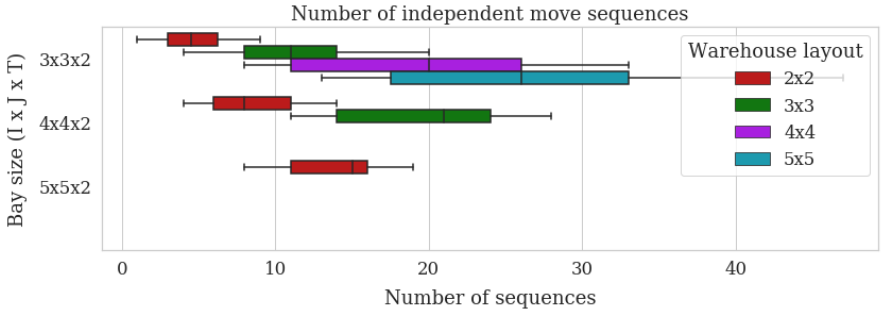


Fig. 5.38 Box plots (without outliers) of the number of independent sequences for all bay and warehouse layout sizes.

The overall longest sequence contains 26 dependent moves, compared to 19 moves for a single tier. The mean longest move sequence for all instances increases from 0.98 to 3.79. A reason could be that the additional tier increases the average depth of the virtual lanes. Deeper lanes most likely also cause more dependent move sequences. An overview with box plots of the longest move sequence for each bay and warehouse layout size is provided in Fig. A.36.

5.5 Business Insights

In this section, we discuss the main insights from our computational experiments on the pre-marshalling of multibay warehouses with up to four access directions. We already presented the benefits of the different access directions for a single bay in Section 4.5.2. These findings can also be confirmed for multibay warehouses: Practitioners should make use of additional access directions whenever possible. The sorting effort can be tremendously reduced when additional access directions are available. Similar to the previous result section, we split the discussion into three: dimensions (1) Efficiency and feasibility, (2) Layout decisions, and (3) Operational control.

Efficiency and feasibility: A great advantage of multibay warehouses is that they allow warehouse capacity to be scaled by changing the number of bays while keeping the same bay size. This way, we gain the benefit of fast access times for smaller sized bays and are able to increase the layout as much as desired. Increasing the layout size had a smaller impact on the additional sorting effort, compared to increasing the bay size. Overall, the amount of sorting effort in terms of required moves, travel distance, and processing time was surprisingly low for all randomly generated instances with four access directions—even for large warehouses and high fill levels.

Another benefit of multibay warehouses is that the fill level can be increased much more—to 90%, and potentially more—compared to a single bay, without having any infeasible instances. Infeasible instances occur mainly in situations where we do not have enough space to clear the space in front and then to subsequently move a unit load. The number of free storage locations required to solve a single bay is not increasing for a multitude of these bays. Thus, the fill level can be raised with an increasing number of bays. However, the distances may increase due to the limited number of move options.

Finally, similar to the experiments for the single bay UPMP, two tiers are not working very well. The biggest issue is the extremely high sorting effort compared to a single tier with the same warehouse capacity. In our randomly generated instances, there is a high probability of unsorted stacks. All unit loads in front of the same lane need to be moved. A potential solution to still enable stacking capability may be to use only shallow lanes with directly accessible stacks or to keep the stacks pure by not mixing SKUs within them.

Layout decisions: Determining the right aisle configuration in terms of number, shape, and size of bays, as well as the available access directions and appropriate aisle space, must be addressed in layout decisions. Although layout decisions are not part of this work, our findings may nonetheless prove interesting and useful for planners and in future research.

Our results confirm that the access directions of the bays play a key role in both improving access times and reducing sorting effort. In many warehouse

setups, storage bays are limited to a single access direction, even though two or more access directions could be instantly realized.

Furthermore, our experiments show that increasing warehouse capacity by increasing layout size does not increase sorting effort by as much as increasing the bay size does (e.g., bay size 3x3x1 with a 6x6 layout has a much lower sorting effort than a bay size of 6x6x1 in a 3x3 layout). However, there is a trade-off in terms of space utilization and access times/sorting effort: A large number of small bays also requires more aisle space for traveling, compared to larger bay sizes. Due to the additional aisle space needed for smaller-sized bays, the footprint of a warehouse with smaller bays is greater than that of a warehouse with larger bays. Ultimately, finding the right compromise of fast access times and a good space utilization must be individually assessed for each use case.

In these use cases, several factors—such as the external layout or the product mix, but also the respective performance and capacity requirements (e.g., fast access times versus high storage density)—play a key role in finding the right configuration. In setups where a high number of priority groups are required, smaller-sized bays can handle this high variety better than larger ones. More specifically, when comparing different numbers of priority groups, the gap in the number of moves widens with each bay size increment.

Operational control: In terms of operational control for robot fleets, we see some notable differences related to the size of the bays. In dense warehouse configurations, conflict-free routing is a necessity. The shortest path to the next target location may be blocked by another AMR. In layouts with many small bays, there are more alternative options for path-finding than in a warehouse with large bays. Increasing aisle width would also solve the problem; however, more aisle space does not improve unit load accessibility and does not shorten travel distances.

A positive effect of small bays is that the number of dependent move sequences is very low. This means that a parallelization to perform pre-marshalling is easily possible. For an increase in bay size, dependent move sequences become longer and longer.

5.6 Conclusion

After introducing our novel two-step approach for the single bay UPMP in Chapter 4, we show that the solution approach is also applicable to multibay warehouses with a much larger action and solutions space, which extends the scope to a greater number of real use cases. Our adapted and improved implementation of the algorithm solves the multibay UPMP with four access directions in real-world-sized warehouses containing up to 1,089 storage locations, with no timeouts within a limit of 10 minutes. In addition to minimizing

the number of moves needed to sort a warehouse, we also incorporate—and tremendously reduce—the total move distance.

The extensive experiments provide valuable insights for the configuration of dense, multibay block stacking warehouses. We show that multiple access directions are highly beneficial in multibay warehouses because they significantly reduce sorting effort in terms of the number of moves, travel distance, and processing time. The lower-bound heuristics proved highly accurate throughout our experiments, with an overall root node gap of 0%. Scaling warehouse capacity by adapting the layout size has a much smaller impact on the sorting effort than adapting the bay size. Retaining a bay size also preserves benefits such as access times or the number of independent move sequences. Although for the single bay UPMP, a fill level of 80% can lead to infeasible instances, we do not encounter any infeasible instances at a 90% fill level for the MUPMP. The fill level can be raised further with increasing warehouse size.

Thus far, in our artificially generated instances, unit loads are placed in the warehouse completely at random. Data from real use cases should be obtained to validate our solution approach and to gain further insights. An interesting route of future research would be to use the lower-bound heuristics for other decision problems (e.g., storage location assignment decisions). A AMR control system would require the investigation of all decision problems presented in Chapter 3. If the warehouse is continuously maintained in a better-sorted state, the required number of moves will be even lower. Another feature we have not yet addressed is the mix of different bay sizes and bay shapes within the same warehouse. This may offer possibilities such as fitting bay shape and size to a certain product mix, or more complex external layouts.

Furthermore, it would be interesting to benchmark our search procedure with other exact-solution approaches in terms of the total move distance. Currently, we find optimal results for the minimization of the number of moves, and distance is used only as a tie-breaking criterion. Hence, we cannot guarantee optimal results for the total move distance. Also, regarding the strong interaction with vehicle dispatching, which is not considered, and the total travel distance (including empty travel), further research is necessary. Finally, the application of efficient heuristic approaches (e.g., [HTT20]) and robust solution approaches for the stochastic MUPMP (e.g., [TV16]) can also increase the applicability in a wide variety of use cases.

Chapter 6
Conclusion

The concluding chapter is divided into three parts. First, we provide an extended summary of this work and recap the main contributions. Second, we discuss the implications for practice and supply chain networks. Third, we describe our study's limitations and offer an outlook on future research topics.

6.1 Summary

A key element of agile supply-chain networks is the ability to adapt storage space. The most flexible type of warehouses with the least infrastructure are block stacking storage systems. Block storage can be found in a large number of industries, such as in the production of FMCG, logistics services, maritime transport, and commerce.

Although large quantities of AMRs are already used to transport goods in factories, this is scarcely the case for the organization of block stacking warehouses: The problem-solving skills and flexibility of human operators in large, complex environments are hard to match. This thesis contributes to enabling autonomous block stacking storage systems by:

- presenting a vision for autonomous block stacking warehouses.
- introducing the most important decision problems that need to be considered as the aggregated ABSWP.
- formally defining the UPMP and MUPMP to sort a block stacking storage system during off-peak hours.
- developing a novel two-step solution approach based on a network flow model and an adapted tree search procedure.
- extending the solution approach to multibay block stacking storage systems.
- provisioning extensive experiments, including publicly available benchmark datasets.

In Chapter 2, we start by describing the fundamentals of block stacking warehouses, including the types of vehicles and unit loads, as well as the process flow and exemplary areas of application, including four examples of real use cases in the beverage, hygienic paper, and manufacturing industries. The four use cases show that the practical requirements for each industry and business are highly individualized. One example is product characteristics: Some goods, like beverage crates, are quite standardized and easily stackable. Others, like soft hygienic paper products, require a careful assessment: *Does a pallet have overhanging products or a large gap? Is the top surface flat? How much is the load-bearing capacity?*

In addition to individual requirements, there may also be differences in the prioritization of optimization objectives. In urban areas with limited space and high real estate prices, storage space utilization may be more important than in rural areas outside of big cities.

This is followed by an overview of different definitions of autonomous systems in the engineering and industrial contexts, wherein we present exemplary advantageous characteristics of automated and nonautomated systems. Today's automated systems are especially hampered by their lack of such advantageous characteristics of human operators as adaptability, creativity, curiosity, self-governance, and learning ability. In our vision of autonomous block stacking warehouses, the advantageous characteristics of both automated and manually operated warehouses are brought together to better meet the desired properties of storage systems. To illustrate such a warehouse, we describe key benefits and how they can be achieved (e.g., a lean setup process and minimum interfaces to the outside world).

Chapter 3 dealt with the specification of the operational decision problems and the elaboration of associated literature. Based on our findings, we described the major challenges that must be solved to achieve autonomously organized block stacking storage systems as well as the opportunities that arise from them.

Because the decision problems considered are highly interrelated, we introduced the integrated ABSWP, which includes the most relevant problems. Given the broad field of research and the complexity of all decision problems, developing solutions for the ABSWP is an ambitious goal. With the introduction of our novel pre-marshalling solution approaches in this thesis, as well as the development of our open-source simulation framework SLAPstack [Pfr+22], we contribute the first important components to long-term research endeavors in that direction. One of the greatest opportunities in autonomous storage systems is that a storage system is rarely utilized continuously over the course of a day. Thus, there is usually some idle time of the AMRs (e.g., off-peak hours during the night or on weekends) during which they could rearrange the unit loads and prepare the storage system for a future demand sequence. In our simulation study based on three months of data from a real

use case, the AMRs are in use approximately 25% of the time and idle for the remaining 75% [Pfr+22].

The problem of sorting block stacking storage systems based on their retrieval priority group in idle time (i.e., the UPMP) is the main subject in the following. The main advantage of these sorting operations is that they allow us to apply fully shared storage policies in highly dense storage setups without the downside of accessibility problems due to blockage that prevents reaching requested goods.

In Chapter 4, we formally introduced the single-bay UPMP for sorting a block stacking storage system with a minimum number of relocation moves until all blockage is resolved. The special feature for bay access in block stacking storage systems is that bays can be accessed from any combination of up to four directions. The available access directions depend on the surrounding environment (i.e., aisle space or walls). For the simplest case with only one access direction, we showed that a search-based algorithm from the CPMP can be directly transferred to the UPMP. For any other combination of multiple access directions, we presented a novel two-step solution approach. First, a network flow model was solved to determine the access direction for each stack and fix these directions throughout the pre-marshalling operation. Second, we extended lower-bound heuristics with a MIP model to estimate the required number of moves and ran an adapted tree search procedure. In computational experiments based on randomly generated instances, we solved large bay sizes with up to 81 storage locations (bay size 9x9x1) and demonstrated the usefulness of different combinations of multiple access directions. Our results show that a higher number of access directions leads to a significant reduction of the required number of moves, far more solved instances and, ultimately, faster runtimes as well.

Chapter 5 is a continuation of the single-bay UPMP extending the scope to a multibay setup. The MUPMP requires also taking into account the move distance, because warehouses with multiple bays have much larger footprints. The main extensions are as follows: (1) an additional pre-processing step with a new representation of the warehouse that includes multiple bays and designated access points from the available access directions, as well as the calculation of all distances between these access points; (2) improved selection of one result from the network flow model via the more advanced lower bound heuristics; (3) incremental calculation of the lower-bound heuristics to speed up the search procedure; (4) new tie-breaking rules that also incorporate the move distance to pick a node from the heap queue; (5) an additional upper bound based on greedy search that may reduce memory consumption; and (7) an algorithm to partition the resulting move sequence in as many independent sequences as possible to simplify the handling of pre-marshalling operations with multiple AMRs. The adapted and improved implementation of our solution approach solves real-world-sized random in-

stances with up to 1,089 storage locations and 90% fill level with no timeouts. Although we determined solutions with the minimum number of moves, we were also able to reduce the move distance by 81.65% (compared to the solution approach that minimizes only the number of moves). Our extensive experiments showed that multiple bays allow fill levels to be increased while still being able to solve all instances. A detailed analysis and comparison of the access direction combinations, tiers, priority groups, as well as bay and layout sizes, provided insights for planners and practitioners. Overall, we hope that this work lays the foundation and will drive real applications as well as further research related to block stacking storage systems.

6.2 Implications for supply-chain networks

Pre-marshalling and rearrangement operations allow highly dense block stacking warehouses to operate under a fully shared storage policy. Because the preparation of the warehouse requires future retrieval times of the unit loads, information sharing within a supply network is crucial. However, even if the information is available, there may be endogenous and exogenous uncertainty such as market volatility, technological changes, daily traffic jams or events like the recent COVID-19 pandemic and the war in Ukraine. The following example describes how companies are affected by the dynamic environment.

Fig. 6.1 shows an exemplary supply network in the retail industry. A producer of FMCG at locations A, B, and C delivers to the distribution centers of a retailer. From the distribution centers, the products are delivered to different shops where consumers can purchase them. Periods of strong sales (e.g., at Shop C) can spur the need for additional warehouse space in the short term. In these situations, goods are often transported to other locations or external warehouses. Likewise, it may be necessary to respond to geographical shifts in the logistics network over time (e.g., new supplier) and relocate warehouses to more geographically suitable locations. Volatile production and trading environments ensure that ideal locations exist only briefly.

Block stacking storage systems offer clear advantages in adapting to any upcoming influences. They allow warehouse capacity to be scaled at short notice and today are particularly relevant in practice for meeting temporary needs of additional storage capacity. We believe that autonomous block stacking storage systems, which are simple to set up and operate, can significantly improve factories' and supply-chain networks' adaptability to any kind of uncertainty. Autonomously organized block storage bases its decisions on data and real-time information, which will lead to further improvements of efficiency, reliability, and visibility. Moreover, autonomous block stacking systems are less labor-intensive, but still require logistics personnel for ar-

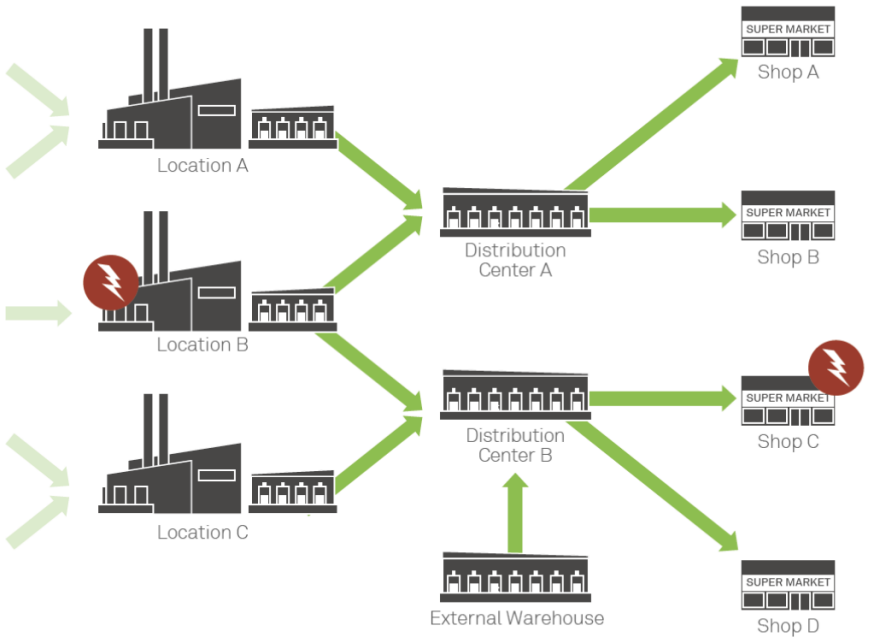


Fig. 6.1 Example of a supply network in the retail industry.

eas such as maintenance, exception and failure handling, reception, ramp management, and security. We believe that an autonomous system based on AMRs will help to scale, shift, and build up storage space precisely where it is needed in a dynamic supply network. Ultimately this will reduce waste, such as by avoiding unnecessary detours and establishing more resource-efficient routes.

We also want to highlight the concept of shared warehouses, where several companies operate a storage space together. Autonomous systems provide the transparency as well as billing and accounting possibilities that are needed to realize such a concept. Autonomous block stacking warehouses appear especially suitable for shared warehouse concepts, because companies would not need to commit large investments into dependencies with other companies.

6.3 Limitations and future research

Although we were able to solve quite large real-world-sized instances, there very well may be bigger warehouse scenarios that also usually involve stochastic events. A logical next step would be to test further heuristics and robust approaches that are still able to find good solutions. We may also question whether real scenarios are as difficult to solve as the randomly

generated instances. Real-world warehouses do not often become so messy, because blockage is already prevented when making storage location assignment decisions. To determine the sorting effort in realistic warehouse scenarios, two things are required: (1) a real-world dataset where retrieval times of the unit loads are available or can be derived and (2) a simulation model that integrates the decision problems and permits analysis of the different solution strategies' dynamics. We already introduced an open-source simulation framework for block stacking warehouses in [Pfr+22] as well as a large-scale benchmark dataset. However, the simulation model was initially tailored to solve the SLAP, and further work is needed to extend the simulation model with rearrangement and pre-marshalling decisions. Furthermore, the use case in [Pfr+22] does not fit, due to the huge warehouse size, the small variety of products, and the limited amount of reliable retrieval information. A fully shared storage strategy makes little sense because there is only a small amount of SKUs in a very large warehouse, and thus SKUs are stored in dedicated lanes. An interesting area of research would be to create feasible variants of simulation models that fit the requirements of each specific use case by employing automatic simulation model generation via component-based synthesis as shown by our work in [Kal+21].

In addition to the endeavors of providing a capable simulation framework, we also deployed our pre-marshalling algorithm at a buffer storage for the material supply of a matrix assembly system ("cyber-physical production system") at the Innovation Center of the Chair for Material Handling Systems and Warehousing (FLW) at TU Dortmund. The buffer storage (see purple laser in the bottom image of Fig. 6.2) applies our lower-bound heuristics for storage location assignment decisions as well, and demonstrates functionality at a small scale. Bringing a similar concept into a real application would require solving a raft of additional problems, such as other shapes and restrictions of external layouts (e.g., pillars), different types of unit loads, weight and height restrictions, and handling of dangerous goods or allergens.

Our approach for the MUPMP finds solutions with a minimum number of moves, while the move distance is minimized through tie-breaking decisions. It would be interesting to compare the results with an exact-solution approach for the total travel distance. Also for the total processing time, more accurate kinematic modeling of the acceleration and speed of AMRs, with and without carrying a load, may be taken into account. The total move distance in our implementation considers only the loaded travel distance for each move. A more precise approach would be to also include the unloaded travel times and the exact positions of AMRs.

Using the network flow model to determine the access directions and fix them throughout the pre-marshalling process works very well and simplifies traffic management within a storage bay. Nevertheless, fixing the access directions may lead to solutions that require more moves than a solution

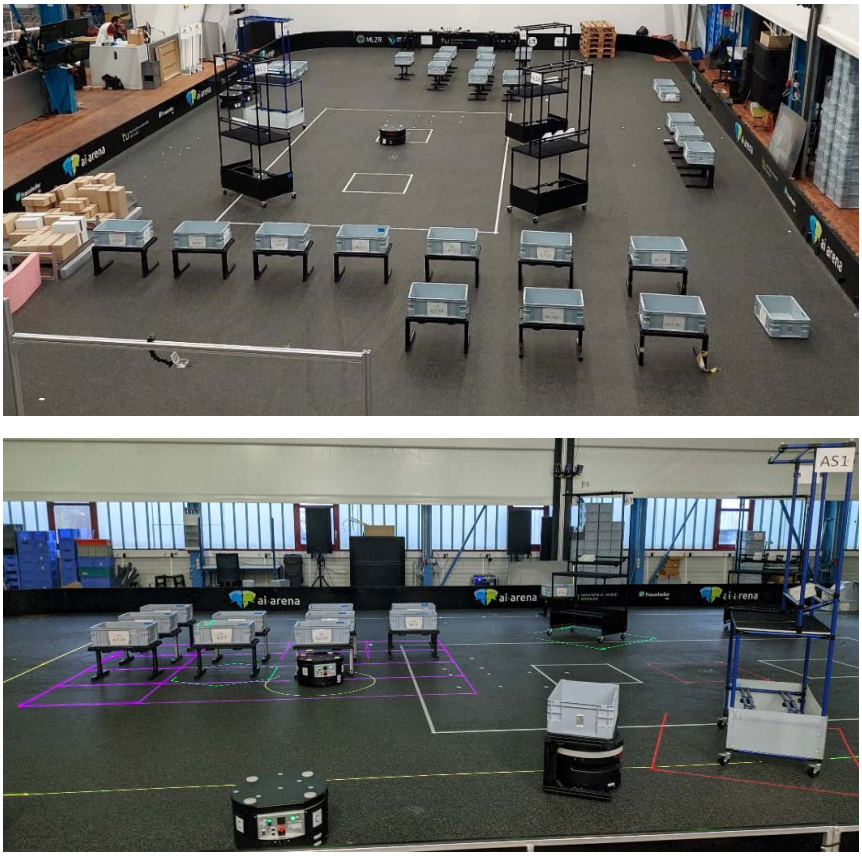


Fig. 6.2 Images of the cyber-physical production system with grid-based block storage at the Innovation Center of the Chair of Material Handling Systems and Warehousing (FLW), TU Dortmund University.

where the access directions are able to cross over a stack and change over time (see Section 4.4.3).

Another interesting direction of research concerns the internal layout. Our results show that multiple access directions are beneficial because they reduce the number of moves needed to sort a storage system and improve the accessibility of unit loads. However, additional aisle space may be required to permit accessing a bay from multiple directions. A second factor related to this trade-off between storage space utilization and accessibility is bay size. Warehouses with fewer large bays have an improved overall space utilization compared to warehouses with more small bays because they require less aisle space. Yet again, there is the drawback of increased sorting effort and blockage due to deeper lanes. The design of good layout configurations, considering a mix of bay shapes and sizes together with access directions,

needs still to be investigated. As shown in Section 3.2, most research oversimplifies warehouse operations (e.g., considering SC cycles) and does not address different access directions. At the same time, we also ask ourselves whether an aisle configuration is really necessary, or whether a layout should evolve incrementally with each storage location assignment decision. Our lower-bound heuristics may be an essential ingredient in assessing possible storage locations in regard to blockage.

Finally, it also remains to be determined when a relocation or pre-marshalling process is executed. In most cases, there are obvious off-peak hours throughout a day that can be utilized for pre-marshalling. However, they may also be situations where only a limited amount of time, or only a part of the AMRs, is available. It must be decided at what point in time, and to which extent, a rearrangement process is triggered and eventually stopped.

Bibliography

[ABD13] Eelco van Asperen, Bram Borgman, and Rommert Dekker. "Evaluating impact of truck announcements on container stacking efficiency". In: *Flexible Services and Manufacturing Journal* 25.4 (2013), pp. 543–556.

[ABM17] Riccardo Accorsi, Giulia Baruffaldi, and Riccardo Manzini. "Design and manage deep lane storage system layout. An iterative decision-support model". In: *The International Journal of Advanced Manufacturing Technology* 92.1-4 (2017), pp. 57–67.

[Alb+22] Thomas Albrecht, Jochen Luz, Waldemar Osterhoff, Kai Pfeiffer, Karl Rapp, and Günter Ullrich. *AGV Guide - Autonomy of Mobile Robots*. 2022.

[Bal93] WW Rouse Ball. "Mathematical Recreations and Essays". In: *Bulletin des sciences mathématiques* 17 (1893), pp. 105–107.

[BE16] Nils Boysen and Simon Emde. "The parallel stack loading problem to minimize blockages". In: *European Journal of Operational Research* 249.2 (2016), pp. 618–627.

[Ber68] J. R. Berry. "Elements of warehouse layout". In: *The International Journal of Production Research* 7.2 (1968), pp. 105–121.

[BF12] Andreas Bortfeldt and Florian Forster. "A tree search procedure for the container pre-marshalling problem". en. In: *European Journal of Operational Research* 217.3 (2012), pp. 531–540.

[BH19] John J Bartholdi and Steven T Hackman. *Warehouse & Distribution Science: Release 0.98*. Supply Chain and Logistics Institute Atlanta, 2019.

[Böv+19] Hanna Böving, Elise Glaß, Erik Haberzeth, and Susanne Umbach. "Digitalisierte Arbeit und menschliche Initiative. Empirische Analysen aus Logistik und Einzelhandel". In: *Bildung 2.1 für Arbeit 4.0?* Springer, 2019, pp. 141–160.

[Bun18] Bundesvereinigung Logistik. *BVL-Umfragen - Die BVL: Das Logistik-Netzwerk für Fach- und Führungskräfte — bvl.de*. https://www.bvl.de/service/zahlen-daten-fakten/umfrage/ergebnisse-fachkraeftemangel. [Accessed 2023-01-13, 15:01]. 2018.

[BV04] Russell W Bent and Pascal Van Hentenryck. "Scenario-based planning for partially dynamic vehicle routing with stochastic customers". In: *Operations Research* 52.6 (2004), pp. 977–987.

[BV07] Russell Bent and Pascal Van Hentenryck. "Waiting and Relocation Strategies in Online Stochastic Vehicle Routing." In: *IJCAI*. Vol. 7. 2007, pp. 1816–1821.

[CC73] Nicos Christofides and I. Colloff. "The Rearrangement of Items in a Warehouse". In: *Operations Research* 21.2 (1973), pp. 577–589.

[CCR18] Thomas Chabot, Leandro Coelho, and Jacques Renaud. "Mathematical Models for the Warehouse Reassignment Problem". In: *CIRRELT working paper* (2018).

[CG12] Héctor J Carlo and Germán E Giraldo. "Toward perpetually organized unit-load warehouses". In: *Computers & Industrial Engineering* 63.4 (2012), pp. 1003–1012.

[Che+21] Hualong Chen, Yuanqiao Wen, Man Zhu, Yamin Huang, Changshi Xiao, Tao Wei, and Axel Hahn. "From automation system to autonomous system: An architecture perspective". In: *Journal of Marine Science and Engineering* 9.6 (2021), p. 645.

[Cho+11] Ri Choe, Taejin Park, Myung-Seob Oh, Jaeho Kang, and Kwang Ryel Ryu. "Generating a rehandling-free intra-block remarshaling plan for an automated container yard". In: *Journal of Intelligent Manufacturing* 22.2 (2011), pp. 201–217.

[Cho+15] Ri Choe, Tae Sung Kim, Taekwang Kim, and Kwang Ryel Ryu. "Crane scheduling for opportunistic remarshaling of containers in an automated stacking yard". In: *Flexible Services and Manufacturing Journal* 27.2 (2015), pp. 331–349.

[Cov17] Filip Covic. "Re-marshalling in automated container yards with terminal appointment systems". In: *Flexible Services and Manufacturing Journal* 29.3 (2017), pp. 433–503.

[CSV11] Marco Caserta, Silvia Schwarze, and Stefan Voß. "Container Rehandling at Maritime Container Terminals". In: *Handbook of Terminal Planning*. Operations Research/Computer Science Interfaces Series. 2011, pp. 247–269.

[CSV12] Marco Caserta, Silvia Schwarze, and Stefan Voß. "A mathematical formulation and complexity considerations for the blocks relocation problem". In: *European Journal of Operational Research* 219.1 (2012), pp. 96–104.

[Der17] Shahab Derhami. "Designing Optimal Layouts for Block Stacking Warehouses". Auburn University, 2017.

[Des+03] Guy Desaulniers, André Langevin, Diane Riopel, and Bryan Villeneuve. "Dispatching and conflict-free routing of automated guided vehicles: An exact approach". In: *International Journal of Flexible Manufacturing Systems* 15.4 (2003), pp. 309–331.

[Dij+59] Edsger W Dijkstra et al. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[DSG16] Shahab Derhami, Jeffrey S Smith, and Kevin R Gue. "A simulation model to evaluate the layout for block stacking warehouses". In: *14TH IMHRC Proceedings*. Progress in Material Handling Research (2016).

[DSG17] Shahab Derhami, Jeffrey S Smith, and Kevin R Gue. "Optimising space utilisation in block stacking warehouses". In: *International Journal of Production Research* 55.21 (2017), pp. 6436–6452.

[DSG19] Shahab Derhami, Jeffrey S Smith, and Kevin R Gue. "Space-efficient layouts for block stacking warehouses". In: *IISE Transactions* 51.9 (2019), pp. 957–971.

[DSG20] Shahab Derhami, Jeffrey S Smith, and Kevin R Gue. "A simulation-based optimization approach to design optimal layouts for block stacking warehouses". In: *International Journal of Production Economics* 223 (2020), p. 107525.

[DVD20] Matthias De Ryck, Mark Versteyhe, and Frederik Debrouwere. "Automated guided vehicle systems, state-of-the-art control algorithms and techniques". In: *Journal of Manufacturing Systems* 54 (2020), pp. 152–173.

[EMM12] Christopher Expósito-Izquierdo, Belén Melián-Batista, and Marcos Moreno-Vega. "Pre-Marshalling Problem: Heuristic solution method and instances generator". In: *Expert Systems with Applications* 39.9 (2012), pp. 8337–8349.

[ET84] Pius J Egbelu and Jose MA Tanchoco. "Characterization of automatic guided vehicle dispatching rules". In: *The International Journal of Production Research* 22.3 (1984), pp. 359–374.

[EW93] Pius J Egbelu and CT Wu. "A comparison of dwell point rules in an automated storage/retrieval system". In: *The International Journal of Production Research* 31.11 (1993), pp. 2515–2530.

[FK19] Kai Furmans and Christoph Kilger. *Betrieb von Logistiksystemen*. Springer, 2019.

[Fot+21] Johannes Fottner, Dana Clauer, Fabian Hormes, Michael Freitag, Thies Beinke, Ludger Overmeyer, S Gottwald, Ralf Elbert, Tessa Sarnow, Thorsten Schmidt, et al. "Autonomous systems in intralogistics–state of the art and future research challenges". In: *Journal LOGISTICS RESEARCH* (2021).

[Fra+21] Giuseppe Fragapane, René de Koster, Fabio Sgarbossa, and Jan Ola Strandhagen. "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda". In: *European Journal of Operational Research* (2021).

[Fra16] Edward H Frazelle. *World-class warehousing and material handling*. McGraw-Hill Education, 2016.

[GD91] Marc Goetschalckx and H Donald Ratldff. "Optimal lane depths for single and multiple products in block stacking storage systems". In: *Iie Transactions* 23.3 (1991), pp. 245–258.

[Ge+20] Peixin Ge, Ying Meng, Jiyin Liu, Lixin Tang, and Ren Zhao. "Logistics optimisation of slab pre-marshalling problem in steel industry". In: *International Journal of Production Research* (2020), pp. 4050–4070.

[Ge+21] Peixin Ge, Ren Zhao, Defeng Sun, and Yun Dong. "Integrated optimisation of storage and pre-marshalling moves in a slab warehouse". In: *International Journal of Production Research* (2021), pp. 1–23.

[GF08] Harald Gleißner and J Christian Femerling. *Bestands-, Bevorratungs-und Bereitstellungsmanagement*. Springer, 2008, pp. 139–164.

[GGM07] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. "Research on warehouse operation: A comprehensive review". In: *European journal of operational research* 177.1 (2007), pp. 1–21.

[GGM10] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. "Research on warehouse design and performance evaluation: A comprehensive review". In: *European journal of operational research* 203.3 (2010), pp. 539–549.

[GIM12] Kevin R Gue, Goran Ivanović, and Russell D Meller. "A unit-load warehouse with multiple pickup and deposit points and non-traditional aisles". In: *Transportation Research Part E: Logistics and Transportation Review* 48.4 (2012), pp. 795–806.

[GK07] Kevin R. Gue and Byung Soo Kim. "Puzzle-based storage systems". In: *Naval Research Logistics (NRL)* 54.5 (2007), pp. 556–567.

[GM09] Kevin R Gue and Russell D Meller. "Aisle configurations for unit-load warehouses". In: *IIE transactions* 41.3 (2009), pp. 171–182.

[GN92] N. Gupta and D. Nau. "On the Complexity of Blocks-World Planning". In: *Artif. Intell.* (1992).

[GR90] Marc Goetschalckx and H Donald Ratliff. "Shared storage policies based on the duration stay of unit loads". In: *Management science* 36.9 (1990), pp. 1120–1132.

[Han+87] Min-Hong Han, Leon F McGinnis, Jin Shen Shieh, and John A White. "On sequencing retrievals in an automated storage/retrieval system". In: *IIE transactions* 19.1 (1987), pp. 56–66.

[HE00] Chih-Hsiung Hu and Pius J Egbelu. "A framework for the selection of idle vehicle home locations in an automated guided vehicle system". In: *International Journal of Production Research* 38.3 (2000), pp. 543–562.

[Hes63] James L Heskett. "Cube-per-order index-a key to warehouse stock location". In: *Transportation and distribution Management* 3.1 (1963), pp. 27–31.

[HMA15] Christopher-Eyk Hrabia, Nils Masuch, and Sahin Albayrak. "A metrics framework for quantifying autonomy in complex systems". In: *Multiagent System Technologies: 13th German Conference, MATES 2015, Cottbus, Germany, September 28-30, 2015, Revised Selected Papers 13*. Springer. 2015, pp. 22–41.

[HNR68] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[Hön+16] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. "Multi-agent path finding with kinematic constraints". In: *Twenty-Sixth International Conference on Automated Planning and Scheduling*. 2016.

[Hor+21] Natália Horňáková, Lukáš Jurík, Henrieta Hrablik Chovanová, Dagmar Cagáňová, and Dagmar Babčanová. "AHP method application in selection of appropriate material handling equipment in selected industrial enterprise". In: *Wireless Networks* 27.3 (2021), pp. 1683–1691.

[HSG76] Warren H Hausman, Leroy B Schwarz, and Stephen C Graves. "Optimal storage assignment in automatic warehousing systems". In: *Management science* 22.6 (1976), pp. 629–638.

[HTT20] André Hottung, Shunji Tanaka, and Kevin Tierney. "Deep learning assisted heuristic tree search for the container pre-marshalling problem". en. In: *Computers & Operations Research* 113 (2020), p. 104781.

[IFO22] IFO Institut. *Fachkräftemangel steigt auf Allzeithoch — ifo.de*. https://www.ifo.de/pressemitteilung/2022-08-02/fachkraeftemangel-steigt-auf-allzeithoch. [Accessed 2023-01-13, 14:50]. 2022.

[Jim+23] Celia Jiménez-Piqueras, Rubén Ruiz, Consuelo Parreño-Torres, and Ramon Alvarez-Valdes. "A constraint programming approach for the premarshalling problem". In: *European Journal of Operational Research* 306.2 (2023), pp. 668–678.

[JYD20] Guang Jin, Peng Yang, and Guofang Duan. "Multiple Deep Layout of Robotic Mobile Fulfillment System". In: *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*. 2020, pp. 230–234.

[Kal+21] Fadil Kallat, Jakob Pfrommer, Jan Bessai, Jakob Rehof, and Anne Meyer. "Automatic Building of a Repository for Component-based Synthesis of Warehouse Simulation Models". In: *Procedia CIRP* 104 (2021), pp. 1440–1445.

[Kam+17] Mina Kamel, Javier Alonso-Mora, Roland Siegwart, and Juan Nieto. "Nonlinear model predictive control for multi-micro aerial vehicle robust collision avoidance". In: *arXiv preprint arXiv:1703.01164* (2017).

[Kan+06] Jaeho Kang, Myung-Seob Oh, Eun Yeong Ahn, Kwang Ryel Ryu, and Kap Hwan Kim. "Planning for intra-block remarshalling in a container terminal". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. 2006, pp. 1211–1220.

[KB98] Kap Hwan Kim and Jong Wook Bae. "Re-marshaling export containers in port container terminals". In: *Computers & Industrial Engineering* 35.3-4 (1998), pp. 655–658.

[Kem12] Nils Kemme. "Design and operation of automated container storage systems". University of Hamburg, 2012.

[KL02] Sven Koenig and Maxim Likhachev. "D^* lite". In: *Aaai/iaai* 15 (2002), pp. 476–483.

[KL76] Carl Kallina and Jeffrey Lynn. "Application of the cube-per-order index rule for stock location in a distribution warehouse". In: *Interfaces* 7.1 (1976), pp. 37–46.

[Kof+11] Monika Kofler, Andreas Beham, Stefan Wagner, Michael Affenzeller, and Werner Achleitner. "Re-warehousing vs. healing: Strategies for warehouse storage location assignment". In: *3rd IEEE International Symposium on Logistics and Industrial Informatics* (2011), pp. 77–82.

[Kön+07] Felix G. König, Macro Lübbecke, Rolf Möhring, Guido Schäfer, and Ines Spenke. "Solutions to Real-World Instances of PSPACE-Complete Stacking". In: *Algorithms – ESA 2007*. Ed. by Lars Arge, Michael Hoffmann, and Emo Welzl. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 729–740.

[KXL18] Ruslan K. Krenzler, Lin Xie, and Hanyi Li. "Deterministic Pod Repositioning Problem in Robotic Mobile Fulfillment Systems". In: *ArXiv* abs/1810.05514 (2018). arXiv: 1810.05514.

[LC09] Yusin Lee and Shih-Liang Chao. "A neighborhood search heuristic for premarshalling export containers". In: *European Journal of Operational Research* 196.2 (2009), pp. 468–475.

[LD06] Tuan Le-Anh and MBM De Koster. "A review of design and control of automated guided vehicle systems". In: *European Journal of Operational Research* 171.1 (2006), pp. 1–23.

[Lee+04] Hau L Lee et al. "The triple-A supply chain". In: *Harvard business review* 82.10 (2004), pp. 102–113.

[Lee+06] L.H. Lee, E.P. Chew, K.C. Tan, and Y. Han. "An optimization model for storage yard management in transshipment hubs". In: *OR Spectrum* 28.4 (2006), pp. 539–561.

[Lee+21] Sangmin Lee, Dae-Eun Lim, Younkook Kang, and Hae Joong Kim. "Clustered multi-task sequence-to-sequence learning for autonomous vehicle repositioning". In: *IEEE Access* 9 (2021), pp. 14504–14515.

[Lee19] Hueon Lee. *Optimizing Block-Stacking Operations with Relocation*. University of Arkansas, 2019.

[LH07] Yusin Lee and Nai-Yun Hsu. "An optimization model for the container pre-marshalling problem". In: *Computers & operations research* 34.11 (2007), pp. 3295–3313.

[Li+11] Bin Li, Daqing Zhang, Lin Sun, Chao Chen, Shijian Li, Guande Qi, and Qiang Yang. "Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset". In: *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE. 2011, pp. 63–68.

[LK14] Jana Lehnfeld and Sigrid Knust. "Loading, unloading and premarshalling of stacks in storage areas: Survey and classification". In: *European Journal of Operational Research* 239.2 (2014), pp. 297–312.

[LMK97] T Nick Larson, Heather March, and Andrew Kusiak. "A heuristic approach to warehouse layout with class-based storage". In: *IIE transactions* 29.4 (1997), pp. 337–348.

[LW90a] Richard J. Linn and Richard A. Wysk. "An expert system based controller for an automated storage/retrieval system". In: *International Journal of Production Research* 28.4 (1990).

[LW90b] Richard J. Linn and Richard A. Wysk. "An expert system framework for automated storage and retrieval system control". In: *Computers & Industrial Engineering* 18.1 (1990), pp. 37–48.

[Mar16] Heinrich Martin. *Transport-und Lagerlogistik: Systematik, Planung, Einsatz und Wirtschaftlichkeit*. Springer-Verlag, 2016.

[MBG21] Vittorio Maniezzo, Marco A. Boschetti, and Walter J. Gutjahr. "Stochastic premarshalling of block stacking warehouses". In: *Omega* 102 (2021), p. 102336.

[Mee00] Robert van der Meer. "Operational control of internal transport". Erasmus Research Institute of Management Rotterdam, 2000.

[Mee02] Patrick Johannes Maria Meersmans. "Optimization of container handling systems". Erasmus University Rotterdam, 2002.

[Mer18] Marius Merschformann. "Active Repositioning of Storage Units in Robotic Mobile Fulfillment Systems". In: *Operations Research Proceedings 2017*. Operations Research Proceedings. 2018, pp. 379–385.

[MGM19] Livia Maglić, Marko Gulić, and Lovro Maglić. "Optimization of container relocation operations in port container terminals". In: *Transport* (2019), pp. 1–11.

[MLP95] B. Muralidharan, Richard J. Linn, and Ram Pandit. "Shuffling heuristics for the storage location assignment in an AS/RS". In: *International Journal of Production Research* 33.6 (1995).

[Mül+21] Manuel Müller, Timo Müller, Behrang Ashtari Talkhestani, Philipp Marks, Nasser Jazdi, and Michael Weyrich. "Industrial autonomous systems: a survey on definitions, characteristics and abilities". In: *at-Automatisierungstechnik* 69.1 (2021), pp. 3–13.

[ÖGM14] Ömer Öztürkoğlu, Kevin R Gue, and Russell D Meller. "A constructive aisle design model for unit-load warehouses with multiple pickup and deposit points". In: *European Journal of Operational Research* 236.1 (2014), pp. 382–394.

[PAP22] Consuelo Parreño-Torres, Ramon Alvarez-Valdes, and Francisco Parreño. "A beam search algorithm for minimizing crane times in premarshalling problems". In: *European Journal of Operational Research* (2022).

[Par+20] Consuelo Parreño-Torres, Ramon Alvarez-Valdes, Rubén Ruiz, and Kevin Tierney. "Minimizing crane times in pre-marshalling problems". In: *Transportation Research Part E: Logistics and Transportation Review* 137 (2020), p. 101917.

[PAR19] Consuelo Parreño-Torres, Ramon Alvarez-Valdes, and Rubén Ruiz. "Integer programming models for the pre-marshalling problem". In: *European Journal of Operational Research* 274.1 (2019), pp. 142–154.

[PC15] Jennifer A. Pazour and Héctor J. Carlo. "Warehouse reshuffling: Insights and optimization". In: *Transportation Research Part E: Logistics and Transportation Review* 73 (2015), pp. 207–226.

[Pfr+22] Jakob Pfrommer, Alexandru Rinciog, Sohaib Zahid, Michael Morrissey, and Anne Meyer. "SLAPStack: A Simulation Framework and a Large-Scale Benchmark Use Case for Autonomous Block Stacking Warehouses". In: *International Conference on Computational Logistics*. 2022, pp. 291–305.

[Pil+13] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. "A review of dynamic vehicle routing problems". In: *European Journal of Operational Research* 225.1 (2013), pp. 1–11.

[PL11] Mike Phillips and Maxim Likhachev. "Sipp: Safe interval path planning for dynamic environments". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 5628–5635.

[Pla22] Martin Placek. *Revenue of the pallet market worldwide from 2019 to 2027*. https://www.statista.com/statistics/1228432/global-pallet-market-size/. Accessed: 2022-11-30, 09:54. 2022.

[PM20] Jakob Pfrommer and Anne Meyer. "Autonomously organized block stacking warehouses: A review of decision problems and major challenges". In: *Logistics Journal: Proceedings* 2020.12 (2020).

[PMT22] Jakob Pfrommer, Anne Meyer, and Kevin Tierney. "Solving the unit-load pre-marshalling problem in block stacking storage systems with multiple access directions". In: *arXiv preprint arXiv:2207.09118* (2022).

[PPR09] Kiyeok Park, Taejin Park, and Kwang Ryel Ryu. "Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms". In: *Proceedings of the 2009 ACM symposium on Applied Computing*. 2009, pp. 1098–1105.

[Pra13] Matthias Prandtstetter. "A dynamic programming based branch-and-bound algorithm for the container pre-marshalling problem". In: *Technical report, IT Austrian institute of technology* (2013).

[PS01] Rolf Pfeifer and Christian Scheier. *Understanding intelligence*. MIT press, 2001.

[Rim+21a] Adrien Rimélé, Michel Gamache, Michel Gendreau, Philippe Grangier, and Louis-Martin Rousseau. "Robotic mobile fulfillment systems: a mathematical modelling framework for e-commerce applications". In: *International Journal of Production Research* (2021), pp. 1–17.

[Rim+21b] Adrien Rimélé, Philippe Grangier, Michel Gamache, Michel Gendreau, and Louis-Martin Rousseau. "E-commerce warehousing: learning a storage policy". In: *arXiv preprint arXiv:2101.08828* (2021).

[Rim+21c] Adrien Rimélé, Philippe Grangier, Michel Gamache, Michel Gendreau, and Louis-Martin Rousseau. "Supervised learning and tree search for real-time storage allocation in Robotic Mobile Fulfillment Systems". In: *arXiv preprint arXiv:2106.02450* (2021).

[Rou+00] Bart Rouwenhorst, Boris Reuter, V Stockrahm, Geert-Jan van Houtum, RJ Mantel, and Willem HM Zijm. "Warehouse design and control: Framework and literature review". In: *European journal of operational research* 122.3 (2000), pp. 515–533.

[RP13] Andrea Rendl and Matthias Prandtstetter. *Constraint Models for the Container Pre-Marshaling Problem*. 2013.

[RPH16] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. "A survey on dynamic and stochastic vehicle routing problems". In: *International Journal of Production Research* 54.1 (2016), pp. 215–231.

[RSM19] J Reyes, E Solano-Charris, and J Montoya-Torres. "The storage location assignment problem: A literature review". In: *International Journal of Industrial Engineering Computations* 10.2 (2019), pp. 199–224.

[Sch14] Peter C Schuur. "The Cube per Order Index Slotting Strategy, how bad can it be?" In: *13TH IMHRC Proceedings*. Progress in Material Handling Research (2014).

[Sei88] Abraham Seidmann. "Intelligent control schemes for automated storage and retrieval systems". In: *The International Journal Of Production Research* 26.5 (1988), pp. 931–952.

[SF07] Bernd Scholz-Reiter and Michael Freitag. "Autonomous processes in assembly systems". In: *CIRP annals* 56.2 (2007), pp. 712–729.

[Sha+15] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[Sna+11] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. "The hybrid reciprocal velocity obstacle". In: *IEEE Transactions on Robotics* 27.4 (2011), pp. 696–706.

[Swa16] Lucas Swaine. “The origins of autonomy”. In: *History of Political Thought* 37.2 (2016), pp. 216–237.

[Tan+02] Lixin Tang, Jiyin Liu, Aiying Rong, and Zihou Yang. “Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules”. In: *International Journal of Production Research* 40.7 (2002), pp. 1583–1595.

[Tan+15] Lixin Tang, Jiyin Liu, Fei Yang, Feng Li, and Kun Li. “Modeling and solution for the ship stowage planning problem of coils in the steel industry”. In: *Naval Research Logistics (NRL)* 62.7 (2015), pp. 564–581.

[Tan+19] Shunji Tanaka, Kevin Tierney, Consuelo Parreño-Torres, Ramon Alvarez-Valdes, and Rubén Ruiz. “A branch and bound approach for large pre-marshalling problems”. In: *European Journal of Operational Research* 278.1 (2019), pp. 211–225.

[Tem05] Horst Tempelmeier. *Bestandsmanagement in supply chains*. BoD–Books on Demand, 2005.

[TH14] Michael Ten Hompel and Michael Henke. “Logistik 4.0”. In: *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Springer, 2014, pp. 615–624.

[TPV17] Kevin Tierney, Dario Pacino, and Stefan Voß. “Solving the pre-marshalling problem to optimality with A* and IDA*”. In: *Flexible Services and Manufacturing Journal* 29.2 (2017), pp. 223–259.

[TSD18] Michael Ten Hompel, Thorsten Schmidt, and Johannes Dregger. *Materialflusssysteme: Förder-und Lagertechnik*. Springer, 2018.

[TT16] Shunji Tanaka and Kenta Takii. “A Faster Branch-and-Bound Algorithm for the Block Relocation Problem”. In: *IEEE Transactions on Automation Science and Engineering* 13.1 (2016). Conference Name: IEEE Transactions on Automation Science and Engineering, pp. 181–190.

[TT18] Shunji Tanaka and Kevin Tierney. “Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm”. In: *European Journal of Operational Research* 264.1 (2018), pp. 165–180.

[TV16] Kevin Tierney and Stefan Voß. “Solving the Robust Container Pre-Marshalling Problem”. In: *Computational Logistics*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 131–145.

[TZL12] Lixin Tang, Ren Zhao, and Jiyin Liu. “Models and algorithms for shuffling problems in steel plants”. In: *Naval Research Logistics (NRL)* 59.7 (2012), pp. 502–524.

[Ull+15] Günter Ullrich et al. “Automated guided vehicle systems”. In: *Springer-Verlag Berlin Heidelberg. doi* 10 (2015), pp. 978–3.

[Ulm17] Marlin Wolf Ulmer. *Approximate dynamic programming for dynamic vehicle routing*. Vol. 61. Operations Research/Computer Science Interfaces Series. Springer, 2017.

[US 23] U.S. Bureau of Labor Statistics. *Databases, tables, calculators by subject*. https://data.bls.gov/. [Accessed 2023-01-17, 10:08]. 2023.

[Vah14] Behnam Vahdani. “Vehicle positioning in cell manufacturing systems via robust optimization”. In: *Applied Soft Computing* 24 (2014), pp. 78–85.

[VG00] Jeroen P Van Den Berg and AJRM Gademann. "Simulation study of an automated storage/retrieval system". In: *International Journal of Production Research* 38.6 (2000), pp. 1339–1356.

[VTF16] Marialena Vagia, Aksel A Transeth, and Sigurd A Fjerdingen. "A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed?" In: *Applied ergonomics* 53 (2016), pp. 190–202.

[WDM08] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI magazine* 29.1 (2008), pp. 9–9.

[YJD21] Peng Yang, Guang Jin, and Guofang Duan. "Modelling and analysis for multi-deep compact robotic mobile fulfilment system". In: *International Journal of Production Research* (2021), pp. 1–16.

[YJM04] Jian Yang, Patrick Jaillet, and Hani Mahmassani. "Real-time multivehicle truckload pickup and delivery problems". In: *Transportation Science* 38.2 (2004), pp. 135–148.

[ZBM19] Bernard Zweers, Sandjai Bhulai, and Rob van der Mei. "Optimizing preprocessing and relocation moves in the Stochastic Container Relocation Problem". In: *European Journal of Operational Research* (2019).

[Zha+10] Huidong Zhang, Songshan Guo, Wenbin Zhu, Andrew Lim, and Brenda Cheang. "An Investigation of IDA* Algorithms for the Container Relocation Problem". In: *Trends in Applied Intelligent Systems*. Lecture Notes in Computer Science. 2010, pp. 31–40.

[Zhu+12] Wenbin Zhu, Hu Qin, Andrew Lim, and Huidong Zhang. "Iterative Deepening A* Algorithms for the Container Relocation Problem". In: *IEEE Transactions on Automation Science and Engineering* 9.4 (2012), pp. 710–722.

[ZKX16] Bipan Zou, M. B. M. de Koster, and Xianhao Xu. "Evaluating Dedicated and Shared Storage Policies in Robot-Based Compact Storage and Retrieval Systems". In: *ERIM Report Series Reference* (2016).

[ZW06] G. Zäpfel and M. Wasner. "Warehouse sequencing in the steel supply chain as a generalized job shop model". In: *International Journal of Production Economics* 104 (2006), pp. 482–501.

Appendix

A.1 Figures

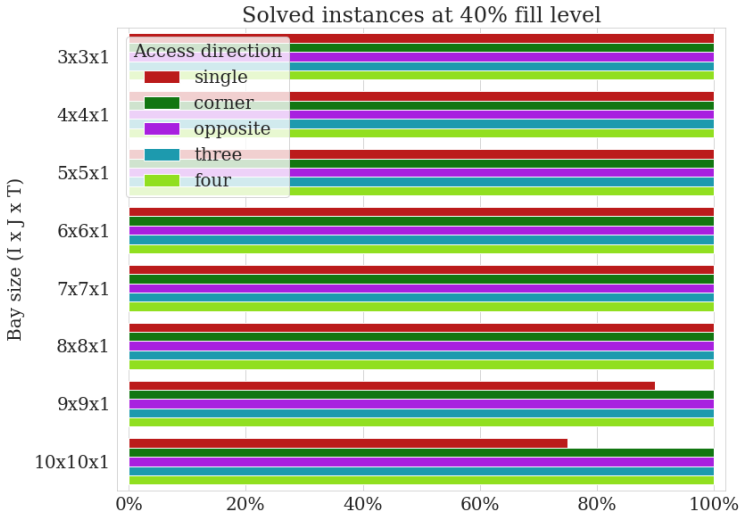


Fig. A.1 Overview of solved instances for a single tier at a fill level of 40%.

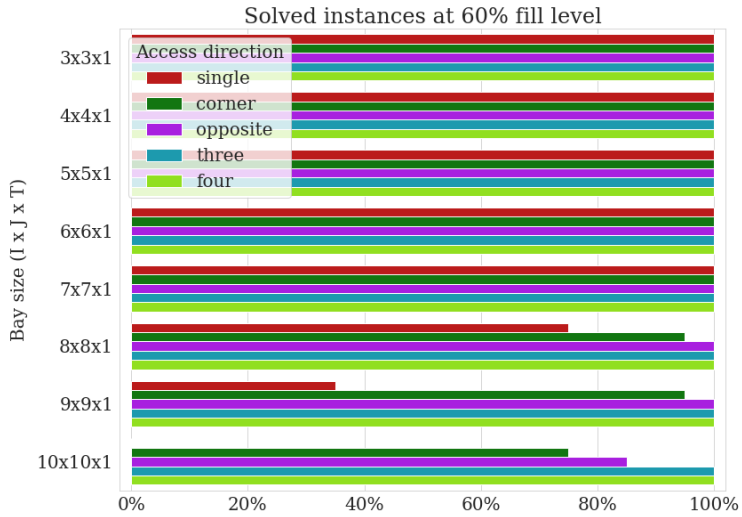


Fig. A.2 Overview of solved instances for a single tier at a fill level of 60%.

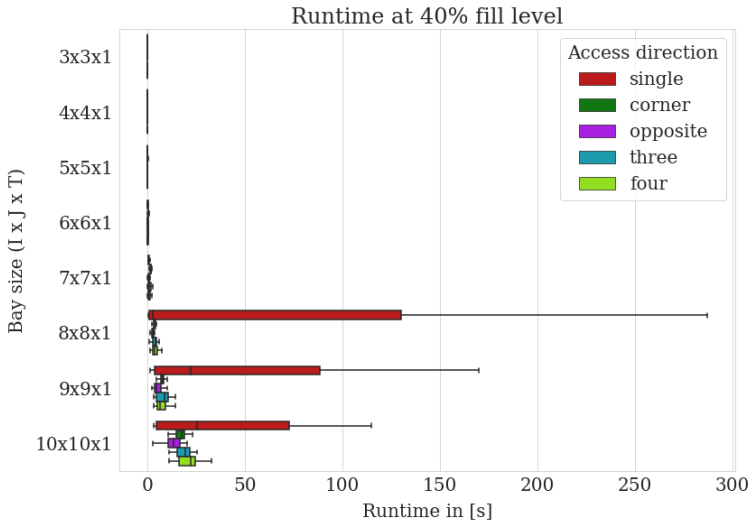


Fig. A.3 Box plots (without outliers) of the runtimes for a single tier at a fill level of 40%.

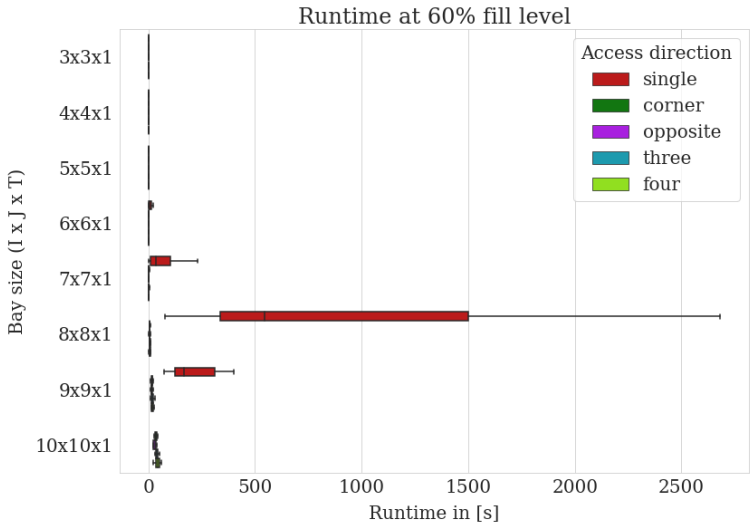


Fig. A.4 Box plots (without outliers) of the runtimes for a single tier at a fill level of 60%.

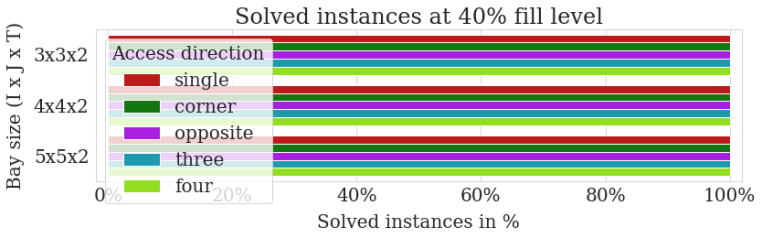


Fig. A.5 Overview of solved instances with two tiers at a fill level of 40%.

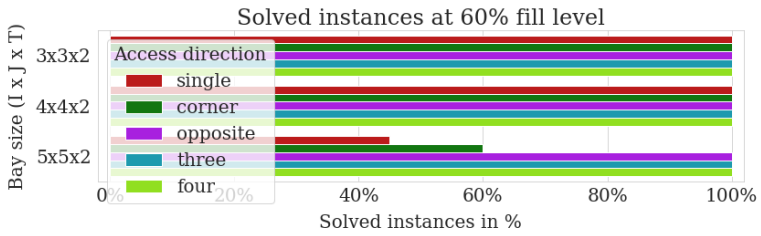


Fig. A.6 Overview of solved instances with two tiers at a fill level of 60%.

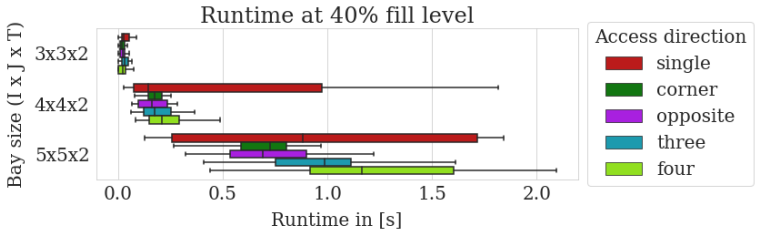


Fig. A.7 Box plots (without outliers) of the runtimes with two tiers at a fill level of 40%.

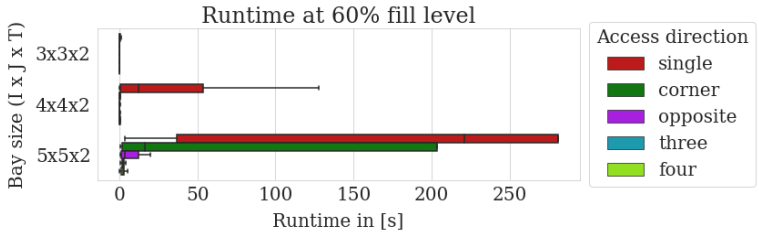


Fig. A.8 Box plots (without outliers) of the runtimes with two tiers at a fill level of 60%.

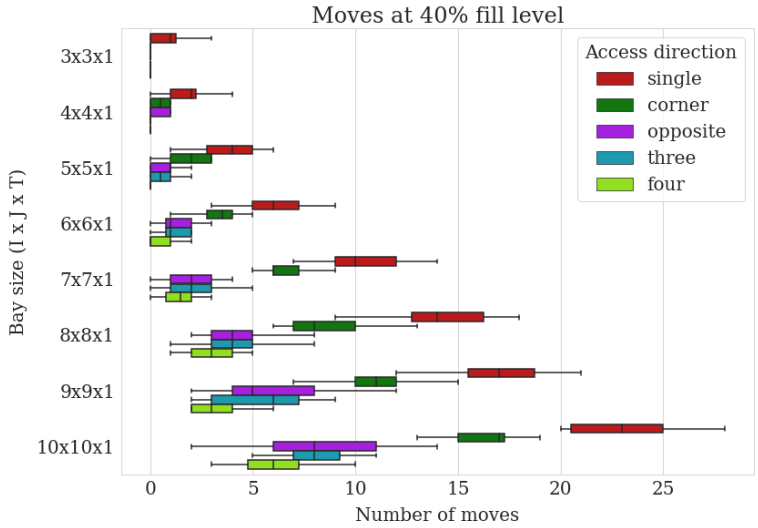


Fig. A.9 Box plots (without outliers) for a single tier showing the number of moves for each access variant and each bay size at the fill level of 40%.

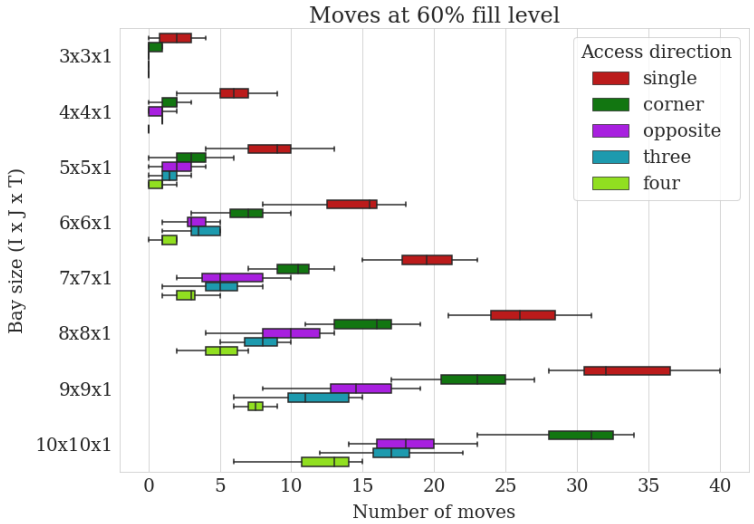


Fig. A.10 Box plots (without outliers) for a single tier showing the number of moves for each access variant and each bay size at the fill level of 60%.

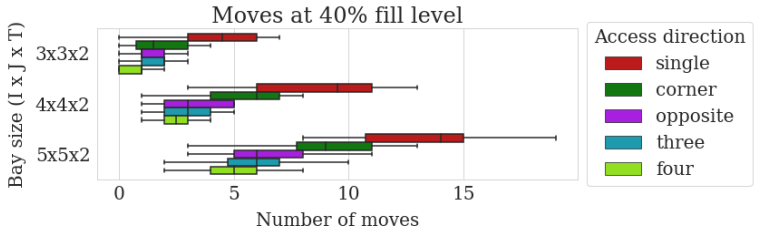


Fig. A.11 Box plots (without outliers) for two tiers showing the number of moves for each access variant and each bay size at the fill level of 40%.

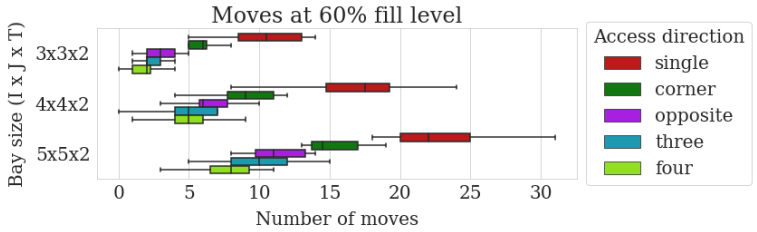


Fig. A.12 Box plots (without outliers) for two tiers showing the number of moves for each access variant and each bay size at the fill level of 60%.

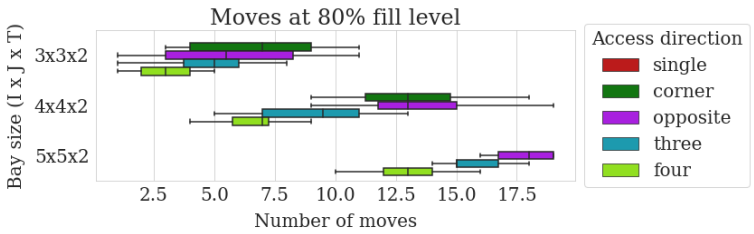


Fig. A.13 Box plots (without outliers) for two tiers showing the number of moves for each access variant and each bay size at the fill level of 80%.

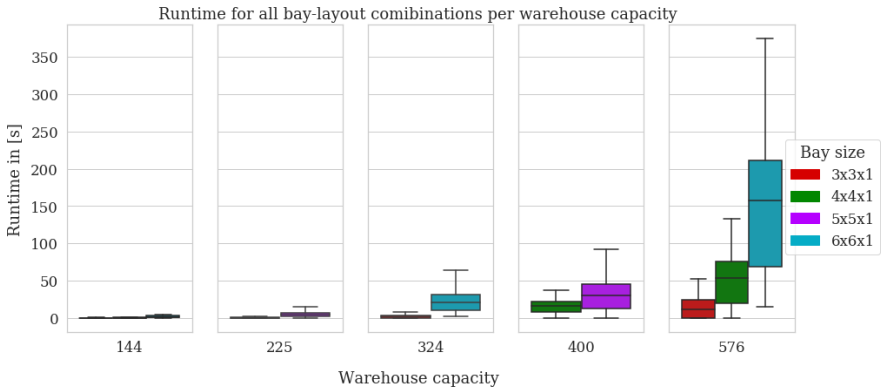


Fig. A.14 Box plots (without outliers) of the runtime in seconds for all warehouse capacities with several combinations of bay and layout sizes.

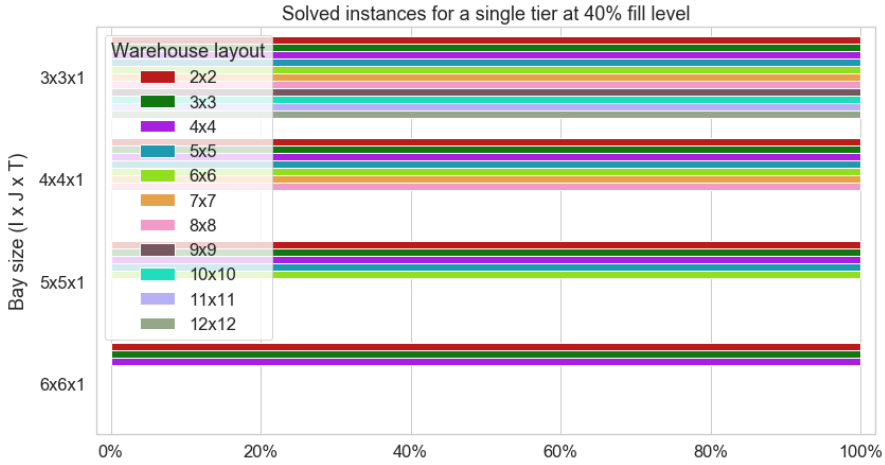


Fig. A.15 Solved instances per bay size for each layout size at a fill level of 40%.

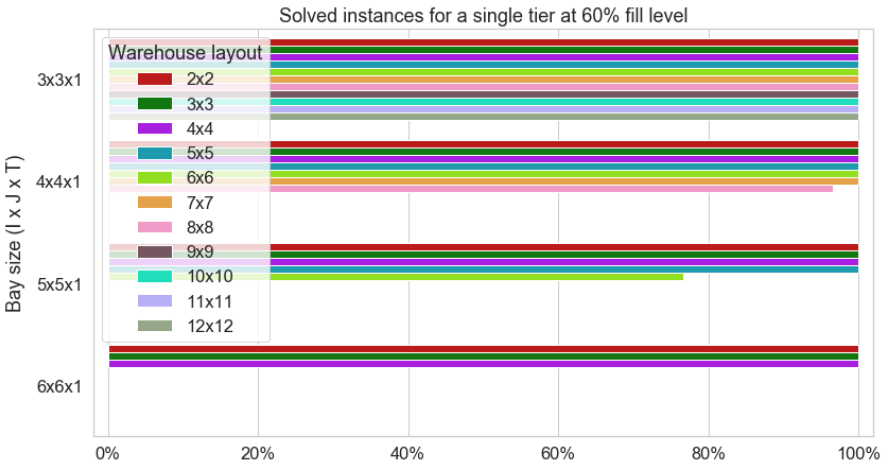


Fig. A.16 Solved instances per bay size for each layout size at a fill level of 60%.

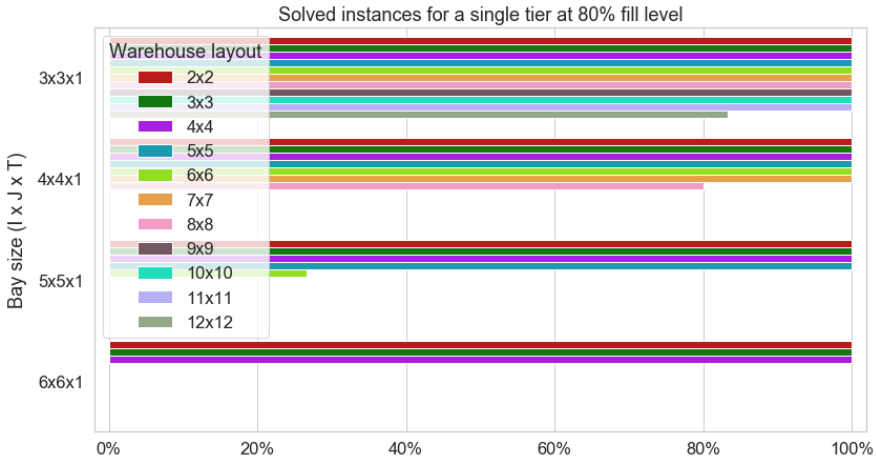


Fig. A.17 Solved instances per bay size for each layout size at a fill level of 80%.

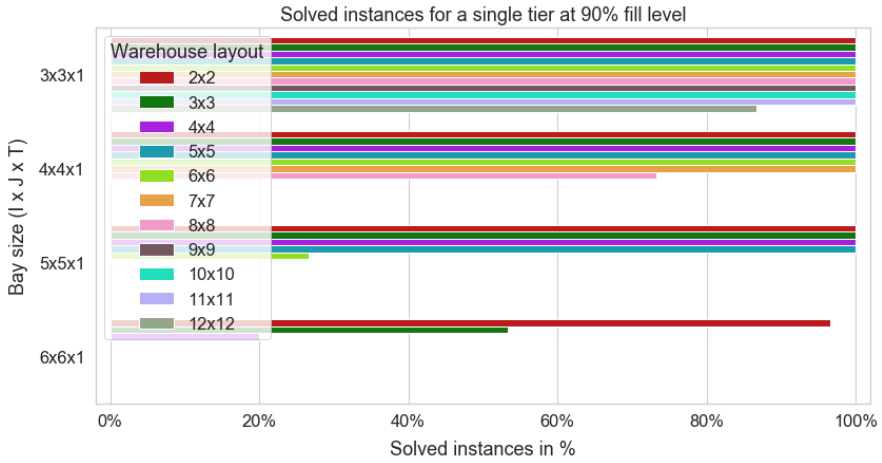


Fig. A.18 Solved instances per bay size for each layout size at a fill level of 90%.

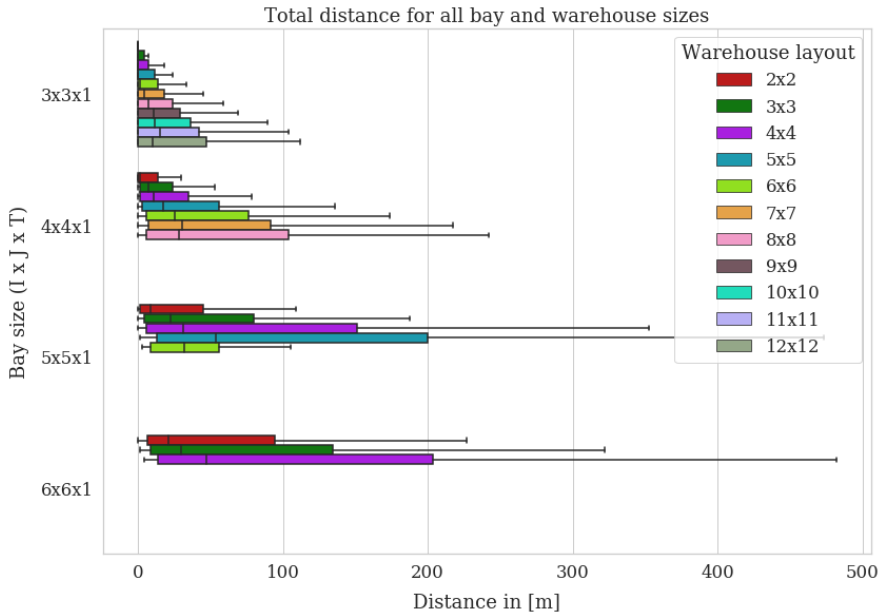


Fig. A.19 Box plots (without outliers) of the total move distance per bay size for each layout size.

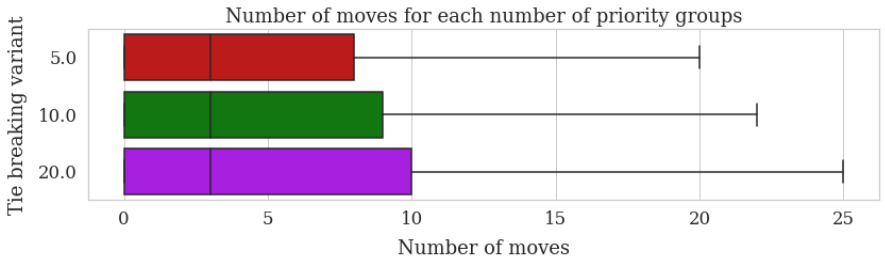


Fig. A.20 Box plots (without outliers) of the number of moves for each number of priority groups per bay size.

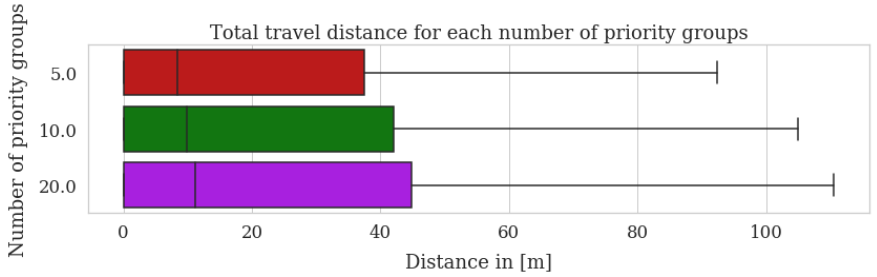


Fig. A.21 Box plots (without outliers) of the total move distance in meter for each number of priority groups per bay size.

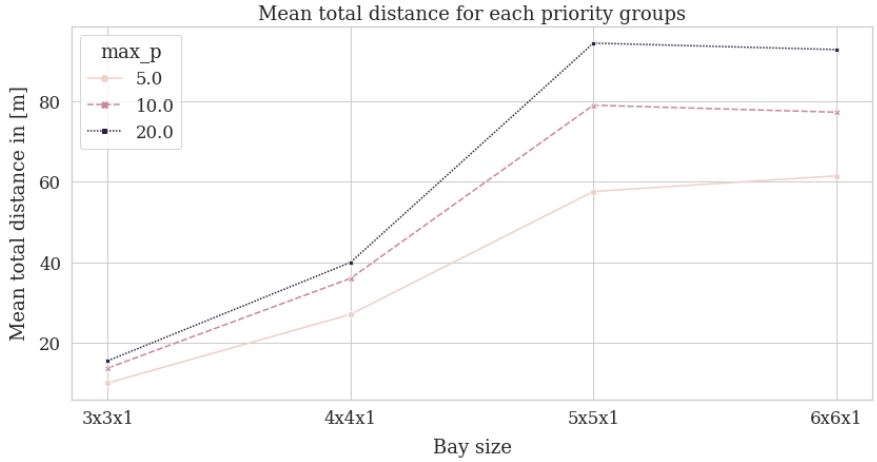


Fig. A.22 Mean total move distance for each number of priority groups per bay size. Only instances that are solved by all three variants are included.

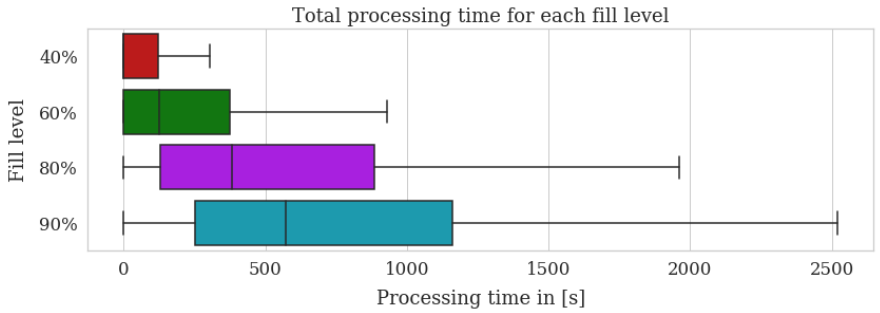


Fig. A.23 Box plots (without outliers) of the total move processing times in meter for each fill level.

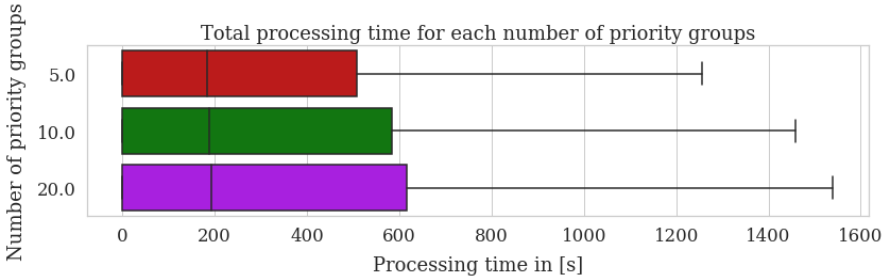


Fig. A.24 Box plots (without outliers) of the total move processing time for each number of priority groups per bay size.

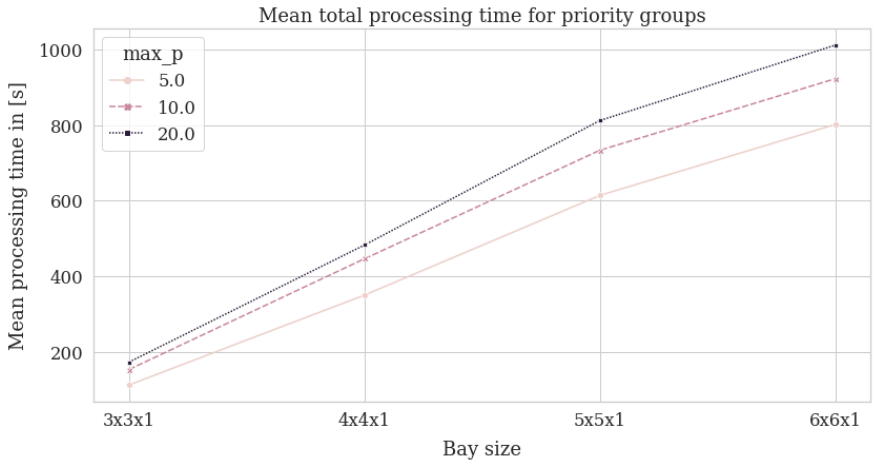


Fig. A.25 Mean total move processing time in seconds for each number of priority groups per bay size.

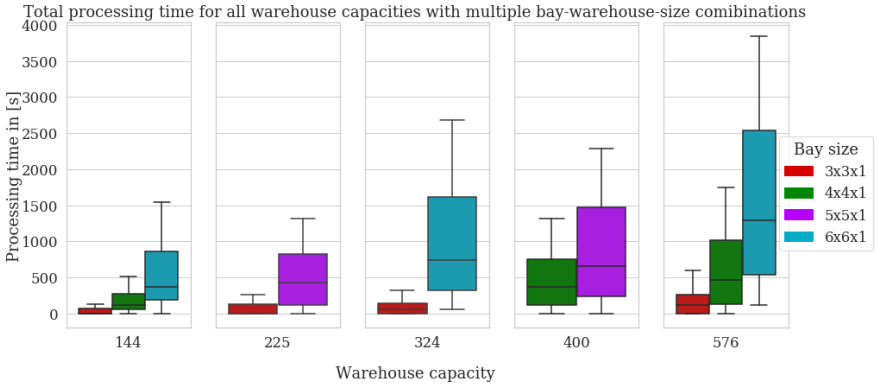


Fig. A.26 Box plots (without outliers) of the total move processing time in seconds for all warehouse capacities with several combinations of bay and layout sizes.

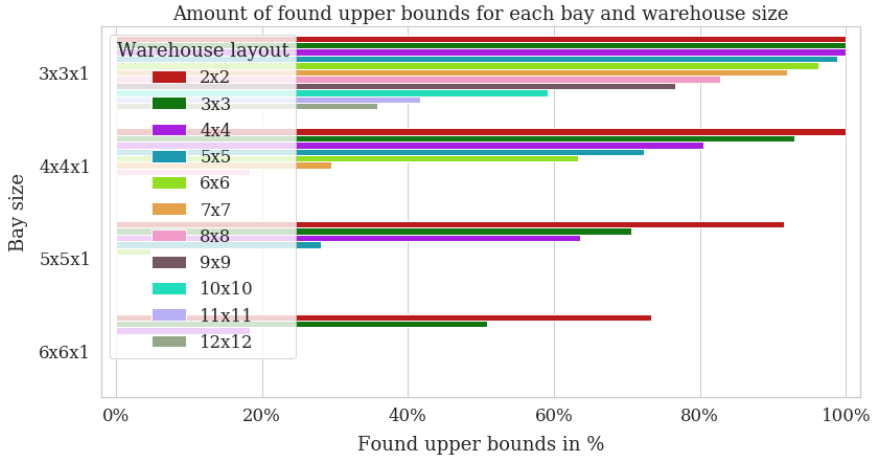


Fig. A.27 Amount of found upper bounds for all bay and layout sizes.

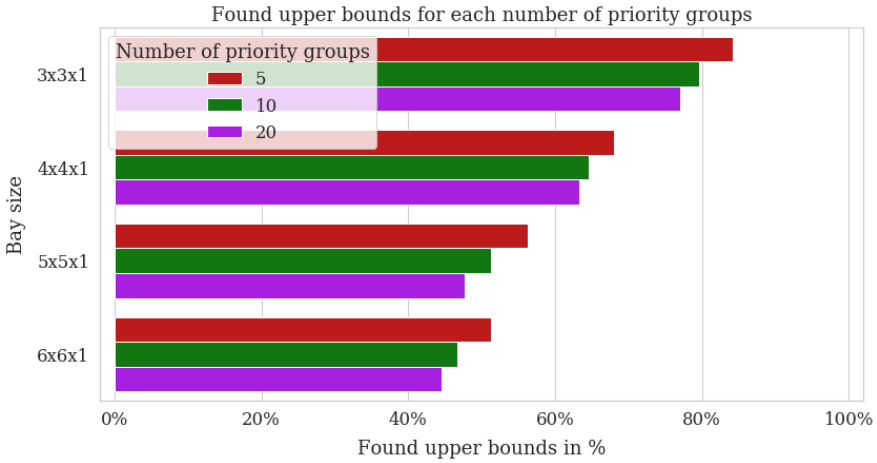


Fig. A.28 Amount of found upper bounds for priority groups and all bay sizes.

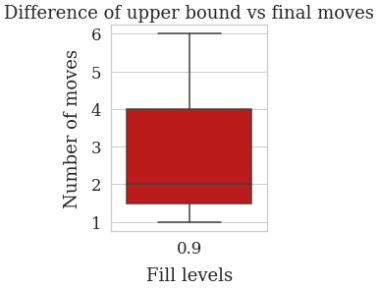


Fig. A.29 Box plots (without outliers) of the difference of the upper bound compared to the final number of moves for all fill levels.

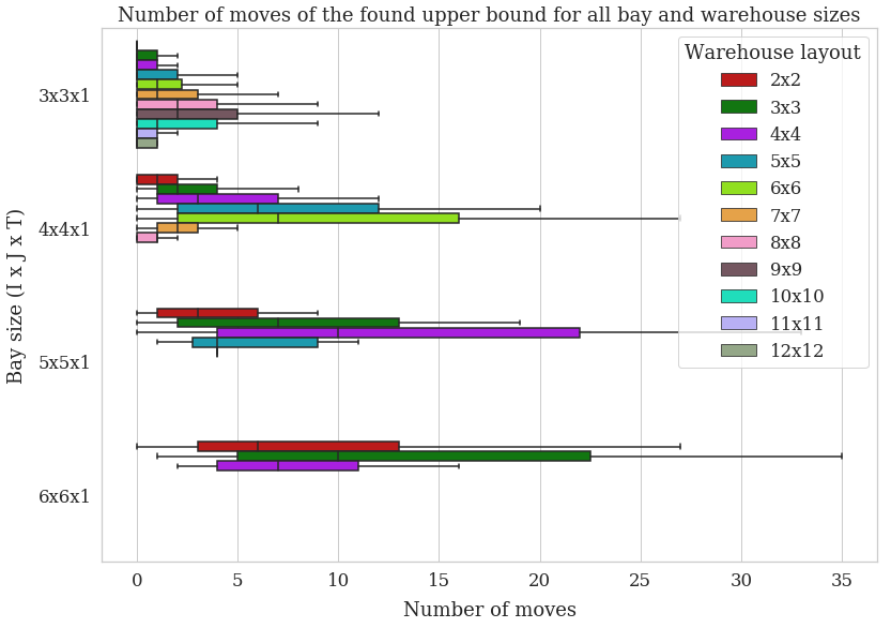


Fig. A.30 Box plots (without outliers) of the number of moves of the found upper bounds for all bay and layout sizes.

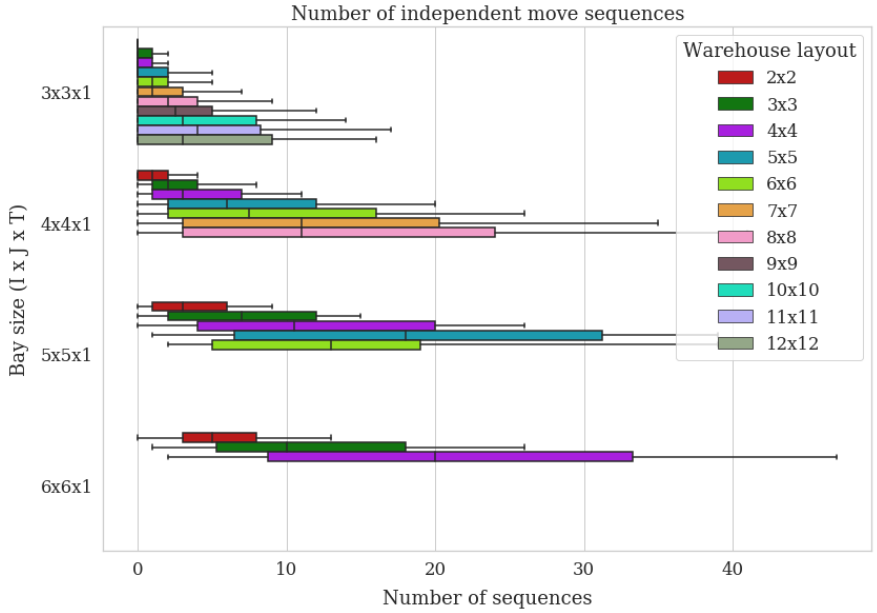


Fig. A.31 Box plots (without outliers) of the number of independent move sequences for all bay and layout sizes.

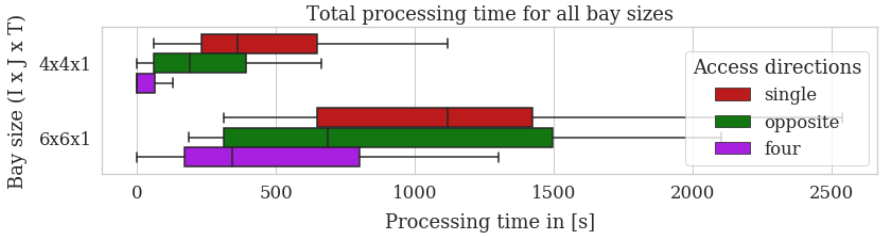


Fig. A.32 Box plots (without outliers) of the total move processing time in seconds per bay size for each access direction.

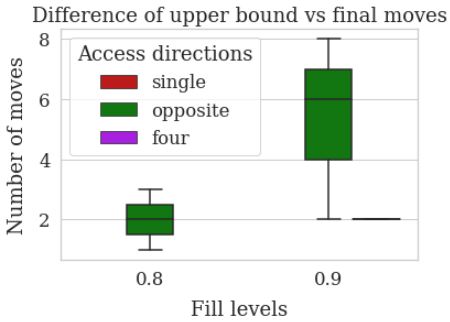


Fig. A.33 Box plots (without outliers) of the number of moves for the upper bound over the fill level for each access direction.

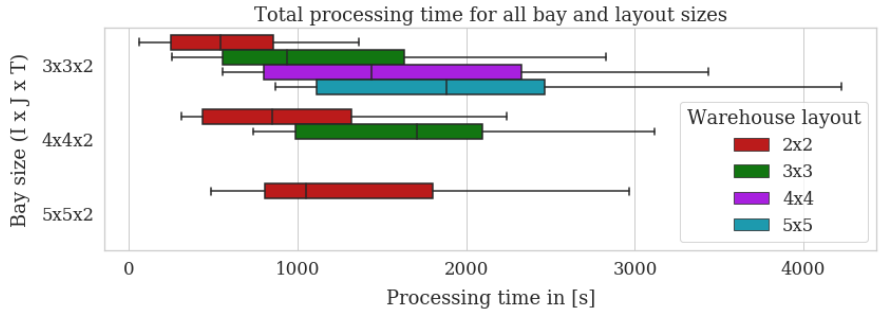


Fig. A.34 Box plots (without outliers) of the total move processing time in seconds per bay size for each layout size.

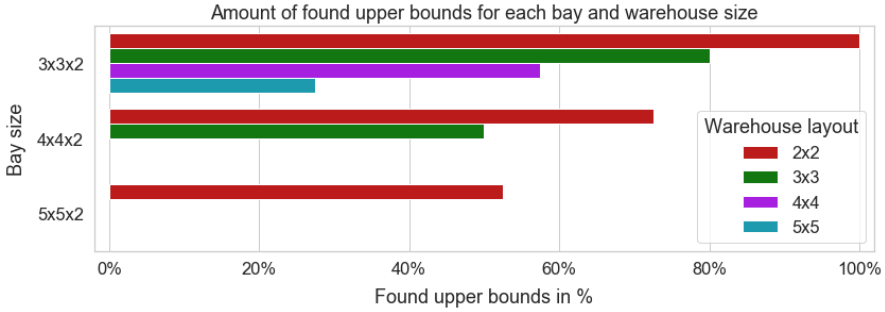


Fig. A.35 Amount of found upper bounds for all bay and layout sizes.

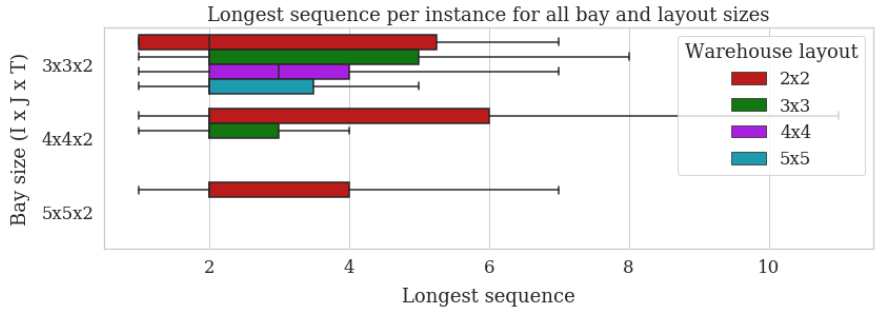


Fig. A.36 Box plots (without outliers) of the longest sequence per instance for all bay and warehouse layout sizes.

A.2 Tables

Bay size (I x J x T)	Access variant	Fill level	No. solved	No. infeasible	No. timeout	Mean moves	Mean visited nodes	Mean runtime
3x3x1	corner	40	20	0	0	1.0	2.0	0.02
3x3x1	corner	60	20	0	0	1.0	2.0	0.01
3x3x1	corner	80	20	0	0	5.0	18.0	0.11
3x3x1	four	40	20	0	0	0.0	1.0	0.00
3x3x1	four	60	20	0	0	0.0	1.0	0.00
3x3x1	four	80	20	0	0	1.0	2.0	0.01
3x3x1	opposite	40	20	0	0	0.0	1.0	0.00
3x3x1	opposite	60	20	0	0	1.0	2.0	0.02
3x3x1	opposite	80	20	0	0	2.0	3.0	0.03
3x3x1	single	40	20	0	0	3.0	5.0	0.02
3x3x1	single	60	20	0	0	4.0	6.0	0.02
3x3x1	single	80	11	9	0	7.0	31.0	0.06
3x3x1	three	40	20	0	0	0.0	1.0	0.00
3x3x1	three	60	20	0	0	0.0	1.0	0.00
3x3x1	three	80	20	0	0	1.0	2.0	0.02
4x4x1	corner	40	20	0	0	1.0	2.0	0.03
4x4x1	corner	60	20	0	0	4.0	5.0	0.09
4x4x1	corner	80	20	0	0	7.0	248.0	4.54
4x4x1	four	40	20	0	0	0.0	1.0	0.00
4x4x1	four	60	20	0	0	1.0	2.0	0.05
4x4x1	four	80	20	0	0	1.0	2.0	0.04
4x4x1	opposite	40	20	0	0	1.0	2.0	0.04
4x4x1	opposite	60	20	0	0	2.0	3.0	0.06
4x4x1	opposite	80	20	0	0	4.0	5.0	0.07
4x4x1	single	40	20	0	0	4.0	6.0	0.04
4x4x1	single	60	20	0	0	9.0	91.0	0.43
4x4x1	single	80	9	11	0	13.0	2011.0	5.91
4x4x1	three	40	20	0	0	1.0	2.0	0.04
4x4x1	three	60	20	0	0	2.0	3.0	0.06
4x4x1	three	80	20	0	0	2.0	3.0	0.05
5x5x1	corner	40	20	0	0	3.0	4.0	0.20
5x5x1	corner	60	20	0	0	6.0	13.0	0.59
5x5x1	corner	80	20	0	0	11.0	1073.0	55.66
5x5x1	four	40	20	0	0	1.0	2.0	0.15
5x5x1	four	60	20	0	0	2.0	3.0	0.25
5x5x1	four	80	20	0	0	3.0	4.0	0.29
5x5x1	opposite	40	20	0	0	2.0	3.0	0.17
5x5x1	opposite	60	20	0	0	4.0	5.0	0.36
5x5x1	opposite	80	20	0	0	7.0	8.0	0.35
5x5x1	single	40	20	0	0	6.0	7.0	0.09
5x5x1	single	60	20	0	0	13.0	186.0	1.95
5x5x1	single	80	20	0	0	23.0	62279.0	423.43
5x5x1	three	40	20	0	0	2.0	3.0	0.19
5x5x1	three	60	20	0	0	3.0	4.0	0.30
5x5x1	three	80	20	0	0	5.0	6.0	0.37
6x6x1	corner	40	20	0	0	6.0	7.0	0.67

6x6x1	corner	60	20	0	0	10.0	11.0	1.41
6x6x1	corner	80	18	0	2	14.0	39939.0	3231.90
6x6x1	four	40	20	0	0	2.0	3.0	0.70
6x6x1	four	60	20	0	0	2.0	3.0	0.84
6x6x1	four	80	20	0	0	6.0	7.0	1.36
6x6x1	opposite	40	20	0	0	3.0	4.0	0.50
6x6x1	opposite	60	20	0	0	5.0	6.0	0.84
6x6x1	opposite	80	20	0	0	12.0	545.0	52.08
6x6x1	single	40	20	0	0	11.0	78.0	1.77
6x6x1	single	60	20	0	0	18.0	4495.0	105.03
6x6x1	single	80	10	0	10	26.0	150220.0	2300.22
6x6x1	three	40	20	0	0	2.0	3.0	0.57
6x6x1	three	60	20	0	0	5.0	6.0	1.39
6x6x1	three	80	20	0	0	7.0	8.0	1.25
7x7x1	corner	40	20	0	0	10.0	11.0	2.28
7x7x1	corner	60	20	0	0	13.0	14.0	3.68
7x7x1	corner	80	8	0	12	20.0	11431.0	2444.61
7x7x1	four	40	20	0	0	3.0	4.0	2.45
7x7x1	four	60	20	0	0	5.0	6.0	3.32
7x7x1	four	80	20	0	0	9.0	10.0	4.33
7x7x1	opposite	40	20	0	0	4.0	5.0	1.23
7x7x1	opposite	60	20	0	0	10.0	37.0	13.20
7x7x1	opposite	80	19	0	1	15.0	1114.0	331.86
7x7x1	single	40	20	0	0	14.0	165.0	6.76
7x7x1	single	60	20	0	0	23.0	18116.0	723.35
7x7x1	single	80	1	0	19	29.0	47054.0	1400.01
7x7x1	three	40	20	0	0	5.0	6.0	2.63
7x7x1	three	60	20	0	0	8.0	10.0	4.19
7x7x1	three	80	20	0	0	11.0	58.0	33.10
8x8x1	corner	40	20	0	0	13.0	14.0	4.77
8x8x1	corner	60	19	0	1	19.0	91.0	40.65
8x8x1	corner	80	2	0	18	25.0	145.0	37.47
8x8x1	four	40	20	0	0	5.0	6.0	7.26
8x8x1	four	60	20	0	0	7.0	8.0	9.52
8x8x1	four	80	20	0	0	12.0	13.0	10.67
8x8x1	opposite	40	20	0	0	8.0	9.0	4.51
8x8x1	opposite	60	20	0	0	13.0	1704.0	993.64
8x8x1	opposite	80	14	0	6	24.0	8684.0	2635.02
8x8x1	single	40	20	0	0	18.0	12397.0	884.72
8x8x1	single	60	15	0	5	31.0	38486.0	2680.28
8x8x1	single	80	0	0	20	-	-	-
8x8x1	three	40	20	0	0	8.0	9.0	6.00
8x8x1	three	60	20	0	0	10.0	11.0	9.23
8x8x1	three	80	18	0	2	17.0	149.0	78.08
9x9x1	corner	40	20	0	0	18.0	19.0	12.33
9x9x1	corner	60	19	0	1	27.0	28.0	20.72
9x9x1	corner	80	2	0	18	36.0	3515.0	1529.92
9x9x1	four	40	20	0	0	6.0	7.0	14.23
9x9x1	four	60	20	0	0	11.0	12.0	30.43
9x9x1	four	80	20	0	0	20.0	21.0	27.57
9x9x1	opposite	40	20	0	0	12.0	13.0	10.33

9x9x1	opposite	60	20	0	0	19.0	53.0	55.47
9x9x1	opposite	80	4	0	16	27.0	939.0	663.01
9x9x1	single	40	18	0	2	21.0	15643.0	1781.60
9x9x1	single	60	7	0	13	40.0	33546.0	3189.55
9x9x1	single	80	0	0	20	-	-	-
9x9x1	three	40	20	0	0	9.0	10.0	14.30
9x9x1	three	60	20	0	0	15.0	16.0	30.43
9x9x1	three	80	14	0	6	23.0	347.0	581.62
10x10x1	corner	40	20	0	0	22.0	23.0	23.04
10x10x1	corner	60	15	0	5	34.0	1313.0	1395.86
10x10x1	corner	80	0	0	20	-	-	-
10x10x1	four	40	20	0	0	10.0	11.0	32.84
10x10x1	four	60	20	0	0	20.0	21.0	61.24
10x10x1	four	80	16	0	4	24.0	162.0	306.45
10x10x1	opposite	40	20	0	0	14.0	15.0	20.18
10x10x1	opposite	60	17	0	3	23.0	86.0	109.63
10x10x1	opposite	80	0	0	20	-	-	-
10x10x1	single	40	15	0	5	28.0	18106.0	3021.48
10x10x1	single	60	0	0	20	-	-	-
10x10x1	single	80	0	0	20	-	-	-
10x10x1	three	40	20	0	0	11.0	12.0	25.31
10x10x1	three	60	20	0	0	25.0	26.0	54.72
10x10x1	three	80	11	0	9	33.0	942.0	1821.09

Table A.1 Results for the single bay UPMP with a single tier rounded to the second decimal

Tie breaking variant	Bay size (I x J x T)	Warehouse Size (L x W)	No. of priority groups	Fill level	No. solved	No. infeasible	No. time-out	Mean moves	Mean visited nodes	Mean runtime
f_dist	3x3x1	2x2	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	2x2	5	0.6	10	0	0	0.10	1.10	0.00
f_dist	3x3x1	2x2	5	0.8	10	0	0	0.10	1.10	0.00
f_dist	3x3x1	2x2	5	0.9	10	0	0	0.60	1.80	0.01
f_dist	3x3x1	2x2	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	2x2	10	0.6	10	0	0	0.10	1.10	0.00
f_dist	3x3x1	2x2	10	0.8	10	0	0	0.20	1.20	0.00
f_dist	3x3x1	2x2	10	0.9	10	0	0	0.50	1.70	0.01
f_dist	3x3x1	2x2	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	2x2	20	0.6	10	0	0	0.10	1.10	0.00
f_dist	3x3x1	2x2	20	0.8	10	0	0	0.20	1.20	0.00
f_dist	3x3x1	2x2	20	0.9	10	0	0	0.60	1.80	0.01
f_dist	3x3x1	3x3	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	3x3	5	0.6	10	0	0	0.30	1.30	0.03
f_dist	3x3x1	3x3	5	0.8	10	0	0	0.30	1.90	0.06
f_dist	3x3x1	3x3	5	0.9	10	0	0	1.30	8.90	0.26
f_dist	3x3x1	3x3	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	3x3	10	0.6	10	0	0	0.30	1.30	0.04
f_dist	3x3x1	3x3	10	0.8	10	0	0	0.40	2.00	0.08

f_dist	3x3x1	3x3	10	0.9	10	0	0	1.40	9.10	0.29
f_dist	3x3x1	3x3	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	3x3	20	0.6	10	0	0	0.30	1.30	0.04
f_dist	3x3x1	3x3	20	0.8	10	0	0	0.40	2.00	0.08
f_dist	3x3x1	3x3	20	0.9	10	0	0	1.40	5.10	0.20
f_dist	3x3x1	4x4	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	4x4	5	0.6	10	0	0	0.20	1.20	0.08
f_dist	3x3x1	4x4	5	0.8	10	0	0	0.70	3.30	0.59
f_dist	3x3x1	4x4	5	0.9	10	0	0	1.50	35.50	3.21
f_dist	3x3x1	4x4	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	4x4	10	0.6	10	0	0	0.30	1.30	0.12
f_dist	3x3x1	4x4	10	0.8	10	0	0	1.10	5.40	1.09
f_dist	3x3x1	4x4	10	0.9	10	0	0	1.80	36.20	3.53
f_dist	3x3x1	4x4	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	4x4	20	0.6	10	0	0	0.30	1.30	0.13
f_dist	3x3x1	4x4	20	0.8	10	0	0	1.10	5.20	1.14
f_dist	3x3x1	4x4	20	0.9	10	0	0	1.90	36.40	3.91
f_dist	3x3x1	5x5	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	5x5	5	0.6	10	0	0	0.20	1.20	0.21
f_dist	3x3x1	5x5	5	0.8	10	0	0	1.00	4.90	2.67
f_dist	3x3x1	5x5	5	0.9	9	0	1	2.00	83.44	18.24
f_dist	3x3x1	5x5	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	5x5	10	0.6	10	0	0	0.20	1.20	0.22
f_dist	3x3x1	5x5	10	0.8	10	0	0	1.40	23.10	11.71
f_dist	3x3x1	5x5	10	0.9	9	0	1	2.67	194.44	41.77
f_dist	3x3x1	5x5	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	5x5	20	0.6	10	0	0	0.20	1.20	0.24
f_dist	3x3x1	5x5	20	0.8	10	0	0	1.70	154.50	63.18
f_dist	3x3x1	5x5	20	0.9	9	0	1	3.00	277.44	62.41
f_dist	3x3x1	6x6	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	6x6	5	0.6	10	0	0	0.20	1.20	0.49
f_dist	3x3x1	6x6	5	0.8	10	0	0	1.60	15.30	18.78
f_dist	3x3x1	6x6	5	0.9	9	0	1	2.11	113.00	55.27
f_dist	3x3x1	6x6	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	6x6	10	0.6	10	0	0	0.30	1.30	0.76
f_dist	3x3x1	6x6	10	0.8	9	0	1	1.78	23.11	28.45
f_dist	3x3x1	6x6	10	0.9	8	0	2	3.00	202.50	103.06
f_dist	3x3x1	6x6	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	6x6	20	0.6	10	0	0	0.40	1.40	1.10
f_dist	3x3x1	6x6	20	0.8	8	0	2	2.00	59.50	70.80
f_dist	3x3x1	6x6	20	0.9	7	0	3	3.14	302.00	157.55
f_dist	3x3x1	7x7	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	7x7	5	0.6	10	0	0	0.60	3.90	15.22
f_dist	3x3x1	7x7	5	0.8	10	0	0	2.00	41.30	97.72
f_dist	3x3x1	7x7	5	0.9	6	0	4	2.17	225.50	238.29
f_dist	3x3x1	7x7	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	7x7	10	0.6	10	0	0	0.50	3.80	15.22
f_dist	3x3x1	7x7	10	0.8	8	0	2	1.88	31.00	78.18
f_dist	3x3x1	7x7	10	0.9	4	0	6	2.50	101.75	126.11
f_dist	3x3x1	7x7	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	7x7	20	0.6	10	0	0	0.60	3.80	16.29

f_dist	3x3x1	7x7	20	0.8	8	0	2	2.12	52.62	128.32
f_dist	3x3x1	7x7	20	0.9	4	0	6	3.00	178.50	218.33
f_dist	3x3x1	8x8	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	8x8	5	0.6	10	0	0	0.40	2.00	10.18
f_dist	3x3x1	8x8	5	0.8	6	0	4	1.83	24.67	117.10
f_dist	3x3x1	8x8	5	0.9	3	0	7	2.00	15.33	46.57
f_dist	3x3x1	8x8	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	8x8	10	0.6	10	0	0	0.60	8.00	56.87
f_dist	3x3x1	8x8	10	0.8	4	0	6	1.75	26.00	125.18
f_dist	3x3x1	8x8	10	0.9	2	0	8	3.00	92.50	229.78
f_dist	3x3x1	8x8	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	8x8	20	0.6	10	0	0	0.90	9.10	72.38
f_dist	3x3x1	8x8	20	0.8	3	0	7	1.67	17.00	91.76
f_dist	3x3x1	8x8	20	0.9	0	0	10	-	-	-
f_dist	3x3x1	9x9	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	9x9	5	0.6	10	0	0	0.70	4.00	55.03
f_dist	3x3x1	9x9	5	0.8	3	0	7	2.33	36.00	310.57
f_dist	3x3x1	9x9	5	0.9	3	0	7	3.00	82.00	357.98
f_dist	3x3x1	9x9	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	9x9	10	0.6	8	0	2	0.75	3.12	39.64
f_dist	3x3x1	9x9	10	0.8	2	0	8	2.00	23.00	209.17
f_dist	3x3x1	9x9	10	0.9	0	0	10	-	-	-
f_dist	3x3x1	9x9	20	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	9x9	20	0.6	9	0	1	1.11	7.67	109.59
f_dist	3x3x1	9x9	20	0.8	1	0	9	1.00	2.00	13.37
f_dist	3x3x1	9x9	20	0.9	0	0	10	-	-	-
f_dist	3x3x1	10x10	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	10x10	5	0.6	10	0	0	1.10	3.80	88.91
f_dist	3x3x1	10x10	5	0.8	1	0	9	2.00	8.00	151.18
f_dist	3x3x1	10x10	5	0.9	0	0	10	-	-	-
f_dist	3x3x1	10x10	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	10x10	10	0.6	8	0	2	1.12	3.62	86.94
f_dist	3x3x1	10x10	10	0.8	0	0	10	-	-	-
f_dist	3x3x1	10x10	10	0.9	0	0	10	-	-	-
f_dist	3x3x1	10x10	20	0.4	10	0	0	0.10	1.10	3.34
f_dist	3x3x1	10x10	20	0.6	7	0	3	1.43	5.14	142.94
f_dist	3x3x1	10x10	20	0.8	0	0	10	-	-	-
f_dist	3x3x1	10x10	20	0.9	0	0	10	-	-	-
f_dist	3x3x1	11x11	5	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	11x11	5	0.6	10	0	0	0.90	2.60	93.03
f_dist	3x3x1	11x11	5	0.8	0	0	10	-	-	-
f_dist	3x3x1	11x11	5	0.9	0	0	10	-	-	-
f_dist	3x3x1	11x11	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	11x11	10	0.6	8	0	2	1.00	2.25	74.11
f_dist	3x3x1	11x11	10	0.8	0	0	10	-	-	-
f_dist	3x3x1	11x11	10	0.9	0	0	10	-	-	-
f_dist	3x3x1	11x11	20	0.4	10	0	0	0.10	1.10	6.03
f_dist	3x3x1	11x11	20	0.6	7	0	3	1.29	3.57	161.90
f_dist	3x3x1	11x11	20	0.8	0	0	10	-	-	-
f_dist	3x3x1	11x11	20	0.9	0	0	10	-	-	-
f_dist	3x3x1	12x12	5	0.4	10	0	0	0.00	1.00	0.00

f_dist	3x3x1	12x12	5	0.6	8	0	2	1.00	2.12	97.37
f_dist	3x3x1	12x12	5	0.8	0	0	10	-	-	-
f_dist	3x3x1	12x12	5	0.9	0	0	10	-	-	-
f_dist	3x3x1	12x12	10	0.4	10	0	0	0.00	1.00	0.00
f_dist	3x3x1	12x12	10	0.6	4	0	6	1.00	2.00	87.46
f_dist	3x3x1	12x12	10	0.8	0	0	10	-	-	-
f_dist	3x3x1	12x12	10	0.9	0	0	10	-	-	-
f_dist	3x3x1	12x12	20	0.4	10	0	0	0.10	1.10	8.76
f_dist	3x3x1	12x12	20	0.6	2	0	8	1.00	2.00	92.12
f_dist	3x3x1	12x12	20	0.8	0	0	10	-	-	-
f_dist	3x3x1	12x12	20	0.9	0	0	10	-	-	-
f_dist	4x4x1	2x2	5	0.4	10	0	0	0.20	1.20	0.01
f_dist	4x4x1	2x2	5	0.6	10	0	0	0.10	1.10	0.01
f_dist	4x4x1	2x2	5	0.8	10	0	0	1.50	80.20	1.71
f_dist	4x4x1	2x2	5	0.9	10	0	0	1.70	35.00	0.47
f_dist	4x4x1	2x2	10	0.4	10	0	0	0.20	1.20	0.01
f_dist	4x4x1	2x2	10	0.6	10	0	0	0.30	1.30	0.02
f_dist	4x4x1	2x2	10	0.8	10	0	0	2.20	109.10	2.61
f_dist	4x4x1	2x2	10	0.9	10	0	0	2.20	36.80	0.57
f_dist	4x4x1	2x2	20	0.4	10	0	0	0.20	1.20	0.01
f_dist	4x4x1	2x2	20	0.6	10	0	0	0.40	1.40	0.03
f_dist	4x4x1	2x2	20	0.8	10	0	0	2.20	105.10	2.72
f_dist	4x4x1	2x2	20	0.9	10	0	0	2.30	65.30	1.02
f_dist	4x4x1	3x3	5	0.4	10	0	0	0.40	1.40	0.14
f_dist	4x4x1	3x3	5	0.6	10	0	0	0.90	2.70	0.59
f_dist	4x4x1	3x3	5	0.8	9	0	1	2.67	299.22	33.17
f_dist	4x4x1	3x3	5	0.9	8	0	2	3.62	1561.38	72.88
f_dist	4x4x1	3x3	10	0.4	10	0	0	0.40	1.40	0.14
f_dist	4x4x1	3x3	10	0.6	10	0	0	1.20	8.00	1.86
f_dist	4x4x1	3x3	10	0.8	7	0	3	3.57	252.71	31.46
f_dist	4x4x1	3x3	10	0.9	6	0	4	4.50	2956.50	152.53
f_dist	4x4x1	3x3	20	0.4	10	0	0	0.40	1.40	0.16
f_dist	4x4x1	3x3	20	0.6	10	0	0	1.20	8.10	2.08
f_dist	4x4x1	3x3	20	0.8	7	0	3	3.57	487.29	61.76
f_dist	4x4x1	3x3	20	0.9	6	0	4	4.50	3326.00	180.63
f_dist	4x4x1	4x4	5	0.4	10	0	0	0.50	2.90	1.74
f_dist	4x4x1	4x4	5	0.6	10	0	0	2.10	31.20	24.59
f_dist	4x4x1	4x4	5	0.8	5	0	5	2.60	266.20	108.79
f_dist	4x4x1	4x4	5	0.9	1	0	9	3.00	34.00	11.58
f_dist	4x4x1	4x4	10	0.4	10	0	0	0.60	2.90	1.96
f_dist	4x4x1	4x4	10	0.6	10	0	0	2.50	189.90	118.78
f_dist	4x4x1	4x4	10	0.8	3	0	7	3.00	63.00	34.86
f_dist	4x4x1	4x4	10	0.9	1	0	9	5.00	971.00	221.85
f_dist	4x4x1	4x4	20	0.4	10	0	0	0.60	3.30	2.55
f_dist	4x4x1	4x4	20	0.6	9	0	1	2.44	111.44	89.28
f_dist	4x4x1	4x4	20	0.8	3	0	7	3.00	56.00	32.11
f_dist	4x4x1	4x4	20	0.9	0	0	10	-	-	-
f_dist	4x4x1	5x5	5	0.4	10	0	0	0.80	3.30	6.53
f_dist	4x4x1	5x5	5	0.6	7	0	3	2.29	35.14	90.81
f_dist	4x4x1	5x5	5	0.8	2	0	8	2.50	169.00	226.05
f_dist	4x4x1	5x5	5	0.9	0	0	10	-	-	-

f_dist	4x4x1	5x5	10	0.4	10	0	0	1.00	3.60	8.82
f_dist	4x4x1	5x5	10	0.6	6	0	4	2.50	26.00	64.33
f_dist	4x4x1	5x5	10	0.8	1	0	9	3.00	24.00	46.94
f_dist	4x4x1	5x5	10	0.9	0	0	10	-	-	-
f_dist	4x4x1	5x5	20	0.4	10	0	0	1.00	3.90	10.57
f_dist	4x4x1	5x5	20	0.6	5	0	5	2.40	40.20	110.97
f_dist	4x4x1	5x5	20	0.8	1	0	9	3.00	19.00	39.57
f_dist	4x4x1	5x5	20	0.9	0	0	10	-	-	-
f_dist	4x4x1	6x6	5	0.4	9	0	1	0.56	1.56	5.01
f_dist	4x4x1	6x6	5	0.6	5	0	5	2.20	36.60	209.89
f_dist	4x4x1	6x6	5	0.8	0	0	10	-	-	-
f_dist	4x4x1	6x6	5	0.9	0	0	10	-	-	-
f_dist	4x4x1	6x6	10	0.4	9	0	1	0.78	3.67	24.98
f_dist	4x4x1	6x6	10	0.6	2	0	8	1.50	3.50	22.44
f_dist	4x4x1	6x6	10	0.8	0	0	10	-	-	-
f_dist	4x4x1	6x6	10	0.9	0	0	10	-	-	-
f_dist	4x4x1	6x6	20	0.4	9	0	1	0.78	4.11	30.79
f_dist	4x4x1	6x6	20	0.6	1	0	9	1.00	2.00	9.64
f_dist	4x4x1	6x6	20	0.8	0	0	10	-	-	-
f_dist	4x4x1	6x6	20	0.9	0	0	10	-	-	-
f_dist	4x4x1	7x7	5	0.4	9	0	1	1.00	3.22	45.01
f_dist	4x4x1	7x7	5	0.6	2	0	8	2.00	16.00	232.77
f_dist	4x4x1	7x7	5	0.8	0	0	10	-	-	-
f_dist	4x4x1	7x7	5	0.9	0	0	10	-	-	-
f_dist	4x4x1	7x7	10	0.4	9	0	1	1.33	5.56	77.23
f_dist	4x4x1	7x7	10	0.6	1	0	9	2.00	5.00	81.22
f_dist	4x4x1	7x7	10	0.8	0	0	10	-	-	-
f_dist	4x4x1	7x7	10	0.9	0	0	10	-	-	-
f_dist	4x4x1	7x7	20	0.4	9	0	1	1.33	5.78	86.11
f_dist	4x4x1	7x7	20	0.6	1	0	9	2.00	5.00	85.97
f_dist	4x4x1	7x7	20	0.8	0	0	10	-	-	-
f_dist	4x4x1	7x7	20	0.9	0	0	10	-	-	-
f_dist	4x4x1	8x8	5	0.4	8	0	2	0.88	4.00	88.39
f_dist	4x4x1	8x8	5	0.6	0	0	10	-	-	-
f_dist	4x4x1	8x8	5	0.8	0	0	10	-	-	-
f_dist	4x4x1	8x8	5	0.9	0	0	10	-	-	-
f_dist	4x4x1	8x8	10	0.4	7	0	3	0.71	2.29	51.68
f_dist	4x4x1	8x8	10	0.6	0	0	10	-	-	-
f_dist	4x4x1	8x8	10	0.8	0	0	10	-	-	-
f_dist	4x4x1	8x8	10	0.9	0	0	10	-	-	-
f_dist	4x4x1	8x8	20	0.4	8	0	2	1.12	4.38	129.20
f_dist	4x4x1	8x8	20	0.6	0	0	10	-	-	-
f_dist	4x4x1	8x8	20	0.8	0	0	10	-	-	-
f_dist	4x4x1	8x8	20	0.9	0	0	10	-	-	-
f_dist	5x5x1	2x2	5	0.4	10	0	0	0.20	1.20	0.03
f_dist	5x5x1	2x2	5	0.6	10	0	0	1.30	4.70	0.44
f_dist	5x5x1	2x2	5	0.8	7	0	3	4.14	986.71	37.88
f_dist	5x5x1	2x2	5	0.9	9	0	1	6.11	7204.11	192.49
f_dist	5x5x1	2x2	10	0.4	10	0	0	0.10	1.10	0.01
f_dist	5x5x1	2x2	10	0.6	10	0	0	1.80	7.70	0.77
f_dist	5x5x1	2x2	10	0.8	7	0	3	5.00	1923.00	71.48

f_dist	5x5x1	2x2	10	0.9	7	0	3	6.14	8907.86	196.41
f_dist	5x5x1	2x2	20	0.4	10	0	0	0.20	1.20	0.03
f_dist	5x5x1	2x2	20	0.6	10	0	0	2.00	19.90	2.10
f_dist	5x5x1	2x2	20	0.8	3	0	7	5.33	5220.00	187.59
f_dist	5x5x1	2x2	20	0.9	6	0	4	6.33	7253.17	230.45
f_dist	5x5x1	3x3	5	0.4	10	0	0	0.70	2.40	1.19
f_dist	5x5x1	3x3	5	0.6	9	0	1	3.44	180.33	78.45
f_dist	5x5x1	3x3	5	0.8	1	0	9	4.00	599.00	179.79
f_dist	5x5x1	3x3	5	0.9	0	0	10	-	-	-
f_dist	5x5x1	3x3	10	0.4	10	0	0	0.80	2.90	1.69
f_dist	5x5x1	3x3	10	0.6	5	0	5	4.00	387.60	150.94
f_dist	5x5x1	3x3	10	0.8	0	0	10	-	-	-
f_dist	5x5x1	3x3	10	0.9	0	0	10	-	-	-
f_dist	5x5x1	3x3	20	0.4	10	0	0	1.00	8.20	5.30
f_dist	5x5x1	3x3	20	0.6	5	0	5	4.40	493.20	210.03
f_dist	5x5x1	3x3	20	0.8	0	0	10	-	-	-
f_dist	5x5x1	3x3	20	0.9	0	0	10	-	-	-
f_dist	5x5x1	4x4	5	0.4	10	0	0	1.50	13.50	29.01
f_dist	5x5x1	4x4	5	0.6	3	0	7	4.00	220.67	403.79
f_dist	5x5x1	4x4	5	0.8	0	0	10	-	-	-
f_dist	5x5x1	4x4	5	0.9	0	0	10	-	-	-
f_dist	5x5x1	4x4	10	0.4	10	0	0	1.90	14.00	32.41
f_dist	5x5x1	4x4	10	0.6	0	0	10	-	-	-
f_dist	5x5x1	4x4	10	0.8	0	0	10	-	-	-
f_dist	5x5x1	4x4	10	0.9	0	0	10	-	-	-
f_dist	5x5x1	4x4	20	0.4	10	0	0	2.00	30.10	65.50
f_dist	5x5x1	4x4	20	0.6	0	0	10	-	-	-
f_dist	5x5x1	4x4	20	0.8	0	0	10	-	-	-
f_dist	5x5x1	4x4	20	0.9	0	0	10	-	-	-
f_dist	5x5x1	5x5	5	0.4	8	0	2	1.75	10.75	75.57
f_dist	5x5x1	5x5	5	0.6	0	0	10	-	-	-
f_dist	5x5x1	5x5	5	0.8	0	0	10	-	-	-
f_dist	5x5x1	5x5	5	0.9	0	0	10	-	-	-
f_dist	5x5x1	5x5	10	0.4	7	0	3	2.14	15.14	109.19
f_dist	5x5x1	5x5	10	0.6	0	0	10	-	-	-
f_dist	5x5x1	5x5	10	0.8	0	0	10	-	-	-
f_dist	5x5x1	5x5	10	0.9	0	0	10	-	-	-
f_dist	5x5x1	5x5	20	0.4	4	0	6	2.25	20.25	143.46
f_dist	5x5x1	5x5	20	0.6	0	0	10	-	-	-
f_dist	5x5x1	5x5	20	0.8	0	0	10	-	-	-
f_dist	5x5x1	5x5	20	0.9	0	0	10	-	-	-
f_dist	5x5x1	6x6	5	0.4	1	0	9	2.00	8.00	207.83
f_dist	5x5x1	6x6	5	0.6	0	0	10	-	-	-
f_dist	5x5x1	6x6	5	0.8	0	0	10	-	-	-
f_dist	5x5x1	6x6	5	0.9	0	0	10	-	-	-
f_dist	5x5x1	6x6	10	0.4	0	0	10	-	-	-
f_dist	5x5x1	6x6	10	0.6	0	0	10	-	-	-
f_dist	5x5x1	6x6	10	0.8	0	0	10	-	-	-
f_dist	5x5x1	6x6	10	0.9	0	0	10	-	-	-
f_dist	5x5x1	6x6	20	0.4	0	0	10	-	-	-
f_dist	5x5x1	6x6	20	0.6	0	0	10	-	-	-

f_dist	5x5x1	6x6	20	0.8	0	0	10	-	-	-
f_dist	5x5x1	6x6	20	0.9	0	0	10	-	-	-
f_dist	6x6x1	2x2	5	0.4	10	0	0	1.30	10.20	1.95
f_dist	6x6x1	2x2	5	0.6	10	0	0	3.80	820.40	95.38
f_dist	6x6x1	2x2	5	0.8	1	0	9	5.00	1370.00	102.03
f_dist	6x6x1	2x2	5	0.9	0	0	10	-	-	-
f_dist	6x6x1	2x2	10	0.4	10	0	0	1.80	11.00	2.33
f_dist	6x6x1	2x2	10	0.6	9	0	1	3.89	464.22	64.21
f_dist	6x6x1	2x2	10	0.8	0	0	10	-	-	-
f_dist	6x6x1	2x2	10	0.9	0	0	10	-	-	-
f_dist	6x6x1	2x2	20	0.4	10	0	0	1.80	9.60	2.23
f_dist	6x6x1	2x2	20	0.6	7	0	3	4.00	450.29	67.51
f_dist	6x6x1	2x2	20	0.8	0	0	10	-	-	-
f_dist	6x6x1	2x2	20	0.9	0	0	10	-	-	-
f_dist	6x6x1	3x3	5	0.4	9	0	1	2.22	60.33	52.90
f_dist	6x6x1	3x3	5	0.6	0	0	10	-	-	-
f_dist	6x6x1	3x3	5	0.8	0	0	10	-	-	-
f_dist	6x6x1	3x3	5	0.9	0	0	10	-	-	-
f_dist	6x6x1	3x3	10	0.4	7	0	3	2.43	21.29	26.40
f_dist	6x6x1	3x3	10	0.6	0	0	10	-	-	-
f_dist	6x6x1	3x3	10	0.8	0	0	10	-	-	-
f_dist	6x6x1	3x3	10	0.9	0	0	10	-	-	-
f_dist	6x6x1	3x3	20	0.4	7	0	3	2.43	21.86	29.00
f_dist	6x6x1	3x3	20	0.6	0	0	10	-	-	-
f_dist	6x6x1	3x3	20	0.8	0	0	10	-	-	-
f_dist	6x6x1	3x3	20	0.9	0	0	10	-	-	-
f_dist	6x6x1	4x4	5	0.4	3	0	7	2.67	20.67	95.21
f_dist	6x6x1	4x4	5	0.6	0	0	10	-	-	-
f_dist	6x6x1	4x4	5	0.8	0	0	10	-	-	-
f_dist	6x6x1	4x4	5	0.9	0	0	10	-	-	-
f_dist	6x6x1	4x4	10	0.4	2	0	8	3.00	53.00	241.65
f_dist	6x6x1	4x4	10	0.6	0	0	10	-	-	-
f_dist	6x6x1	4x4	10	0.8	0	0	10	-	-	-
f_dist	6x6x1	4x4	10	0.9	0	0	10	-	-	-
f_dist	6x6x1	4x4	20	0.4	1	0	9	3.00	38.00	184.46
f_dist	6x6x1	4x4	20	0.6	0	0	10	-	-	-
f_dist	6x6x1	4x4	20	0.8	0	0	10	-	-	-
f_dist	6x6x1	4x4	20	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	2x2	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	2x2	5	0.6	10	0	0	0.10	1.10	0.00
f_dist_h	3x3x1	2x2	5	0.8	10	0	0	0.10	1.10	0.00
f_dist_h	3x3x1	2x2	5	0.9	10	0	0	0.60	1.80	0.01
f_dist_h	3x3x1	2x2	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	2x2	10	0.6	10	0	0	0.10	1.10	0.00
f_dist_h	3x3x1	2x2	10	0.8	10	0	0	0.20	1.20	0.00
f_dist_h	3x3x1	2x2	10	0.9	10	0	0	0.50	1.70	0.01
f_dist_h	3x3x1	2x2	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	2x2	20	0.6	10	0	0	0.10	1.10	0.00
f_dist_h	3x3x1	2x2	20	0.8	10	0	0	0.20	1.20	0.00
f_dist_h	3x3x1	2x2	20	0.9	10	0	0	0.60	1.80	0.01
f_dist_h	3x3x1	3x3	5	0.4	10	0	0	0.00	1.00	0.00

f_dist_h	3x3x1	3x3	5	0.6	10	0	0	0.30	1.30	0.03
f_dist_h	3x3x1	3x3	5	0.8	10	0	0	0.30	1.80	0.06
f_dist_h	3x3x1	3x3	5	0.9	10	0	0	1.30	8.70	0.25
f_dist_h	3x3x1	3x3	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	3x3	10	0.6	10	0	0	0.30	1.30	0.03
f_dist_h	3x3x1	3x3	10	0.8	10	0	0	0.40	1.90	0.07
f_dist_h	3x3x1	3x3	10	0.9	10	0	0	1.40	8.80	0.27
f_dist_h	3x3x1	3x3	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	3x3	20	0.6	10	0	0	0.30	1.30	0.04
f_dist_h	3x3x1	3x3	20	0.8	10	0	0	0.40	1.90	0.07
f_dist_h	3x3x1	3x3	20	0.9	10	0	0	1.40	4.60	0.17
f_dist_h	3x3x1	4x4	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	4x4	5	0.6	10	0	0	0.20	1.20	0.08
f_dist_h	3x3x1	4x4	5	0.8	10	0	0	0.70	3.00	0.51
f_dist_h	3x3x1	4x4	5	0.9	10	0	0	1.50	34.70	3.10
f_dist_h	3x3x1	4x4	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	4x4	10	0.6	10	0	0	0.30	1.30	0.12
f_dist_h	3x3x1	4x4	10	0.8	10	0	0	1.10	4.90	0.95
f_dist_h	3x3x1	4x4	10	0.9	10	0	0	1.80	34.90	3.34
f_dist_h	3x3x1	4x4	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	4x4	20	0.6	10	0	0	0.30	1.30	0.13
f_dist_h	3x3x1	4x4	20	0.8	10	0	0	1.10	4.90	1.04
f_dist_h	3x3x1	4x4	20	0.9	10	0	0	1.90	35.10	3.69
f_dist_h	3x3x1	5x5	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	5x5	5	0.6	10	0	0	0.20	1.20	0.21
f_dist_h	3x3x1	5x5	5	0.8	10	0	0	1.00	4.90	2.67
f_dist_h	3x3x1	5x5	5	0.9	9	0	1	2.00	76.44	16.13
f_dist_h	3x3x1	5x5	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	5x5	10	0.6	10	0	0	0.20	1.20	0.22
f_dist_h	3x3x1	5x5	10	0.8	10	0	0	1.40	20.50	10.06
f_dist_h	3x3x1	5x5	10	0.9	9	0	1	2.67	184.33	38.11
f_dist_h	3x3x1	5x5	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	5x5	20	0.6	10	0	0	0.20	1.20	0.24
f_dist_h	3x3x1	5x5	20	0.8	10	0	0	1.70	149.70	59.93
f_dist_h	3x3x1	5x5	20	0.9	9	0	1	3.00	262.33	57.33
f_dist_h	3x3x1	6x6	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	6x6	5	0.6	10	0	0	0.20	1.20	0.49
f_dist_h	3x3x1	6x6	5	0.8	10	0	0	1.60	14.10	17.13
f_dist_h	3x3x1	6x6	5	0.9	9	0	1	2.11	107.89	51.43
f_dist_h	3x3x1	6x6	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	6x6	10	0.6	10	0	0	0.30	1.30	0.76
f_dist_h	3x3x1	6x6	10	0.8	9	0	1	1.78	22.11	26.66
f_dist_h	3x3x1	6x6	10	0.9	8	0	2	3.00	192.25	94.77
f_dist_h	3x3x1	6x6	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	6x6	20	0.6	10	0	0	0.40	1.40	1.10
f_dist_h	3x3x1	6x6	20	0.8	8	0	2	2.00	50.88	61.04
f_dist_h	3x3x1	6x6	20	0.9	7	0	3	3.14	293.86	150.60
f_dist_h	3x3x1	7x7	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	7x7	5	0.6	10	0	0	0.60	3.40	12.56
f_dist_h	3x3x1	7x7	5	0.8	10	0	0	2.00	38.90	89.45
f_dist_h	3x3x1	7x7	5	0.9	7	0	3	2.43	260.14	254.09

f_dist_h	3x3x1	7x7	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	7x7	10	0.6	10	0	0	0.50	3.30	12.49
f_dist_h	3x3x1	7x7	10	0.8	8	0	2	1.88	28.88	71.14
f_dist_h	3x3x1	7x7	10	0.9	4	0	6	2.50	91.75	112.59
f_dist_h	3x3x1	7x7	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	7x7	20	0.6	10	0	0	0.60	3.40	13.91
f_dist_h	3x3x1	7x7	20	0.8	8	0	2	2.12	46.88	108.54
f_dist_h	3x3x1	7x7	20	0.9	4	0	6	3.00	173.50	208.91
f_dist_h	3x3x1	8x8	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	8x8	5	0.6	10	0	0	0.40	1.90	9.13
f_dist_h	3x3x1	8x8	5	0.8	6	0	4	1.83	20.33	94.74
f_dist_h	3x3x1	8x8	5	0.9	4	0	6	2.50	88.25	180.22
f_dist_h	3x3x1	8x8	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	8x8	10	0.6	10	0	0	0.60	7.60	52.85
f_dist_h	3x3x1	8x8	10	0.8	4	0	6	1.75	24.25	113.00
f_dist_h	3x3x1	8x8	10	0.9	2	0	8	3.00	86.00	205.34
f_dist_h	3x3x1	8x8	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	8x8	20	0.6	10	0	0	0.90	9.00	71.34
f_dist_h	3x3x1	8x8	20	0.8	3	0	7	1.67	16.00	84.23
f_dist_h	3x3x1	8x8	20	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	9x9	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	9x9	5	0.6	10	0	0	0.70	3.40	44.18
f_dist_h	3x3x1	9x9	5	0.8	3	0	7	2.33	32.33	272.22
f_dist_h	3x3x1	9x9	5	0.9	3	0	7	3.00	71.67	297.09
f_dist_h	3x3x1	9x9	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	9x9	10	0.6	8	0	2	0.75	3.00	37.36
f_dist_h	3x3x1	9x9	10	0.8	2	0	8	2.00	22.00	196.50
f_dist_h	3x3x1	9x9	10	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	9x9	20	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	9x9	20	0.6	9	0	1	1.11	6.89	94.12
f_dist_h	3x3x1	9x9	20	0.8	1	0	9	1.00	2.00	13.41
f_dist_h	3x3x1	9x9	20	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	10x10	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	10x10	5	0.6	10	0	0	1.10	3.60	82.76
f_dist_h	3x3x1	10x10	5	0.8	1	0	9	2.00	8.00	151.40
f_dist_h	3x3x1	10x10	5	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	10x10	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	10x10	10	0.6	8	0	2	1.12	3.12	70.18
f_dist_h	3x3x1	10x10	10	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	10x10	10	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	10x10	20	0.4	10	0	0	0.10	1.10	3.35
f_dist_h	3x3x1	10x10	20	0.6	7	0	3	1.43	4.14	108.27
f_dist_h	3x3x1	10x10	20	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	10x10	20	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	11x11	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	11x11	5	0.6	10	0	0	0.90	2.40	81.78
f_dist_h	3x3x1	11x11	5	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	11x11	5	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	11x11	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	11x11	10	0.6	8	0	2	1.00	2.25	73.92
f_dist_h	3x3x1	11x11	10	0.8	0	0	10	-	-	-

f_dist_h	3x3x1	11x11	10	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	11x11	20	0.4	10	0	0	0.10	1.10	6.03
f_dist_h	3x3x1	11x11	20	0.6	7	0	3	1.29	3.29	144.03
f_dist_h	3x3x1	11x11	20	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	11x11	20	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	12x12	5	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	12x12	5	0.6	8	0	2	1.00	2.12	97.50
f_dist_h	3x3x1	12x12	5	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	12x12	5	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	12x12	10	0.4	10	0	0	0.00	1.00	0.00
f_dist_h	3x3x1	12x12	10	0.6	4	0	6	1.00	2.00	87.74
f_dist_h	3x3x1	12x12	10	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	12x12	10	0.9	0	0	10	-	-	-
f_dist_h	3x3x1	12x12	20	0.4	10	0	0	0.10	1.10	8.77
f_dist_h	3x3x1	12x12	20	0.6	2	0	8	1.00	2.00	92.04
f_dist_h	3x3x1	12x12	20	0.8	0	0	10	-	-	-
f_dist_h	3x3x1	12x12	20	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	2x2	5	0.4	10	0	0	0.20	1.20	0.01
f_dist_h	4x4x1	2x2	5	0.6	10	0	0	0.10	1.10	0.01
f_dist_h	4x4x1	2x2	5	0.8	10	0	0	1.50	77.80	1.60
f_dist_h	4x4x1	2x2	5	0.9	10	0	0	1.70	31.30	0.41
f_dist_h	4x4x1	2x2	10	0.4	10	0	0	0.20	1.20	0.01
f_dist_h	4x4x1	2x2	10	0.6	10	0	0	0.30	1.30	0.02
f_dist_h	4x4x1	2x2	10	0.8	10	0	0	2.20	98.60	2.23
f_dist_h	4x4x1	2x2	10	0.9	10	0	0	2.20	34.50	0.52
f_dist_h	4x4x1	2x2	20	0.4	10	0	0	0.20	1.20	0.01
f_dist_h	4x4x1	2x2	20	0.6	10	0	0	0.40	1.40	0.03
f_dist_h	4x4x1	2x2	20	0.8	10	0	0	2.20	98.10	2.43
f_dist_h	4x4x1	2x2	20	0.9	10	0	0	2.30	62.20	0.95
f_dist_h	4x4x1	3x3	5	0.4	10	0	0	0.40	1.40	0.14
f_dist_h	4x4x1	3x3	5	0.6	10	0	0	0.90	2.60	0.56
f_dist_h	4x4x1	3x3	5	0.8	9	0	1	2.67	274.67	29.36
f_dist_h	4x4x1	3x3	5	0.9	8	0	2	3.62	1515.25	67.87
f_dist_h	4x4x1	3x3	10	0.4	10	0	0	0.40	1.40	0.15
f_dist_h	4x4x1	3x3	10	0.6	10	0	0	1.20	7.60	1.72
f_dist_h	4x4x1	3x3	10	0.8	7	0	3	3.57	229.57	27.03
f_dist_h	4x4x1	3x3	10	0.9	6	0	4	4.50	2902.83	145.67
f_dist_h	4x4x1	3x3	20	0.4	10	0	0	0.40	1.40	0.16
f_dist_h	4x4x1	3x3	20	0.6	10	0	0	1.20	7.60	1.87
f_dist_h	4x4x1	3x3	20	0.8	7	0	3	3.57	467.57	57.20
f_dist_h	4x4x1	3x3	20	0.9	6	0	4	4.50	3293.67	176.80
f_dist_h	4x4x1	4x4	5	0.4	10	0	0	0.50	2.60	1.49
f_dist_h	4x4x1	4x4	5	0.6	10	0	0	2.10	29.20	22.07
f_dist_h	4x4x1	4x4	5	0.8	5	0	5	2.60	254.80	100.46
f_dist_h	4x4x1	4x4	5	0.9	1	0	9	3.00	33.00	11.11
f_dist_h	4x4x1	4x4	10	0.4	10	0	0	0.60	2.70	1.71
f_dist_h	4x4x1	4x4	10	0.6	10	0	0	2.50	182.70	109.47
f_dist_h	4x4x1	4x4	10	0.8	3	0	7	3.00	59.00	31.59
f_dist_h	4x4x1	4x4	10	0.9	1	0	9	5.00	919.00	196.90
f_dist_h	4x4x1	4x4	20	0.4	10	0	0	0.60	2.70	1.84
f_dist_h	4x4x1	4x4	20	0.6	10	0	0	2.70	199.80	133.35

f_dist_h	4x4x1	4x4	20	0.8	3	0	7	3.00	52.00	28.93
f_dist_h	4x4x1	4x4	20	0.9	1	0	9	6.00	2933.00	586.04
f_dist_h	4x4x1	5x5	5	0.4	10	0	0	0.80	2.80	5.44
f_dist_h	4x4x1	5x5	5	0.6	8	0	2	2.50	58.50	123.62
f_dist_h	4x4x1	5x5	5	0.8	2	0	8	2.50	138.00	165.49
f_dist_h	4x4x1	5x5	5	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	5x5	10	0.4	10	0	0	1.00	3.40	8.06
f_dist_h	4x4x1	5x5	10	0.6	7	0	3	2.86	58.57	116.89
f_dist_h	4x4x1	5x5	10	0.8	1	0	9	3.00	16.00	29.63
f_dist_h	4x4x1	5x5	10	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	5x5	20	0.4	10	0	0	1.00	3.50	9.00
f_dist_h	4x4x1	5x5	20	0.6	6	0	4	2.83	69.50	149.55
f_dist_h	4x4x1	5x5	20	0.8	1	0	9	3.00	16.00	31.61
f_dist_h	4x4x1	5x5	20	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	6x6	5	0.4	9	0	1	0.56	1.56	5.00
f_dist_h	4x4x1	6x6	5	0.6	5	0	5	2.20	30.00	165.75
f_dist_h	4x4x1	6x6	5	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	6x6	5	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	6x6	10	0.4	9	0	1	0.78	3.33	21.93
f_dist_h	4x4x1	6x6	10	0.6	2	0	8	1.50	3.50	22.40
f_dist_h	4x4x1	6x6	10	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	6x6	10	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	6x6	20	0.4	9	0	1	0.78	3.33	23.15
f_dist_h	4x4x1	6x6	20	0.6	1	0	9	1.00	2.00	9.59
f_dist_h	4x4x1	6x6	20	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	6x6	20	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	7x7	5	0.4	9	0	1	1.00	2.78	36.00
f_dist_h	4x4x1	7x7	5	0.6	2	0	8	2.00	13.50	184.63
f_dist_h	4x4x1	7x7	5	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	7x7	5	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	7x7	10	0.4	9	0	1	1.33	5.11	67.76
f_dist_h	4x4x1	7x7	10	0.6	1	0	9	2.00	4.00	60.85
f_dist_h	4x4x1	7x7	10	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	7x7	10	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	7x7	20	0.4	9	0	1	1.33	5.00	69.03
f_dist_h	4x4x1	7x7	20	0.6	1	0	9	2.00	5.00	86.16
f_dist_h	4x4x1	7x7	20	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	7x7	20	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	5	0.4	9	0	1	1.11	5.22	126.94
f_dist_h	4x4x1	8x8	5	0.6	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	5	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	5	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	10	0.4	9	0	1	1.22	5.44	143.37
f_dist_h	4x4x1	8x8	10	0.6	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	10	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	10	0.9	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	20	0.4	8	0	2	1.12	3.88	108.49
f_dist_h	4x4x1	8x8	20	0.6	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	20	0.8	0	0	10	-	-	-
f_dist_h	4x4x1	8x8	20	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	2x2	5	0.4	10	0	0	0.20	1.20	0.03

f_dist_h	5x5x1	2x2	5	0.6	10	0	0	1.30	4.60	0.42
f_dist_h	5x5x1	2x2	5	0.8	7	0	3	4.14	926.29	33.72
f_dist_h	5x5x1	2x2	5	0.9	9	0	1	6.11	7179.78	190.98
f_dist_h	5x5x1	2x2	10	0.4	10	0	0	0.10	1.10	0.01
f_dist_h	5x5x1	2x2	10	0.6	10	0	0	1.80	6.80	0.68
f_dist_h	5x5x1	2x2	10	0.8	8	0	2	5.00	3681.12	133.68
f_dist_h	5x5x1	2x2	10	0.9	7	0	3	6.14	8861.57	193.68
f_dist_h	5x5x1	2x2	20	0.4	10	0	0	0.20	1.20	0.03
f_dist_h	5x5x1	2x2	20	0.6	10	0	0	2.00	17.20	1.73
f_dist_h	5x5x1	2x2	20	0.8	3	0	7	5.33	5102.00	177.37
f_dist_h	5x5x1	2x2	20	0.9	6	0	4	6.33	7161.83	224.12
f_dist_h	5x5x1	3x3	5	0.4	10	0	0	0.70	2.30	1.11
f_dist_h	5x5x1	3x3	5	0.6	9	0	1	3.44	160.33	64.35
f_dist_h	5x5x1	3x3	5	0.8	1	0	9	4.00	591.00	175.26
f_dist_h	5x5x1	3x3	5	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	3x3	10	0.4	10	0	0	0.80	2.70	1.51
f_dist_h	5x5x1	3x3	10	0.6	5	0	5	4.00	351.20	125.68
f_dist_h	5x5x1	3x3	10	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	3x3	10	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	3x3	20	0.4	10	0	0	1.00	6.60	3.85
f_dist_h	5x5x1	3x3	20	0.6	5	0	5	4.40	447.20	176.84
f_dist_h	5x5x1	3x3	20	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	3x3	20	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	5	0.4	10	0	0	1.50	12.50	25.75
f_dist_h	5x5x1	4x4	5	0.6	3	0	7	4.00	192.33	323.61
f_dist_h	5x5x1	4x4	5	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	5	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	10	0.4	10	0	0	1.90	12.30	27.72
f_dist_h	5x5x1	4x4	10	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	10	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	10	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	20	0.4	10	0	0	2.00	27.30	55.73
f_dist_h	5x5x1	4x4	20	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	20	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	4x4	20	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	5	0.4	8	0	2	1.75	9.25	61.75
f_dist_h	5x5x1	5x5	5	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	5	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	5	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	10	0.4	7	0	3	2.14	13.14	91.89
f_dist_h	5x5x1	5x5	10	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	10	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	10	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	20	0.4	4	0	6	2.25	16.00	114.16
f_dist_h	5x5x1	5x5	20	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	20	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	5x5	20	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	5	0.4	2	0	8	2.50	11.50	258.68
f_dist_h	5x5x1	6x6	5	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	5	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	5	0.9	0	0	10	-	-	-

f_dist_h	5x5x1	6x6	10	0.4	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	10	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	10	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	10	0.9	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	20	0.4	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	20	0.6	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	20	0.8	0	0	10	-	-	-
f_dist_h	5x5x1	6x6	20	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	2x2	5	0.4	10	0	0	1.30	8.10	1.50
f_dist_h	6x6x1	2x2	5	0.6	10	0	0	3.80	776.90	84.96
f_dist_h	6x6x1	2x2	5	0.8	1	0	9	5.00	1353.00	99.06
f_dist_h	6x6x1	2x2	5	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	2x2	10	0.4	10	0	0	1.80	9.70	1.97
f_dist_h	6x6x1	2x2	10	0.6	9	0	1	3.89	400.56	49.57
f_dist_h	6x6x1	2x2	10	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	2x2	10	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	2x2	20	0.4	10	0	0	1.80	8.70	1.94
f_dist_h	6x6x1	2x2	20	0.6	7	0	3	4.00	416.71	57.97
f_dist_h	6x6x1	2x2	20	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	2x2	20	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	5	0.4	9	0	1	2.22	58.67	50.02
f_dist_h	6x6x1	3x3	5	0.6	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	5	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	5	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	10	0.4	7	0	3	2.43	20.43	24.82
f_dist_h	6x6x1	3x3	10	0.6	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	10	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	10	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	20	0.4	7	0	3	2.43	20.29	26.04
f_dist_h	6x6x1	3x3	20	0.6	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	20	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	3x3	20	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	5	0.4	3	0	7	2.67	19.33	85.95
f_dist_h	6x6x1	4x4	5	0.6	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	5	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	5	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	10	0.4	2	0	8	3.00	49.50	221.77
f_dist_h	6x6x1	4x4	10	0.6	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	10	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	10	0.9	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	20	0.4	1	0	9	3.00	34.00	156.30
f_dist_h	6x6x1	4x4	20	0.6	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	20	0.8	0	0	10	-	-	-
f_dist_h	6x6x1	4x4	20	0.9	0	0	10	-	-	-
f_h	3x3x1	2x2	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	2x2	5	0.6	10	0	0	0.10	1.10	0.00
f_h	3x3x1	2x2	5	0.8	10	0	0	0.10	1.10	0.00
f_h	3x3x1	2x2	5	0.9	10	0	0	0.60	1.60	0.01
f_h	3x3x1	2x2	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	2x2	10	0.6	10	0	0	0.10	1.10	0.00
f_h	3x3x1	2x2	10	0.8	10	0	0	0.20	1.20	0.00

f_h	3x3x1	2x2	10	0.9	10	0	0	0.50	1.50	0.00
f_h	3x3x1	2x2	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	2x2	20	0.6	10	0	0	0.10	1.10	0.00
f_h	3x3x1	2x2	20	0.8	10	0	0	0.20	1.20	0.00
f_h	3x3x1	2x2	20	0.9	10	0	0	0.60	1.60	0.01
f_h	3x3x1	3x3	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	3x3	5	0.6	10	0	0	0.30	1.30	0.03
f_h	3x3x1	3x3	5	0.8	10	0	0	0.30	1.30	0.02
f_h	3x3x1	3x3	5	0.9	10	0	0	1.30	2.30	0.05
f_h	3x3x1	3x3	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	3x3	10	0.6	10	0	0	0.30	1.30	0.03
f_h	3x3x1	3x3	10	0.8	10	0	0	0.40	1.40	0.03
f_h	3x3x1	3x3	10	0.9	10	0	0	1.40	2.40	0.06
f_h	3x3x1	3x3	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	3x3	20	0.6	10	0	0	0.30	1.30	0.04
f_h	3x3x1	3x3	20	0.8	10	0	0	0.40	1.40	0.03
f_h	3x3x1	3x3	20	0.9	10	0	0	1.40	2.40	0.07
f_h	3x3x1	4x4	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	4x4	5	0.6	10	0	0	0.20	1.20	0.08
f_h	3x3x1	4x4	5	0.8	10	0	0	0.70	1.70	0.18
f_h	3x3x1	4x4	5	0.9	10	0	0	1.50	2.50	0.21
f_h	3x3x1	4x4	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	4x4	10	0.6	10	0	0	0.30	1.30	0.12
f_h	3x3x1	4x4	10	0.8	10	0	0	1.10	2.10	0.29
f_h	3x3x1	4x4	10	0.9	10	0	0	1.80	2.80	0.27
f_h	3x3x1	4x4	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	4x4	20	0.6	10	0	0	0.30	1.30	0.13
f_h	3x3x1	4x4	20	0.8	10	0	0	1.10	2.10	0.32
f_h	3x3x1	4x4	20	0.9	10	0	0	1.90	2.90	0.31
f_h	3x3x1	5x5	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	5x5	5	0.6	10	0	0	0.20	1.20	0.21
f_h	3x3x1	5x5	5	0.8	10	0	0	1.00	2.00	0.69
f_h	3x3x1	5x5	5	0.9	10	0	0	2.30	3.30	0.89
f_h	3x3x1	5x5	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	5x5	10	0.6	10	0	0	0.20	1.20	0.22
f_h	3x3x1	5x5	10	0.8	10	0	0	1.40	2.40	1.03
f_h	3x3x1	5x5	10	0.9	10	0	0	2.90	3.90	1.19
f_h	3x3x1	5x5	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	5x5	20	0.6	10	0	0	0.20	1.20	0.24
f_h	3x3x1	5x5	20	0.8	10	0	0	1.70	2.70	1.35
f_h	3x3x1	5x5	20	0.9	10	0	0	3.20	4.20	1.42
f_h	3x3x1	6x6	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	6x6	5	0.6	10	0	0	0.20	1.20	0.50
f_h	3x3x1	6x6	5	0.8	10	0	0	1.60	2.60	2.69
f_h	3x3x1	6x6	5	0.9	10	0	0	2.40	3.40	2.18
f_h	3x3x1	6x6	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	6x6	10	0.6	10	0	0	0.30	1.30	0.78
f_h	3x3x1	6x6	10	0.8	10	0	0	2.20	3.20	3.85
f_h	3x3x1	6x6	10	0.9	10	0	0	3.40	4.40	3.25
f_h	3x3x1	6x6	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	6x6	20	0.6	10	0	0	0.40	1.40	1.13

f_h	3x3x1	6x6	20	0.8	10	0	0	2.60	3.60	4.90
f_h	3x3x1	6x6	20	0.9	10	0	0	3.80	4.80	3.91
f_h	3x3x1	7x7	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	7x7	5	0.6	10	0	0	0.60	1.60	3.23
f_h	3x3x1	7x7	5	0.8	10	0	0	2.00	3.00	7.04
f_h	3x3x1	7x7	5	0.9	10	0	0	3.10	4.10	6.03
f_h	3x3x1	7x7	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	7x7	10	0.6	10	0	0	0.50	1.50	2.74
f_h	3x3x1	7x7	10	0.8	10	0	0	2.40	3.40	9.09
f_h	3x3x1	7x7	10	0.9	10	0	0	4.80	5.80	9.74
f_h	3x3x1	7x7	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	7x7	20	0.6	10	0	0	0.60	1.60	3.49
f_h	3x3x1	7x7	20	0.8	10	0	0	2.70	3.70	10.52
f_h	3x3x1	7x7	20	0.9	10	0	0	5.50	6.50	12.03
f_h	3x3x1	8x8	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	8x8	5	0.6	10	0	0	0.40	1.40	4.17
f_h	3x3x1	8x8	5	0.8	10	0	0	2.60	3.60	18.02
f_h	3x3x1	8x8	5	0.9	10	0	0	4.40	5.40	16.86
f_h	3x3x1	8x8	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	8x8	10	0.6	10	0	0	0.60	1.60	6.40
f_h	3x3x1	8x8	10	0.8	10	0	0	3.40	4.40	24.21
f_h	3x3x1	8x8	10	0.9	10	0	0	6.30	7.30	24.92
f_h	3x3x1	8x8	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	8x8	20	0.6	10	0	0	0.90	1.90	10.22
f_h	3x3x1	8x8	20	0.8	10	0	0	3.90	4.90	29.38
f_h	3x3x1	8x8	20	0.9	10	0	0	6.80	7.80	28.76
f_h	3x3x1	9x9	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	9x9	5	0.6	10	0	0	0.70	1.70	13.18
f_h	3x3x1	9x9	5	0.8	10	0	0	4.00	5.00	50.61
f_h	3x3x1	9x9	5	0.9	10	0	0	5.20	6.20	36.38
f_h	3x3x1	9x9	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	9x9	10	0.6	10	0	0	1.30	2.30	25.38
f_h	3x3x1	9x9	10	0.8	10	0	0	5.10	6.10	67.04
f_h	3x3x1	9x9	10	0.9	10	0	0	7.80	8.80	56.12
f_h	3x3x1	9x9	20	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	9x9	20	0.6	10	0	0	1.40	2.40	28.67
f_h	3x3x1	9x9	20	0.8	10	0	0	5.80	6.80	79.93
f_h	3x3x1	9x9	20	0.9	10	0	0	8.80	9.80	67.09
f_h	3x3x1	10x10	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	10x10	5	0.6	10	0	0	1.10	2.10	36.22
f_h	3x3x1	10x10	5	0.8	10	0	0	5.20	6.20	116.29
f_h	3x3x1	10x10	5	0.9	10	0	0	7.40	8.40	89.77
f_h	3x3x1	10x10	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	10x10	10	0.6	10	0	0	1.50	2.50	51.13
f_h	3x3x1	10x10	10	0.8	10	0	0	6.30	7.30	145.04
f_h	3x3x1	10x10	10	0.9	10	0	0	10.30	11.30	131.67
f_h	3x3x1	10x10	20	0.4	10	0	0	0.10	1.10	3.44
f_h	3x3x1	10x10	20	0.6	10	0	0	1.90	2.90	67.39
f_h	3x3x1	10x10	20	0.8	10	0	0	7.20	8.20	173.84
f_h	3x3x1	10x10	20	0.9	10	0	0	11.40	12.40	153.08
f_h	3x3x1	11x11	5	0.4	10	0	0	0.00	1.00	0.00

f_h	3x3x1	11x11	5	0.6	10	0	0	0.90	1.90	53.63
f_h	3x3x1	11x11	5	0.8	10	0	0	5.80	6.80	235.01
f_h	3x3x1	11x11	5	0.9	10	0	0	8.30	9.30	187.80
f_h	3x3x1	11x11	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	11x11	10	0.6	10	0	0	1.50	2.50	92.47
f_h	3x3x1	11x11	10	0.8	10	0	0	7.30	8.30	302.31
f_h	3x3x1	11x11	10	0.9	10	0	0	11.70	12.70	270.90
f_h	3x3x1	11x11	20	0.4	10	0	0	0.10	1.10	6.23
f_h	3x3x1	11x11	20	0.6	10	0	0	1.90	2.90	122.52
f_h	3x3x1	11x11	20	0.8	10	0	0	8.40	9.40	361.81
f_h	3x3x1	11x11	20	0.9	10	0	0	12.80	13.80	308.87
f_h	3x3x1	12x12	5	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	12x12	5	0.6	10	0	0	1.20	2.20	107.18
f_h	3x3x1	12x12	5	0.8	9	0	1	6.22	7.22	375.15
f_h	3x3x1	12x12	5	0.9	10	0	0	9.20	10.20	308.79
f_h	3x3x1	12x12	10	0.4	10	0	0	0.00	1.00	0.00
f_h	3x3x1	12x12	10	0.6	10	0	0	1.90	2.90	174.73
f_h	3x3x1	12x12	10	0.8	8	0	2	7.38	8.38	451.74
f_h	3x3x1	12x12	10	0.9	8	0	2	12.12	13.12	417.88
f_h	3x3x1	12x12	20	0.4	10	0	0	0.10	1.10	9.19
f_h	3x3x1	12x12	20	0.6	10	0	0	2.20	3.20	209.58
f_h	3x3x1	12x12	20	0.8	8	0	2	8.50	9.50	551.45
f_h	3x3x1	12x12	20	0.9	8	0	2	13.25	14.25	471.15
f_h	4x4x1	2x2	5	0.4	10	0	0	0.20	1.20	0.01
f_h	4x4x1	2x2	5	0.6	10	0	0	0.10	1.10	0.01
f_h	4x4x1	2x2	5	0.8	10	0	0	1.50	2.50	0.06
f_h	4x4x1	2x2	5	0.9	10	0	0	1.70	2.70	0.04
f_h	4x4x1	2x2	10	0.4	10	0	0	0.20	1.20	0.01
f_h	4x4x1	2x2	10	0.6	10	0	0	0.30	1.30	0.02
f_h	4x4x1	2x2	10	0.8	10	0	0	2.20	3.20	0.09
f_h	4x4x1	2x2	10	0.9	10	0	0	2.20	3.20	0.06
f_h	4x4x1	2x2	20	0.4	10	0	0	0.20	1.20	0.01
f_h	4x4x1	2x2	20	0.6	10	0	0	0.40	1.40	0.03
f_h	4x4x1	2x2	20	0.8	10	0	0	2.20	3.20	0.11
f_h	4x4x1	2x2	20	0.9	10	0	0	2.30	3.30	0.07
f_h	4x4x1	3x3	5	0.4	10	0	0	0.40	1.40	0.14
f_h	4x4x1	3x3	5	0.6	10	0	0	0.90	1.90	0.32
f_h	4x4x1	3x3	5	0.8	10	0	0	2.90	3.90	0.68
f_h	4x4x1	3x3	5	0.9	10	0	0	4.20	5.20	0.56
f_h	4x4x1	3x3	10	0.4	10	0	0	0.40	1.40	0.15
f_h	4x4x1	3x3	10	0.6	10	0	0	1.20	2.20	0.45
f_h	4x4x1	3x3	10	0.8	10	0	0	4.20	5.20	1.05
f_h	4x4x1	3x3	10	0.9	10	0	0	5.60	6.60	0.78
f_h	4x4x1	3x3	20	0.4	10	0	0	0.40	1.40	0.16
f_h	4x4x1	3x3	20	0.6	10	0	0	1.20	2.20	0.49
f_h	4x4x1	3x3	20	0.8	10	0	0	4.30	5.30	1.17
f_h	4x4x1	3x3	20	0.9	10	0	0	5.60	6.60	0.87
f_h	4x4x1	4x4	5	0.4	10	0	0	0.50	1.50	0.64
f_h	4x4x1	4x4	5	0.6	10	0	0	2.10	3.10	2.73
f_h	4x4x1	4x4	5	0.8	10	0	0	4.30	5.30	3.72
f_h	4x4x1	4x4	5	0.9	10	0	0	6.80	7.80	3.48

f_h	4x4x1	4x4	10	0.4	10	0	0	0.60	1.60	0.82
f_h	4x4x1	4x4	10	0.6	10	0	0	2.50	3.50	3.42
f_h	4x4x1	4x4	10	0.8	10	0	0	5.60	6.60	5.10
f_h	4x4x1	4x4	10	0.9	10	0	0	8.50	9.50	4.57
f_h	4x4x1	4x4	20	0.4	10	0	0	0.60	1.60	0.89
f_h	4x4x1	4x4	20	0.6	10	0	0	2.70	3.70	3.97
f_h	4x4x1	4x4	20	0.8	10	0	0	5.80	6.80	5.73
f_h	4x4x1	4x4	20	0.9	10	0	0	9.20	10.20	5.33
f_h	4x4x1	5x5	5	0.4	10	0	0	0.80	1.80	2.98
f_h	4x4x1	5x5	5	0.6	10	0	0	3.10	4.10	11.83
f_h	4x4x1	5x5	5	0.8	10	0	0	7.60	8.60	18.96
f_h	4x4x1	5x5	5	0.9	10	0	0	11.40	12.40	16.33
f_h	4x4x1	5x5	10	0.4	10	0	0	1.00	2.00	3.92
f_h	4x4x1	5x5	10	0.6	10	0	0	3.70	4.70	14.58
f_h	4x4x1	5x5	10	0.8	10	0	0	9.70	10.70	25.28
f_h	4x4x1	5x5	10	0.9	10	0	0	13.50	14.50	20.18
f_h	4x4x1	5x5	20	0.4	10	0	0	1.00	2.00	4.15
f_h	4x4x1	5x5	20	0.6	10	0	0	3.90	4.90	16.39
f_h	4x4x1	5x5	20	0.8	10	0	0	10.10	11.10	28.09
f_h	4x4x1	5x5	20	0.9	10	0	0	15.20	16.20	24.32
f_h	4x4x1	6x6	5	0.4	10	0	0	0.80	1.80	7.43
f_h	4x4x1	6x6	5	0.6	10	0	0	3.90	4.90	36.17
f_h	4x4x1	6x6	5	0.8	10	0	0	10.70	11.70	65.70
f_h	4x4x1	6x6	5	0.9	10	0	0	15.70	16.70	55.06
f_h	4x4x1	6x6	10	0.4	10	0	0	1.00	2.00	9.60
f_h	4x4x1	6x6	10	0.6	10	0	0	4.60	5.60	44.07
f_h	4x4x1	6x6	10	0.8	10	0	0	14.30	15.30	91.02
f_h	4x4x1	6x6	10	0.9	10	0	0	18.90	19.90	68.94
f_h	4x4x1	6x6	20	0.4	10	0	0	1.00	2.00	10.18
f_h	4x4x1	6x6	20	0.6	10	0	0	5.20	6.20	52.22
f_h	4x4x1	6x6	20	0.8	10	0	0	14.90	15.90	101.85
f_h	4x4x1	6x6	20	0.9	10	0	0	21.20	22.20	82.45
f_h	4x4x1	7x7	5	0.4	10	0	0	1.20	2.20	24.52
f_h	4x4x1	7x7	5	0.6	10	0	0	5.90	6.90	122.12
f_h	4x4x1	7x7	5	0.8	10	0	0	13.30	14.30	185.26
f_h	4x4x1	7x7	5	0.9	10	0	0	20.80	21.80	165.49
f_h	4x4x1	7x7	10	0.4	10	0	0	1.60	2.60	33.74
f_h	4x4x1	7x7	10	0.6	10	0	0	7.40	8.40	160.61
f_h	4x4x1	7x7	10	0.8	10	0	0	17.80	18.80	260.24
f_h	4x4x1	7x7	10	0.9	10	0	0	26.50	27.50	218.84
f_h	4x4x1	7x7	20	0.4	10	0	0	1.60	2.60	35.50
f_h	4x4x1	7x7	20	0.6	10	0	0	8.00	9.00	182.33
f_h	4x4x1	7x7	20	0.8	10	0	0	18.50	19.50	281.59
f_h	4x4x1	7x7	20	0.9	10	0	0	29.80	30.80	257.96
f_h	4x4x1	8x8	5	0.4	10	0	0	1.30	2.30	54.03
f_h	4x4x1	8x8	5	0.6	10	0	0	7.90	8.90	332.22
f_h	4x4x1	8x8	5	0.8	10	0	0	16.90	17.90	469.40
f_h	4x4x1	8x8	5	0.9	10	0	0	28.30	29.30	446.51
f_h	4x4x1	8x8	10	0.4	10	0	0	1.50	2.50	64.04
f_h	4x4x1	8x8	10	0.6	10	0	0	9.80	10.80	419.66
f_h	4x4x1	8x8	10	0.8	8	0	2	21.38	22.38	554.46

f_h	4x4x1	8x8	10	0.9	7	0	3	34.14	35.14	551.15
f_h	4x4x1	8x8	20	0.4	10	0	0	1.70	2.70	76.24
f_h	4x4x1	8x8	20	0.6	9	0	1	9.89	10.89	444.86
f_h	4x4x1	8x8	20	0.8	6	0	4	21.33	22.33	586.60
f_h	4x4x1	8x8	20	0.9	4	0	6	37.75	38.75	565.58
f_h	5x5x1	2x2	5	0.4	10	0	0	0.20	1.20	0.03
f_h	5x5x1	2x2	5	0.6	10	0	0	1.30	2.30	0.19
f_h	5x5x1	2x2	5	0.8	10	0	0	4.80	5.80	0.48
f_h	5x5x1	2x2	5	0.9	10	0	0	6.20	7.20	0.57
f_h	5x5x1	2x2	10	0.4	10	0	0	0.10	1.10	0.01
f_h	5x5x1	2x2	10	0.6	10	0	0	1.80	2.80	0.27
f_h	5x5x1	2x2	10	0.8	10	0	0	5.60	6.60	0.60
f_h	5x5x1	2x2	10	0.9	10	0	0	6.50	7.50	0.78
f_h	5x5x1	2x2	20	0.4	10	0	0	0.20	1.20	0.03
f_h	5x5x1	2x2	20	0.6	10	0	0	2.00	3.00	0.33
f_h	5x5x1	2x2	20	0.8	10	0	0	6.40	7.40	0.75
f_h	5x5x1	2x2	20	0.9	10	0	0	6.90	7.90	0.95
f_h	5x5x1	3x3	5	0.4	10	0	0	0.70	1.70	0.60
f_h	5x5x1	3x3	5	0.6	10	0	0	3.80	4.80	3.30
f_h	5x5x1	3x3	5	0.8	10	0	0	9.20	10.20	5.50
f_h	5x5x1	3x3	5	0.9	10	0	0	12.90	13.90	6.15
f_h	5x5x1	3x3	10	0.4	10	0	0	0.80	1.80	0.72
f_h	5x5x1	3x3	10	0.6	10	0	0	4.90	5.90	4.48
f_h	5x5x1	3x3	10	0.8	10	0	0	10.90	11.90	6.84
f_h	5x5x1	3x3	10	0.9	10	0	0	14.80	15.80	7.88
f_h	5x5x1	3x3	20	0.4	10	0	0	1.00	2.00	0.98
f_h	5x5x1	3x3	20	0.6	10	0	0	5.30	6.30	5.25
f_h	5x5x1	3x3	20	0.8	10	0	0	11.90	12.90	8.02
f_h	5x5x1	3x3	20	0.9	10	0	0	16.20	17.20	20.46
f_h	5x5x1	4x4	5	0.4	10	0	0	1.50	2.50	5.02
f_h	5x5x1	4x4	5	0.6	10	0	0	6.00	7.00	20.24
f_h	5x5x1	4x4	5	0.8	10	0	0	16.00	17.00	36.50
f_h	5x5x1	4x4	5	0.9	10	0	0	22.20	23.20	29.12
f_h	5x5x1	4x4	10	0.4	10	0	0	1.90	2.90	6.56
f_h	5x5x1	4x4	10	0.6	10	0	0	8.00	9.00	28.29
f_h	5x5x1	4x4	10	0.8	10	0	0	19.20	20.20	45.85
f_h	5x5x1	4x4	10	0.9	10	0	0	25.60	26.60	55.09
f_h	5x5x1	4x4	20	0.4	10	0	0	2.00	3.00	7.41
f_h	5x5x1	4x4	20	0.6	10	0	0	8.60	9.60	32.43
f_h	5x5x1	4x4	20	0.8	10	0	0	21.10	22.10	53.41
f_h	5x5x1	4x4	20	0.9	10	0	0	28.00	29.00	62.42
f_h	5x5x1	5x5	5	0.4	10	0	0	2.30	3.30	23.22
f_h	5x5x1	5x5	5	0.6	10	0	0	9.40	10.40	95.19
f_h	5x5x1	5x5	5	0.8	10	0	0	25.70	26.70	183.36
f_h	5x5x1	5x5	5	0.9	10	0	0	33.90	34.90	146.27
f_h	5x5x1	5x5	10	0.4	10	0	0	2.90	3.90	29.82
f_h	5x5x1	5x5	10	0.6	10	0	0	11.70	12.70	124.65
f_h	5x5x1	5x5	10	0.8	10	0	0	30.40	31.40	225.36
f_h	5x5x1	5x5	10	0.9	10	0	0	39.60	40.60	253.22
f_h	5x5x1	5x5	20	0.4	10	0	0	3.50	4.50	38.79
f_h	5x5x1	5x5	20	0.6	10	0	0	12.40	13.40	142.71

f_h	5x5x1	5x5	20	0.8	10	0	0	33.90	34.90	263.95
f_h	5x5x1	5x5	20	0.9	10	0	0	43.80	44.80	325.55
f_h	5x5x1	6x6	5	0.4	10	0	0	4.50	5.50	130.93
f_h	5x5x1	6x6	5	0.6	10	0	0	14.40	15.40	422.60
f_h	5x5x1	6x6	5	0.8	6	0	4	34.67	35.67	611.97
f_h	5x5x1	6x6	5	0.9	8	0	2	48.88	49.88	530.41
f_h	5x5x1	6x6	10	0.4	10	0	0	5.00	6.00	150.19
f_h	5x5x1	6x6	10	0.6	8	0	2	16.50	17.50	496.70
f_h	5x5x1	6x6	10	0.8	2	0	8	39.50	40.50	654.96
f_h	5x5x1	6x6	10	0.9	0	0	10	-	-	-
f_h	5x5x1	6x6	20	0.4	10	0	0	5.70	6.70	179.40
f_h	5x5x1	6x6	20	0.6	5	0	5	16.00	17.00	502.44
f_h	5x5x1	6x6	20	0.8	0	0	10	-	-	-
f_h	5x5x1	6x6	20	0.9	0	0	10	-	-	-
f_h	6x6x1	2x2	5	0.4	10	0	0	1.30	2.30	0.38
f_h	6x6x1	2x2	5	0.6	10	0	0	3.80	4.80	1.10
f_h	6x6x1	2x2	5	0.8	10	0	0	9.00	10.00	1.77
f_h	6x6x1	2x2	5	0.9	10	0	0	15.50	16.50	15.44
f_h	6x6x1	2x2	10	0.4	10	0	0	1.80	2.80	0.55
f_h	6x6x1	2x2	10	0.6	10	0	0	4.10	5.10	1.26
f_h	6x6x1	2x2	10	0.8	10	0	0	10.00	11.00	2.08
f_h	6x6x1	2x2	10	0.9	9	0	1	18.11	31.00	49.00
f_h	6x6x1	2x2	20	0.4	10	0	0	1.80	2.80	0.58
f_h	6x6x1	2x2	20	0.6	10	0	0	4.50	5.50	1.48
f_h	6x6x1	2x2	20	0.8	10	0	0	10.70	11.80	2.51
f_h	6x6x1	2x2	20	0.9	10	0	0	18.40	23.50	31.07
f_h	6x6x1	3x3	5	0.4	10	0	0	2.50	3.50	4.35
f_h	6x6x1	3x3	5	0.6	10	0	0	9.10	10.10	16.24
f_h	6x6x1	3x3	5	0.8	10	0	0	21.20	22.20	26.35
f_h	6x6x1	3x3	5	0.9	8	0	2	30.50	31.50	249.35
f_h	6x6x1	3x3	10	0.4	10	0	0	3.30	4.30	6.07
f_h	6x6x1	3x3	10	0.6	10	0	0	10.40	11.40	19.62
f_h	6x6x1	3x3	10	0.8	10	0	0	23.30	24.30	30.86
f_h	6x6x1	3x3	10	0.9	5	0	5	33.80	34.80	191.53
f_h	6x6x1	3x3	20	0.4	10	0	0	3.40	4.40	6.68
f_h	6x6x1	3x3	20	0.6	10	0	0	11.30	12.30	22.72
f_h	6x6x1	3x3	20	0.8	10	0	0	26.00	27.00	35.78
f_h	6x6x1	3x3	20	0.9	3	0	7	35.33	36.33	186.66
f_h	6x6x1	4x4	5	0.4	10	0	0	5.10	6.10	35.94
f_h	6x6x1	4x4	5	0.6	10	0	0	16.80	17.80	121.69
f_h	6x6x1	4x4	5	0.8	10	0	0	37.10	38.10	187.39
f_h	6x6x1	4x4	5	0.9	6	0	4	51.00	52.00	511.48
f_h	6x6x1	4x4	10	0.4	10	0	0	6.40	7.40	47.44
f_h	6x6x1	4x4	10	0.6	10	0	0	19.80	20.80	151.78
f_h	6x6x1	4x4	10	0.8	10	0	0	41.80	42.80	219.40
f_h	6x6x1	4x4	10	0.9	0	0	10	-	-	-
f_h	6x6x1	4x4	20	0.4	10	0	0	6.90	7.90	53.54
f_h	6x6x1	4x4	20	0.6	10	0	0	22.40	23.40	182.27
f_h	6x6x1	4x4	20	0.8	10	0	0	46.20	47.20	253.82
f_h	6x6x1	4x4	20	0.9	0	0	10	-	-	-
f_h_dist	3x3x1	2x2	5	0.4	10	0	0	0.00	1.00	0.00

f_h_dist	3x3x1	2x2	5	0.6	10	0	0	0.10	1.10	0.00
f_h_dist	3x3x1	2x2	5	0.8	10	0	0	0.10	1.10	0.00
f_h_dist	3x3x1	2x2	5	0.9	10	0	0	0.60	1.60	0.01
f_h_dist	3x3x1	2x2	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	2x2	10	0.6	10	0	0	0.10	1.10	0.00
f_h_dist	3x3x1	2x2	10	0.8	10	0	0	0.20	1.20	0.00
f_h_dist	3x3x1	2x2	10	0.9	10	0	0	0.50	1.50	0.00
f_h_dist	3x3x1	2x2	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	2x2	20	0.6	10	0	0	0.10	1.10	0.00
f_h_dist	3x3x1	2x2	20	0.8	10	0	0	0.20	1.20	0.00
f_h_dist	3x3x1	2x2	20	0.9	10	0	0	0.60	1.60	0.01
f_h_dist	3x3x1	3x3	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	3x3	5	0.6	10	0	0	0.30	1.30	0.03
f_h_dist	3x3x1	3x3	5	0.8	10	0	0	0.30	1.30	0.02
f_h_dist	3x3x1	3x3	5	0.9	10	0	0	1.30	2.30	0.05
f_h_dist	3x3x1	3x3	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	3x3	10	0.6	10	0	0	0.30	1.30	0.04
f_h_dist	3x3x1	3x3	10	0.8	10	0	0	0.40	1.40	0.03
f_h_dist	3x3x1	3x3	10	0.9	10	0	0	1.40	2.40	0.06
f_h_dist	3x3x1	3x3	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	3x3	20	0.6	10	0	0	0.30	1.30	0.04
f_h_dist	3x3x1	3x3	20	0.8	10	0	0	0.40	1.40	0.03
f_h_dist	3x3x1	3x3	20	0.9	10	0	0	1.40	2.40	0.07
f_h_dist	3x3x1	4x4	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	4x4	5	0.6	10	0	0	0.20	1.20	0.08
f_h_dist	3x3x1	4x4	5	0.8	10	0	0	0.70	1.70	0.18
f_h_dist	3x3x1	4x4	5	0.9	10	0	0	1.50	2.50	0.21
f_h_dist	3x3x1	4x4	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	4x4	10	0.6	10	0	0	0.30	1.30	0.12
f_h_dist	3x3x1	4x4	10	0.8	10	0	0	1.10	2.10	0.30
f_h_dist	3x3x1	4x4	10	0.9	10	0	0	1.80	2.80	0.28
f_h_dist	3x3x1	4x4	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	4x4	20	0.6	10	0	0	0.30	1.30	0.13
f_h_dist	3x3x1	4x4	20	0.8	10	0	0	1.10	2.10	0.33
f_h_dist	3x3x1	4x4	20	0.9	10	0	0	1.90	2.90	0.32
f_h_dist	3x3x1	5x5	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	5x5	5	0.6	10	0	0	0.20	1.20	0.21
f_h_dist	3x3x1	5x5	5	0.8	10	0	0	1.00	2.00	0.70
f_h_dist	3x3x1	5x5	5	0.9	10	0	0	2.30	3.30	0.89
f_h_dist	3x3x1	5x5	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	5x5	10	0.6	10	0	0	0.20	1.20	0.22
f_h_dist	3x3x1	5x5	10	0.8	10	0	0	1.40	2.40	1.03
f_h_dist	3x3x1	5x5	10	0.9	10	0	0	2.90	3.90	1.19
f_h_dist	3x3x1	5x5	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	5x5	20	0.6	10	0	0	0.20	1.20	0.24
f_h_dist	3x3x1	5x5	20	0.8	10	0	0	1.70	2.70	1.36
f_h_dist	3x3x1	5x5	20	0.9	10	0	0	3.20	4.20	1.43
f_h_dist	3x3x1	6x6	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	6x6	5	0.6	10	0	0	0.20	1.20	0.51
f_h_dist	3x3x1	6x6	5	0.8	10	0	0	1.60	2.60	2.70
f_h_dist	3x3x1	6x6	5	0.9	10	0	0	2.40	3.40	2.18

f_h_dist	3x3x1	6x6	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	6x6	10	0.6	10	0	0	0.30	1.30	0.77
f_h_dist	3x3x1	6x6	10	0.8	10	0	0	2.20	3.20	3.87
f_h_dist	3x3x1	6x6	10	0.9	10	0	0	3.40	4.40	3.26
f_h_dist	3x3x1	6x6	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	6x6	20	0.6	10	0	0	0.40	1.40	1.12
f_h_dist	3x3x1	6x6	20	0.8	10	0	0	2.60	3.60	4.90
f_h_dist	3x3x1	6x6	20	0.9	10	0	0	3.80	4.80	3.91
f_h_dist	3x3x1	7x7	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	7x7	5	0.6	10	0	0	0.60	1.60	3.22
f_h_dist	3x3x1	7x7	5	0.8	10	0	0	2.00	3.00	7.01
f_h_dist	3x3x1	7x7	5	0.9	10	0	0	3.10	4.10	5.98
f_h_dist	3x3x1	7x7	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	7x7	10	0.6	10	0	0	0.50	1.50	2.76
f_h_dist	3x3x1	7x7	10	0.8	10	0	0	2.40	3.40	8.83
f_h_dist	3x3x1	7x7	10	0.9	10	0	0	4.80	5.80	9.77
f_h_dist	3x3x1	7x7	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	7x7	20	0.6	10	0	0	0.60	1.60	3.55
f_h_dist	3x3x1	7x7	20	0.8	10	0	0	2.70	3.70	10.56
f_h_dist	3x3x1	7x7	20	0.9	10	0	0	5.50	6.50	11.93
f_h_dist	3x3x1	8x8	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	8x8	5	0.6	10	0	0	0.40	1.40	4.22
f_h_dist	3x3x1	8x8	5	0.8	10	0	0	2.60	3.60	17.98
f_h_dist	3x3x1	8x8	5	0.9	10	0	0	4.40	5.40	16.76
f_h_dist	3x3x1	8x8	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	8x8	10	0.6	10	0	0	0.60	1.60	6.44
f_h_dist	3x3x1	8x8	10	0.8	10	0	0	3.40	4.40	24.23
f_h_dist	3x3x1	8x8	10	0.9	10	0	0	6.30	7.30	25.13
f_h_dist	3x3x1	8x8	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	8x8	20	0.6	10	0	0	0.90	1.90	10.14
f_h_dist	3x3x1	8x8	20	0.8	10	0	0	3.90	4.90	29.48
f_h_dist	3x3x1	8x8	20	0.9	10	0	0	6.80	7.80	28.80
f_h_dist	3x3x1	9x9	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	9x9	5	0.6	10	0	0	0.70	1.70	13.29
f_h_dist	3x3x1	9x9	5	0.8	10	0	0	4.00	5.00	50.68
f_h_dist	3x3x1	9x9	5	0.9	10	0	0	5.20	6.20	36.30
f_h_dist	3x3x1	9x9	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	9x9	10	0.6	10	0	0	1.30	2.30	25.17
f_h_dist	3x3x1	9x9	10	0.8	10	0	0	5.10	6.10	66.75
f_h_dist	3x3x1	9x9	10	0.9	10	0	0	7.80	8.80	55.80
f_h_dist	3x3x1	9x9	20	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	9x9	20	0.6	10	0	0	1.40	2.40	28.75
f_h_dist	3x3x1	9x9	20	0.8	10	0	0	5.80	6.80	80.46
f_h_dist	3x3x1	9x9	20	0.9	10	0	0	8.80	9.80	67.01
f_h_dist	3x3x1	10x10	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	10x10	5	0.6	10	0	0	1.10	2.10	35.95
f_h_dist	3x3x1	10x10	5	0.8	10	0	0	5.20	6.20	116.22
f_h_dist	3x3x1	10x10	5	0.9	10	0	0	7.40	8.40	89.90
f_h_dist	3x3x1	10x10	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	10x10	10	0.6	10	0	0	1.50	2.50	51.16
f_h_dist	3x3x1	10x10	10	0.8	10	0	0	6.30	7.30	146.28

f_h_dist	3x3x1	10x10	10	0.9	10	0	0	10.30	11.30	132.39
f_h_dist	3x3x1	10x10	20	0.4	10	0	0	0.10	1.10	3.47
f_h_dist	3x3x1	10x10	20	0.6	10	0	0	1.90	2.90	67.56
f_h_dist	3x3x1	10x10	20	0.8	10	0	0	7.20	8.20	175.04
f_h_dist	3x3x1	10x10	20	0.9	10	0	0	11.40	12.40	155.00
f_h_dist	3x3x1	11x11	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	11x11	5	0.6	10	0	0	0.90	1.90	54.23
f_h_dist	3x3x1	11x11	5	0.8	10	0	0	5.80	6.80	235.67
f_h_dist	3x3x1	11x11	5	0.9	10	0	0	8.30	9.30	189.95
f_h_dist	3x3x1	11x11	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	11x11	10	0.6	10	0	0	1.50	2.50	93.28
f_h_dist	3x3x1	11x11	10	0.8	10	0	0	7.30	8.30	303.25
f_h_dist	3x3x1	11x11	10	0.9	10	0	0	11.70	12.70	274.43
f_h_dist	3x3x1	11x11	20	0.4	10	0	0	0.10	1.10	6.21
f_h_dist	3x3x1	11x11	20	0.6	10	0	0	1.90	2.90	122.53
f_h_dist	3x3x1	11x11	20	0.8	10	0	0	8.40	9.40	363.73
f_h_dist	3x3x1	11x11	20	0.9	10	0	0	12.80	13.80	310.23
f_h_dist	3x3x1	12x12	5	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	12x12	5	0.6	10	0	0	1.20	2.20	107.20
f_h_dist	3x3x1	12x12	5	0.8	9	0	1	6.22	7.22	365.03
f_h_dist	3x3x1	12x12	5	0.9	10	0	0	9.20	10.20	311.27
f_h_dist	3x3x1	12x12	10	0.4	10	0	0	0.00	1.00	0.00
f_h_dist	3x3x1	12x12	10	0.6	10	0	0	1.90	2.90	175.53
f_h_dist	3x3x1	12x12	10	0.8	8	0	2	7.38	8.38	460.42
f_h_dist	3x3x1	12x12	10	0.9	8	0	2	12.12	13.12	419.48
f_h_dist	3x3x1	12x12	20	0.4	10	0	0	0.10	1.10	9.15
f_h_dist	3x3x1	12x12	20	0.6	10	0	0	2.20	3.20	211.83
f_h_dist	3x3x1	12x12	20	0.8	8	0	2	8.50	9.50	536.16
f_h_dist	3x3x1	12x12	20	0.9	8	0	2	13.25	14.25	485.62
f_h_dist	4x4x1	2x2	5	0.4	10	0	0	0.20	1.20	0.01
f_h_dist	4x4x1	2x2	5	0.6	10	0	0	0.10	1.10	0.01
f_h_dist	4x4x1	2x2	5	0.8	10	0	0	1.50	2.50	0.06
f_h_dist	4x4x1	2x2	5	0.9	10	0	0	1.70	2.70	0.04
f_h_dist	4x4x1	2x2	10	0.4	10	0	0	0.20	1.20	0.01
f_h_dist	4x4x1	2x2	10	0.6	10	0	0	0.30	1.30	0.02
f_h_dist	4x4x1	2x2	10	0.8	10	0	0	2.20	3.20	0.09
f_h_dist	4x4x1	2x2	10	0.9	10	0	0	2.20	3.20	0.06
f_h_dist	4x4x1	2x2	20	0.4	10	0	0	0.20	1.20	0.01
f_h_dist	4x4x1	2x2	20	0.6	10	0	0	0.40	1.40	0.03
f_h_dist	4x4x1	2x2	20	0.8	10	0	0	2.20	3.20	0.10
f_h_dist	4x4x1	2x2	20	0.9	10	0	0	2.30	3.30	0.07
f_h_dist	4x4x1	3x3	5	0.4	10	0	0	0.40	1.40	0.14
f_h_dist	4x4x1	3x3	5	0.6	10	0	0	0.90	1.90	0.32
f_h_dist	4x4x1	3x3	5	0.8	10	0	0	2.90	3.90	0.69
f_h_dist	4x4x1	3x3	5	0.9	10	0	0	4.20	5.20	0.56
f_h_dist	4x4x1	3x3	10	0.4	10	0	0	0.40	1.40	0.15
f_h_dist	4x4x1	3x3	10	0.6	10	0	0	1.20	2.20	0.45
f_h_dist	4x4x1	3x3	10	0.8	10	0	0	4.20	5.20	1.05
f_h_dist	4x4x1	3x3	10	0.9	10	0	0	5.60	6.60	0.79
f_h_dist	4x4x1	3x3	20	0.4	10	0	0	0.40	1.40	0.16
f_h_dist	4x4x1	3x3	20	0.6	10	0	0	1.20	2.20	0.49

f_h_dist	4x4x1	3x3	20	0.8	10	0	0	4.30	5.30	1.17
f_h_dist	4x4x1	3x3	20	0.9	10	0	0	5.60	6.60	0.86
f_h_dist	4x4x1	4x4	5	0.4	10	0	0	0.50	1.50	0.64
f_h_dist	4x4x1	4x4	5	0.6	10	0	0	2.10	3.10	2.73
f_h_dist	4x4x1	4x4	5	0.8	10	0	0	4.30	5.30	3.71
f_h_dist	4x4x1	4x4	5	0.9	10	0	0	6.80	7.80	3.47
f_h_dist	4x4x1	4x4	10	0.4	10	0	0	0.60	1.60	0.82
f_h_dist	4x4x1	4x4	10	0.6	10	0	0	2.50	3.50	3.41
f_h_dist	4x4x1	4x4	10	0.8	10	0	0	5.60	6.60	5.11
f_h_dist	4x4x1	4x4	10	0.9	10	0	0	8.50	9.50	4.58
f_h_dist	4x4x1	4x4	20	0.4	10	0	0	0.60	1.60	0.88
f_h_dist	4x4x1	4x4	20	0.6	10	0	0	2.70	3.70	3.99
f_h_dist	4x4x1	4x4	20	0.8	10	0	0	5.80	6.80	5.71
f_h_dist	4x4x1	4x4	20	0.9	10	0	0	9.20	10.20	5.33
f_h_dist	4x4x1	5x5	5	0.4	10	0	0	0.80	1.80	2.99
f_h_dist	4x4x1	5x5	5	0.6	10	0	0	3.10	4.10	11.80
f_h_dist	4x4x1	5x5	5	0.8	10	0	0	7.60	8.60	18.90
f_h_dist	4x4x1	5x5	5	0.9	10	0	0	11.40	12.40	16.20
f_h_dist	4x4x1	5x5	10	0.4	10	0	0	1.00	2.00	3.92
f_h_dist	4x4x1	5x5	10	0.6	10	0	0	3.70	4.70	14.59
f_h_dist	4x4x1	5x5	10	0.8	10	0	0	9.70	10.70	25.21
f_h_dist	4x4x1	5x5	10	0.9	10	0	0	13.50	14.50	20.16
f_h_dist	4x4x1	5x5	20	0.4	10	0	0	1.00	2.00	4.17
f_h_dist	4x4x1	5x5	20	0.6	10	0	0	3.90	4.90	16.28
f_h_dist	4x4x1	5x5	20	0.8	10	0	0	10.10	11.10	28.12
f_h_dist	4x4x1	5x5	20	0.9	10	0	0	15.20	16.20	24.53
f_h_dist	4x4x1	6x6	5	0.4	10	0	0	0.80	1.80	7.38
f_h_dist	4x4x1	6x6	5	0.6	10	0	0	3.90	4.90	36.25
f_h_dist	4x4x1	6x6	5	0.8	10	0	0	10.70	11.70	65.62
f_h_dist	4x4x1	6x6	5	0.9	10	0	0	15.70	16.70	55.00
f_h_dist	4x4x1	6x6	10	0.4	10	0	0	1.00	2.00	9.68
f_h_dist	4x4x1	6x6	10	0.6	10	0	0	4.60	5.60	44.12
f_h_dist	4x4x1	6x6	10	0.8	10	0	0	14.30	15.30	91.58
f_h_dist	4x4x1	6x6	10	0.9	10	0	0	18.90	19.90	69.17
f_h_dist	4x4x1	6x6	20	0.4	10	0	0	1.00	2.00	10.22
f_h_dist	4x4x1	6x6	20	0.6	10	0	0	5.20	6.20	52.86
f_h_dist	4x4x1	6x6	20	0.8	10	0	0	14.90	15.90	101.95
f_h_dist	4x4x1	6x6	20	0.9	10	0	0	21.20	22.20	82.30
f_h_dist	4x4x1	7x7	5	0.4	10	0	0	1.20	2.20	24.49
f_h_dist	4x4x1	7x7	5	0.6	10	0	0	5.90	6.90	123.74
f_h_dist	4x4x1	7x7	5	0.8	10	0	0	13.30	14.30	186.83
f_h_dist	4x4x1	7x7	5	0.9	10	0	0	20.80	21.80	166.83
f_h_dist	4x4x1	7x7	10	0.4	10	0	0	1.60	2.60	33.71
f_h_dist	4x4x1	7x7	10	0.6	10	0	0	7.40	8.40	161.72
f_h_dist	4x4x1	7x7	10	0.8	10	0	0	17.80	18.80	261.54
f_h_dist	4x4x1	7x7	10	0.9	10	0	0	26.50	27.50	219.11
f_h_dist	4x4x1	7x7	20	0.4	10	0	0	1.60	2.60	35.68
f_h_dist	4x4x1	7x7	20	0.6	10	0	0	8.00	9.00	183.04
f_h_dist	4x4x1	7x7	20	0.8	10	0	0	18.50	19.50	282.47
f_h_dist	4x4x1	7x7	20	0.9	10	0	0	29.80	30.80	258.65
f_h_dist	4x4x1	8x8	5	0.4	10	0	0	1.30	2.30	54.01

f_h_dist	4x4x1	8x8	5	0.6	10	0	0	7.90	8.90	333.76
f_h_dist	4x4x1	8x8	5	0.8	10	0	0	16.90	17.90	472.33
f_h_dist	4x4x1	8x8	5	0.9	10	0	0	28.30	29.30	449.64
f_h_dist	4x4x1	8x8	10	0.4	10	0	0	1.50	2.50	64.44
f_h_dist	4x4x1	8x8	10	0.6	10	0	0	9.80	10.80	422.18
f_h_dist	4x4x1	8x8	10	0.8	8	0	2	21.38	22.38	560.60
f_h_dist	4x4x1	8x8	10	0.9	8	0	2	34.62	35.62	561.34
f_h_dist	4x4x1	8x8	20	0.4	10	0	0	1.70	2.70	76.24
f_h_dist	4x4x1	8x8	20	0.6	9	0	1	9.89	10.89	450.53
f_h_dist	4x4x1	8x8	20	0.8	6	0	4	21.33	22.33	586.41
f_h_dist	4x4x1	8x8	20	0.9	4	0	6	37.50	38.50	605.63
f_h_dist	5x5x1	2x2	5	0.4	10	0	0	0.20	1.20	0.03
f_h_dist	5x5x1	2x2	5	0.6	10	0	0	1.30	2.30	0.19
f_h_dist	5x5x1	2x2	5	0.8	10	0	0	4.80	5.80	0.48
f_h_dist	5x5x1	2x2	5	0.9	10	0	0	6.20	7.20	0.57
f_h_dist	5x5x1	2x2	10	0.4	10	0	0	0.10	1.10	0.01
f_h_dist	5x5x1	2x2	10	0.6	10	0	0	1.80	2.80	0.27
f_h_dist	5x5x1	2x2	10	0.8	10	0	0	5.60	6.60	0.59
f_h_dist	5x5x1	2x2	10	0.9	10	0	0	6.50	7.50	0.68
f_h_dist	5x5x1	2x2	20	0.4	10	0	0	0.20	1.20	0.03
f_h_dist	5x5x1	2x2	20	0.6	10	0	0	2.00	3.00	0.34
f_h_dist	5x5x1	2x2	20	0.8	10	0	0	6.40	7.40	0.75
f_h_dist	5x5x1	2x2	20	0.9	10	0	0	6.90	7.90	1.05
f_h_dist	5x5x1	3x3	5	0.4	10	0	0	0.70	1.70	0.60
f_h_dist	5x5x1	3x3	5	0.6	10	0	0	3.80	4.80	3.31
f_h_dist	5x5x1	3x3	5	0.8	10	0	0	9.20	10.20	5.41
f_h_dist	5x5x1	3x3	5	0.9	10	0	0	12.90	13.90	6.30
f_h_dist	5x5x1	3x3	10	0.4	10	0	0	0.80	1.80	0.72
f_h_dist	5x5x1	3x3	10	0.6	10	0	0	4.90	5.90	4.49
f_h_dist	5x5x1	3x3	10	0.8	10	0	0	10.90	11.90	6.78
f_h_dist	5x5x1	3x3	10	0.9	10	0	0	14.80	15.80	8.09
f_h_dist	5x5x1	3x3	20	0.4	10	0	0	1.00	2.00	0.98
f_h_dist	5x5x1	3x3	20	0.6	10	0	0	5.30	6.30	5.24
f_h_dist	5x5x1	3x3	20	0.8	10	0	0	11.90	12.90	8.06
f_h_dist	5x5x1	3x3	20	0.9	10	0	0	16.30	17.30	36.04
f_h_dist	5x5x1	4x4	5	0.4	10	0	0	1.50	2.50	5.00
f_h_dist	5x5x1	4x4	5	0.6	10	0	0	6.00	7.00	20.11
f_h_dist	5x5x1	4x4	5	0.8	10	0	0	16.00	17.00	36.63
f_h_dist	5x5x1	4x4	5	0.9	10	0	0	22.20	23.20	33.62
f_h_dist	5x5x1	4x4	10	0.4	10	0	0	1.90	2.90	6.54
f_h_dist	5x5x1	4x4	10	0.6	10	0	0	8.00	9.00	28.27
f_h_dist	5x5x1	4x4	10	0.8	10	0	0	19.20	20.20	45.79
f_h_dist	5x5x1	4x4	10	0.9	10	0	0	25.60	26.60	55.79
f_h_dist	5x5x1	4x4	20	0.4	10	0	0	2.00	3.00	7.43
f_h_dist	5x5x1	4x4	20	0.6	10	0	0	8.60	9.60	32.35
f_h_dist	5x5x1	4x4	20	0.8	10	0	0	21.10	22.10	53.85
f_h_dist	5x5x1	4x4	20	0.9	10	0	0	28.10	29.10	80.28
f_h_dist	5x5x1	5x5	5	0.4	10	0	0	2.30	3.30	23.39
f_h_dist	5x5x1	5x5	5	0.6	10	0	0	9.40	10.40	95.39
f_h_dist	5x5x1	5x5	5	0.8	10	0	0	25.70	26.70	182.71
f_h_dist	5x5x1	5x5	5	0.9	10	0	0	33.90	34.90	148.19

f_h_dist	5x5x1	5x5	10	0.4	10	0	0	2.90	3.90	30.12
f_h_dist	5x5x1	5x5	10	0.6	10	0	0	11.70	12.70	126.51
f_h_dist	5x5x1	5x5	10	0.8	10	0	0	30.40	31.40	225.12
f_h_dist	5x5x1	5x5	10	0.9	10	0	0	39.60	40.60	257.65
f_h_dist	5x5x1	5x5	20	0.4	10	0	0	3.50	4.50	38.58
f_h_dist	5x5x1	5x5	20	0.6	10	0	0	12.40	13.40	143.24
f_h_dist	5x5x1	5x5	20	0.8	10	0	0	33.90	34.90	264.60
f_h_dist	5x5x1	5x5	20	0.9	10	0	0	43.80	45.10	342.60
f_h_dist	5x5x1	6x6	5	0.4	10	0	0	4.50	5.50	131.17
f_h_dist	5x5x1	6x6	5	0.6	10	0	0	14.40	15.40	424.50
f_h_dist	5x5x1	6x6	5	0.8	6	0	4	34.67	35.67	591.43
f_h_dist	5x5x1	6x6	5	0.9	8	0	2	48.38	49.38	531.33
f_h_dist	5x5x1	6x6	10	0.4	10	0	0	5.00	6.00	150.87
f_h_dist	5x5x1	6x6	10	0.6	8	0	2	16.50	17.50	499.23
f_h_dist	5x5x1	6x6	10	0.8	2	0	8	39.50	40.50	668.27
f_h_dist	5x5x1	6x6	10	0.9	0	0	10	-	-	-
f_h_dist	5x5x1	6x6	20	0.4	10	0	0	5.70	6.70	179.23
f_h_dist	5x5x1	6x6	20	0.6	5	0	5	16.00	17.00	501.43
f_h_dist	5x5x1	6x6	20	0.8	0	0	10	-	-	-
f_h_dist	5x5x1	6x6	20	0.9	0	0	10	-	-	-
f_h_dist	6x6x1	2x2	5	0.4	10	0	0	1.30	2.30	0.38
f_h_dist	6x6x1	2x2	5	0.6	10	0	0	3.80	4.80	1.10
f_h_dist	6x6x1	2x2	5	0.8	10	0	0	9.00	10.00	1.77
f_h_dist	6x6x1	2x2	5	0.9	10	0	0	15.50	16.50	11.16
f_h_dist	6x6x1	2x2	10	0.4	10	0	0	1.80	2.80	0.56
f_h_dist	6x6x1	2x2	10	0.6	10	0	0	4.10	5.10	1.25
f_h_dist	6x6x1	2x2	10	0.8	10	0	0	10.00	11.00	2.07
f_h_dist	6x6x1	2x2	10	0.9	9	0	1	18.11	19.11	22.26
f_h_dist	6x6x1	2x2	20	0.4	10	0	0	1.80	2.80	0.58
f_h_dist	6x6x1	2x2	20	0.6	10	0	0	4.50	5.50	1.48
f_h_dist	6x6x1	2x2	20	0.8	10	0	0	10.70	11.70	2.57
f_h_dist	6x6x1	2x2	20	0.9	10	0	0	18.40	20.60	27.12
f_h_dist	6x6x1	3x3	5	0.4	10	0	0	2.50	3.50	4.40
f_h_dist	6x6x1	3x3	5	0.6	10	0	0	9.10	10.10	16.30
f_h_dist	6x6x1	3x3	5	0.8	10	0	0	21.20	22.20	25.94
f_h_dist	6x6x1	3x3	5	0.9	7	0	3	29.86	30.86	222.21
f_h_dist	6x6x1	3x3	10	0.4	10	0	0	3.30	4.30	6.10
f_h_dist	6x6x1	3x3	10	0.6	10	0	0	10.40	11.40	19.76
f_h_dist	6x6x1	3x3	10	0.8	10	0	0	23.30	24.30	31.26
f_h_dist	6x6x1	3x3	10	0.9	5	0	5	33.80	34.80	156.07
f_h_dist	6x6x1	3x3	20	0.4	10	0	0	3.40	4.40	6.68
f_h_dist	6x6x1	3x3	20	0.6	10	0	0	11.30	12.30	22.77
f_h_dist	6x6x1	3x3	20	0.8	10	0	0	26.00	27.00	35.79
f_h_dist	6x6x1	3x3	20	0.9	4	0	6	36.25	37.25	290.99
f_h_dist	6x6x1	4x4	5	0.4	10	0	0	5.10	6.10	35.83
f_h_dist	6x6x1	4x4	5	0.6	10	0	0	16.80	17.80	122.47
f_h_dist	6x6x1	4x4	5	0.8	10	0	0	37.10	38.10	188.11
f_h_dist	6x6x1	4x4	5	0.9	6	0	4	51.00	52.00	503.87
f_h_dist	6x6x1	4x4	10	0.4	10	0	0	6.40	7.40	47.25
f_h_dist	6x6x1	4x4	10	0.6	10	0	0	19.80	20.80	152.38
f_h_dist	6x6x1	4x4	10	0.8	10	0	0	41.80	42.80	220.15

f_h_dist	6x6x1	4x4	10	0.9	0	0	10	-	-	-
f_h_dist	6x6x1	4x4	20	0.4	10	0	0	6.90	7.90	53.70
f_h_dist	6x6x1	4x4	20	0.6	10	0	0	22.40	23.40	183.63
f_h_dist	6x6x1	4x4	20	0.8	10	0	0	46.20	47.20	255.39
f_h_dist	6x6x1	4x4	20	0.9	0	0	10	-	-	-

Table A.3 Results for experiment *A* for the MUPMP with a single tier rounded to the second decimal

Access variant	Bay size (I x J x T)	Warehouse Size (L x W)	Fill level	No. solved	No. infeasible	No. time-out	Mean moves	Mean visited nodes	Mean runtime
four	4x4x1	2x2	0.4	10	0	0	0.20	1.20	0.01
four	4x4x1	2x2	0.6	10	0	0	0.10	1.10	0.01
four	4x4x1	2x2	0.8	10	0	0	1.50	2.50	0.06
four	4x4x1	2x2	0.9	10	0	0	1.70	2.70	0.04
four	6x6x1	2x2	0.4	10	0	0	1.20	2.20	0.35
four	6x6x1	2x2	0.6	10	0	0	3.30	4.30	0.93
four	6x6x1	2x2	0.8	10	0	0	9.00	10.00	1.73
four	6x6x1	2x2	0.9	10	0	0	15.00	16.80	9.80
opposite	4x4x1	2x2	0.4	10	0	0	1.00	2.00	0.05
opposite	4x4x1	2x2	0.6	10	0	0	2.00	3.00	0.11
opposite	4x4x1	2x2	0.8	10	0	0	4.70	5.70	0.18
opposite	4x4x1	2x2	0.9	10	0	0	7.90	10.00	1.00
opposite	6x6x1	2x2	0.4	10	0	0	4.80	5.80	0.92
opposite	6x6x1	2x2	0.6	10	0	0	11.80	12.80	2.54
opposite	6x6x1	2x2	0.8	8	0	2	26.62	52.12	73.15
opposite	6x6x1	2x2	0.9	1	0	9	31.00	47.00	30.95
single	4x4x1	2x2	0.4	10	0	0	2.40	3.40	0.13
single	4x4x1	2x2	0.6	10	0	0	4.30	5.30	0.25
single	4x4x1	2x2	0.8	10	0	0	8.30	9.30	0.59
single	4x4x1	2x2	0.9	10	0	0	12.80	13.80	1.60
single	6x6x1	2x2	0.4	10	0	0	9.60	10.60	2.06
single	6x6x1	2x2	0.6	9	0	1	19.33	20.33	5.27
single	6x6x1	2x2	0.8	3	0	7	40.00	41.00	39.33
single	6x6x1	2x2	0.9	1	0	9	47.00	48.00	37.53

Table A.4 Results for experiment *B* for the MUPMP with a single tier rounded to the second decimal. The tie breaking variant is *f_h_dist* and the number of priority groups is set to 5.

Bay size (I x J x T)	Warehouse Size (L x W)	Fill level	No. solved	No. infeasible	No. time-out	Mean moves	Mean visited nodes	Mean runtime
3x3x2	2x2	0.4	10	0	0	2.90	3.90	0.18
3x3x2	2x2	0.6	10	0	0	5.60	6.60	0.34
3x3x2	2x2	0.8	10	0	0	11.90	13.00	1.43

3x3x2	2x2	0.9	10	0	0	15.50	178.90	39.40
3x3x2	3x3	0.4	10	0	0	7.00	8.00	2.44
3x3x2	3x3	0.6	10	0	0	13.50	14.50	4.95
3x3x2	3x3	0.8	9	0	1	26.89	38.89	90.28
3x3x2	3x3	0.9	4	0	6	33.00	63.50	121.84
3x3x2	4x4	0.4	10	0	0	11.60	12.60	15.54
3x3x2	4x4	0.6	10	0	0	24.60	25.60	33.25
3x3x2	4x4	0.8	7	0	3	46.00	47.00	248.74
3x3x2	4x4	0.9	0	0	10	-	-	-
3x3x2	5x5	0.4	10	0	0	17.50	18.50	68.83
3x3x2	5x5	0.6	10	0	0	37.10	38.10	148.77
3x3x2	5x5	0.8	3	0	7	67.33	68.33	479.21
3x3x2	5x5	0.9	0	0	10	-	-	-
4x4x2	2x2	0.4	10	0	0	6.70	7.70	1.15
4x4x2	2x2	0.6	10	0	0	15.60	16.60	2.77
4x4x2	2x2	0.8	8	0	2	23.75	24.75	18.47
4x4x2	2x2	0.9	1	0	9	34.00	750.00	515.11
4x4x2	3x3	0.4	10	0	0	15.30	16.30	16.71
4x4x2	3x3	0.6	10	0	0	32.90	33.90	33.87
4x4x2	3x3	0.8	1	0	9	47.00	48.00	509.57
4x4x2	3x3	0.9	0	0	10	-	-	-
5x5x2	2x2	0.4	10	0	0	12.90	13.90	5.12
5x5x2	2x2	0.6	10	0	0	27.10	49.80	28.46
5x5x2	2x2	0.8	1	0	9	45.00	46.00	90.27
5x5x2	2x2	0.9	0	0	10	-	-	-

Table A.5 Results for experiment *C* for the MUPMP with two tiers rounded to the second decimal. The tie breaking variant is *f_h_dist* and the number of priority groups is set to 5.

Bay size (I x J x T)	Access variant	Fill level	No. solved	No. infeasible	No. timeout	Mean moves	Mean visited nodes	Mean runtime
3x3x2	corner	40	20	0	0	1.70	2.70	0.02
3x3x2	corner	60	20	0	0	5.50	19.90	0.20
3x3x2	corner	80	11	0	9	6.82	494.55	4.61
3x3x2	four	40	20	0	0	0.90	1.90	0.02
3x3x2	four	60	20	0	0	1.95	2.95	0.05
3x3x2	four	80	20	0	0	3.15	4.15	0.07
3x3x2	opposite	40	20	0	0	1.25	2.25	0.02
3x3x2	opposite	60	20	0	0	3.00	4.20	0.06
3x3x2	opposite	80	20	0	0	5.75	86.20	1.16
3x3x2	single	40	20	0	0	4.40	9.75	0.03
3x3x2	single	60	20	0	0	10.30	311.05	0.68
3x3x2	single	80	0	20	0	-	-	-
3x3x2	three	40	20	0	0	1.50	2.50	0.03
3x3x2	three	60	20	0	0	2.30	3.30	0.05
3x3x2	three	80	20	0	0	4.75	32.85	0.53
4x4x2	corner	40	20	0	0	5.35	6.45	0.17
4x4x2	corner	60	20	0	0	8.95	141.20	5.49
4x4x2	corner	80	6	0	14	13.17	14034.83	461.68
4x4x2	four	40	20	0	0	2.55	3.55	0.22
4x4x2	four	60	20	0	0	4.80	5.80	0.43
4x4x2	four	80	20	0	0	6.90	61.50	4.13
4x4x2	opposite	40	20	0	0	3.45	4.45	0.17
4x4x2	opposite	60	20	0	0	6.85	8.10	0.36
4x4x2	opposite	80	20	0	0	13.35	9323.25	353.96
4x4x2	single	40	20	0	0	8.70	106.05	0.73
4x4x2	single	60	20	0	0	16.95	22859.35	134.39
4x4x2	single	80	0	0	20	-	-	-
4x4x2	three	40	20	0	0	2.85	3.85	0.19
4x4x2	three	60	20	0	0	4.90	5.90	0.32
4x4x2	three	80	20	0	0	8.90	2047.85	104.70
5x5x2	corner	40	20	0	0	8.95	13.55	1.02
5x5x2	corner	60	12	0	8	15.25	3573.83	424.12
5x5x2	corner	80	0	0	20	-	-	-
5x5x2	four	40	20	0	0	4.85	5.85	1.20
5x5x2	four	60	20	0	0	7.65	8.65	2.67
5x5x2	four	80	15	0	5	12.93	701.00	130.92
5x5x2	opposite	40	20	0	0	6.45	7.45	0.75
5x5x2	opposite	60	20	0	0	11.15	454.05	64.23
5x5x2	opposite	80	8	0	12	18.25	12231.12	1176.80
5x5x2	single	40	20	0	0	13.25	1661.10	25.24
5x5x2	single	60	9	0	11	23.00	30404.56	423.61
5x5x2	single	80	0	0	20	-	-	-
5x5x2	three	40	20	0	0	5.75	6.75	0.96
5x5x2	three	60	20	0	0	10.10	69.00	16.11
5x5x2	three	80	10	0	10	15.60	5840.40	734.11

Table A.2 Results for the single bay UPMP with two tiers rounded to the second decimal